ТЕРНОПІЛЬСЬКА ДЕРЖАВНА МЕДИЧНА АКАДЕМІЯ
ІМ. І.Я. ГОРБАЧЕВСЬКОГО
КАФЕДРА МЕДИЧНОЇ ІНФОРМАТИКИ

В.П. Марценюк, А.В. Семенець

МЕДИЧНА ІНФОРМАТИКА

Інструментальні та експертні системи

Рекомендовано Центральним методичним кабінетом з вищої медичної освіти МОЗ України як навчальний посібник для студентів вищих медичних навчальних закладів III-IV рівнів акредитації

Тернопіль
Укрмедкнига
2003

ББК
М
УДК

Рецензенти:

Наконечний О.Г. — професор, завідувач кафедри системного аналізу та теорії прийняття рішень факультету кібернетики Київського національного університету ім. Т. Шевченка;

Боєчко В.Ф. — доцент, завідувач курсу біофізики Буковинської державної медичної академії;

Слєдзінський І.Ф. — професор Тернопільського державного педагогічного університету ім. В. Гнатюка;

Маланюк П.М. — доцент Тернопільського державного педагогічного університету ім. В. Гнатюка;

Марценюк .П. Семенець А..

М Інструментальні та експертні системи. — Тернопіль: Укрмедкнига, 2003. — 222 с.

ISBN 966-7364-88-7

Навчальний посібник висвітлює застосування важливого класу програмного забезпечення — інструментальних систем до розв'язання прикладних задач медицини. показано можливості об'єктно-орієнтованого підходу та візуального проектування на конкретних прикладах, які можуть бути використані при проведенні практичних занять з медичної інформатики, використовуючи надані лістинги програм. У посібнику також наведено відомості про інтелектуальні можливості комп'ютера при розробці експерних систем медико-діагностичного призначення.

Для студентів вищих медичних закладів, а також усіх зацікавлених розробкою програмного забезпечення в медицині.

ББК
УДК

ISBN

ЗМІСТ

Вступ

Прогресивний розвиток медичної науки і практики став можливим завдяки впровадженню сучасної комп'ютерної техніки. Персональні комп'ютери значною мірою організовують і вдосконалюють роботу лікаря, сприяють систематизації медичної інформації і розробці нових технологій у діагностичному і лікувальному процесах, організації діяльності медичних установ. Використання персональних комп'ютерів надає додаткові можливості в отриманні і передачі медичної інформації. Міжнародні комп'ютерні мережі, система комп'ютерних бібліотек дозволяють лікарю користуватися досягненнями світової медичної науки. Тому поряд з опануванням дисциплін медичного факультету майбутній лікар повинен досконало оволодіти комп'ютерною технікою, її програмним забезпеченням.

Даний навчальний посібник містить висвітлення двох взаємопов'язаних розділів, які відображені у назві навчального посібника – "Інструментальні та експертні системи".

Вивчення медичної інформатики передбачає опанування розділу "Застосування об'єктно-орієнтованого програмування для організації робочого місця лікаря". Його метою є оволодіння засобами об'єктно-орієнтованого програмування (ООП) в середовищі сучасних інструментальних систем. Користуючись засобами ООП, лікар має можливості налаштувати власний комп'ютер на виконання програм, які найчастіше використовуються; здійснювати безпосередній доступ до даних, що створені з використанням найрізноманітніших систем управління базами даних і розміщені на локальному чи віддаленому комп'ютерах тощо.

Далі наведемо формулювання основних практичних задач, що стоять при вивченні даного розділу:

- на основі готових відеофрагментів вміти розробити медіаплеєр для програвання записів хірургічних операцій та маніпуляцій;
- користуючись бібліотекою медичних зображень, розробити програму-переглядач медичних слайдів;
- здійснити комп'ютерне моделювання динамічних систем в медицині з використанням Internet-програмування.

Медична інформатика включає також розділ "Експертні системи в медицині". Експертні системи допомагають прийняти рішення лікареві в ситуаціях, які вимагають значного розумового напруження і обмежені в часі. Найчастіше це — постановка діагнозу хвороби, диференціальна діагностика, вибір тактики лікування. Вагоме місце експертні системи займають в діяльності середнього медичного персоналу, коли рішення необхідно приймати в екстремальних умовах при відсутності лікаря. Крім цього, експертні системи допомагають оптимізувати організацію системи охорони здоров'я. Вони будуються для розв'язання конкретних задач за певними попередньо перевіреними правилами.

Метою даного розділу є вивчення теорії прийняття рішень в медицині, набуття навичок побудови експертних систем та роботи з ними. Це дозволить лікареві вдосконалювати свою майстерність, об'єктивізувати прийняття рішень, тиражувати свій інтелект (у вигляді комп'ютерних програм).

Формулювання основних практичних задач, що стоять при вивченні даного розділу:

- користуючись базою даних медико-діагностичного призначення, розробити експертну систему на основі логічної схеми Байєсівського виведення;
- користуючись базою медичних знань, розробити експертну систему на основі генетичного алгоритму та Internet-програмування.

Зазначимо, що для розв'язування даних задач в посібнику та додатках запропоновано відкриті програмні модулі.

ОГЛЯД СУЧАСНИХ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ПРОЕКТУВАННЯ ТА ЇХ ВИКОРИСТАННЯ У МЕДИЦИНІ

Під **інструментальними системами** мається на увазі категорія програмного забезпечення комп'ютерів, що використовується для створення нових програм. Для сучасних інструментальних систем використовуються тотожні назви: системи програмування, системи розробки. На сьогоднішній день нове програмне забезпечення розробляється при розв'язуванні задач вузьких предметних областей, які поки що не мають загальновизнаного комп'ютерного вирішення. Широке поле для застосування інструментальних систем пропонують задачі обробки медичної інформації, які розглядатимуться далі: задачі побудови мультимедіа-додатків в медицині, задачі моделювання перебігу захворювань та поведінки систем організму, розробка експертних систем медико-діагностичного призначення, задачі цифрової обробки графічної медичної інформації.

Основні етапи розвитку інструментальних систем

Сучасні інструментальні системи є нащадками найдавнішого програмного забезпечення електронно-обчислювальних машин (ЕОМ). З цього приводу наведемо короткий історичний аналіз.

Керування роботою перших ЕОМ здійснювалося за допомогою **машинного коду**. Це означає, що людина — висококваліфікований спеціаліст (найчастіше розробник ЕОМ) вводив за допомогою пристроїв вводу (далеких від теперішньої клавіатури та мишки) послідовності символів "0" та "1". Кожна така послідовність позначала певну команду процесора. Програмування ЕОМ в машинному коді було виснажливою роботою.

На початку 50-х років з'явилися перші транслятори (програми-перекладачі) мови програмування Асемблер в машинний код. Мова Асемблер по сьогоднішній день використовується спеціалістами в галузі системного програмування і зустрічається під назвами: машинно-орієнтована мова програмування, мова програмування низького рівня. Ідея мови Асемблер полягає у заміні команд

процесора (вже згаданих послідовностей з "0" та "1") так званими **мнемокодами** (деякими їх символічними позначеннями). При програмуванні на Асемблер вимагаються досконале володіння командами мікропроцесора, знання його структури та порядку роботи. До основних недаліків Асемблер належить його прив'язаність до конкретного типу мікропроцесора, що часто створює труднощі із розповсюдженням (перенесенням) готових програм на інші типи комп'ютерів. Сьогодні Асемблер рідко використовується для написання повних програмних модулів. Його використовують програмісти-фахівці для оптимізуючих перетворень найбільш напружених ділянок програм. Про необхідність Асемблера свідчить той факт, що команди для її підтримки увійшли в найсучасніші інструментальні системи. Мова Асемблер використовується у програмному забезпеченні для медичної апаратури.

Приклад програмування на Асемблері, що здійснює переміщення рядка символів з одного місця пам'яті в інше:

```
        cld
Repeat:
        lodsw
        cmp ax, 0FFFFh
        je Exit
        stosw
        jmp Repeat
Exit:
```

У 1957 році відбулася важлива подія — поява першої мови програмування високого рівня Fortran (від англійського **For**mula **Tran**slation). Fortran містив команди, що значно спрощували виконання на ЕОМ математичних обчислень. Для програмування на Fortran не вимагалося глибокого знання внутрішньої будови комп'ютера.

1960 рік пов'язаний з реалізацією мови програмування Algol-60. При розробці Algol вперше було приділено увагу методологічним принципам програмування таким, як структурний підхід та технологія розробки програми "зверху-вниз", що передбачає поетапну її деталізацію.

1967 рік. В Норвегії розроблено мову програмування Simula-67, в якій реалізовано об'єктно-орієнтований підхід. Її призначення — моделювання поведінки реальних систем та явищ. У подальшому об'єктно-орієнтований підхід дав можливість створити операційну оболонку з графічним інтерфейсом Windows.

На початку 70-х доступний синтаксис Algol-60 був використаний для мови програмування Pascal. У Pascal вдалося реалізувати ряд важливих типів даних: множина, запис, порядкові типи даних споживача, дані з динамічною (змінною з часом) структурою. З Pascal пов'язано крилатий вираз одного із його розробників Ніклауса Вірта: "Алгоритми + структури даних = програми".

У той же час з'явилася мова програмування С (англійське "Сі"), що мала дуже дивний синтаксис. У ній вдалося одночасно втілити риси мови програмування високого рівня та машинно-орієнтованої мови. С була використана при створенні все ще популярної операційної системи UNIX. На сьогоднішній день С разом з мовою Асемблер використовується при розробці найважливіших системних програм.

70-80-ті роки пов'язані з розробкою перших популярних прикладних програм — текстовий редактор, табличний процесор, програми для роботи з графікою, системи управління базами даних. З'явилася можливість залучити до роботи із комп'ютером широке коло людей-непрофесіоналів комп'ютерної галузі. Слід мати на увазі, що всі ці програми вдалося створити, використовуючи розвинуті інструментальні засоби тодішніх мов програмування.

У 80-ті роки здійснюється поширення об'єктно-орієнтованого підходу до мови програмування. Найповнішою мірою він був втілений в C++ (читається "Сі-плюс-плюс") та Borland Pascal.

90-ті роки пов'язані з візуалізацією засобів створення програм. Відтепер нащадки мов програмування стали називатися системами розробки (development system), а з мовою програмування, перш за все, пов'язують синтаксис команд інструментальних систем. Першим програмним продуктом з можливістю візуального проектування став Visual Basic фірми Microsoft. Та лише в системі Borland Delphi так званий механізм Two-way tools (передбачає можливість візуального створення програми з прототипів) було доведено до кінця. Усі сучасні інструментальні системи (Visual Basic, Visual C++, Delphi,

C++Builder, Jbuilder та ін.) надають можливість візуального проектування та втілюють об'єктно-орієнтований підхід.

І, нарешті, останні події в галузі інструментальних засобів пов'язані з так званими мовами програмування Web-сторінок: JavaScript та VBScript та мовами побудови сценаріїв: PERL, REXX. Їх призначення — написання програм, які дозволяють "оживити" Web-сторінки Internet. Особливістю згаданих мов є можливість виконання їх команд на будь-якому комп'ютері, не використовуючи при цьому виконуваного файла та не вимагаючи інсталяції у ньому відповідної інструментальної системи. Для роботи з медичною інформацією у США спеціально розроблено мову сценаріїв MERL (Medical Retrieval Language), що працює з інтерфейсом CGI (Common Cateway Interface — стандарт інтерфейсу для роботи в Internet). Багато програмістів-професіоналів пов'язує з мовами сценаріїв майбутнє розвитку інструментальних засобів.

Можливості сучасних інструментальних систем у розв'язанні прикладних задач медицини

Наведемо основні категорії додатків, що реалізуються інструментальними системами. У дужках вказані практичні задачі, для яких буде в подальшому запропоновано розв'язок.

1. Додатки баз даних.
2. Мультимедіа-додатки (програвач записів хірургічних операцій).
3. Моделювання динамічних систем (перебіг інфекційного захворювання, моделювання серцево-судинної системи).
4. Розробка експертних систем (експертна система медико-діагностичного призначення, експертна система захворювань органів черевної порожнини).
5. Розробка додатків, що використовують технології обміну даними DDE, OLE.
6. Розробка Internet-додатків.
7. MDI-додатки.
8. Додатки для роботи з графічною інформацією (переглядач медичних слайдів).
9. Додатки для роботи з принтером.

Визначальні характеристики інструментальних систем (на прикладі Delphi)

Компілятор в машинний код, розроблений для Delphi, є одним з найпотужніших серед систем програмування. Він дозволяє обробляти 120 тисяч рядків вхідного коду за хвилину на процесорі 486DX33.

Об'єктно-орієнтована модель програмних компонент. У Delphi вперше повністю реалізовано об'єктно-орієнтовану модель компонент. У стандартну редакцію Delphi входять основні об'єкти, що утворюють ієрархію із 270 базових класів.

Швидка розробка додатка на основі прототипів. Дуже велике число додатків можна розробити, користуючись лише компонентами Delphi, розміщуючи їх у форму та змінюючи властивості.

Масштабовані засоби побудови баз даних. Один і той же додаток можна використати як для локального, так і для клієнт-серверного варіантів. У цьому проявляється гнучкість компонент Delphi для роботи з базами даних.

Існує два варіанти поставки Delhi: Delphi Desktop та Delphi Client-Server. У Delphi Desktop як і в Delphi Client-Server входять:

- компілятор Object Pascal;
- генератор звітів Report Smith;
- середовище візуальної побудови додатків;
- бібліотека візуальних компонент VCL;
- локальний сервер Interbase.

Delphi розроблена на Delphi. Довершеністю будь-якої системи програмування є той факт, що її можна розробити, користуючись нею ж самою. Дана можливість була також перевірена на Delphi.

Відкрита компонентна архітектура. На відміну від багатьох систем програмування компоненти Delphi є відкритими, тобто поставляються з вихідним кодом та реалізовані на самій же Delphi.

Two-way tools — однозначна відповідність між візуальним проектуванням і класичним написанням тексту програми. Ця можливість чи не найперше також була реалізована в Delphi.

Підтримка OLE, DDE та VBX. Це новітні технології обміну даними між додатками.

Середовище розробки, яке налаштовується. Середовище розробки Delphi має багато можливостей налаштування для більш зручної роботи розробника.

Інтелектуальний редактор текстів програм. Редактор текстів програм у Delphi має вмонтовану функцію підказування про типи параметрів функцій чи процедур, про властивості чи методи об'єктів.

Графічний відлагоджувач. Відлагоджувач Delphi дозволяє візуально розставляти точки зупинки, відображати в окремому вікні значення змінних, що досліджуються.

Інспектор об'єктів. Інспектор об'єктів в зручній формі дозволяє переглядати список властивостей та подій компонент, їх значення та змінювати ці значення.

Менеджер проектів. Він дозволяє переглядати модулі проекту, додавати нові чи відключати колишні.

Навігатор об'єктів. Дозволяє переглядати список об'єктів, включених у проект.

Дизайнер меню. Це спеціальна підпрограма, що дозволяє у візуальній формі проектувати команди меню.

Експерти. Під експертами мають на увазі набір програм, що полегшують проектування та налаштування ваших додатків.

Компоненти доступу до баз даних. Вони роблять додатки для баз даних в Delphi масштабованими.

Відповідність між модулями та формами на основі методу розробки "Two-Way Tools".

Делегування і події програмуються простіше. Делегування — це принцип, що дозволяє об'єкту передавати повноваження іншому об'єкту відповідати на подію.

Посилання на класи під час виконання програми. Дуже часто у програмуванні подій виникає ситуація, коли наперед невідомо, який об'єкт пошле запит на виконання обробника події. В Delphi таке уточнення можна зробити під час виконання програми.

Обробка виняткових ситуацій. Виняткові ситуації (наприклад, ділення на нуль) часто виникають під час виконання складних додатків. Delphi має зручні механізми для розв'язування такого роду проблем.

СУЧАСНИЙ СТАН ІНДУСТРІЇ ІНСТРУМЕНТАЛЬНИХ СИСТЕМ

Інструментальні системи минулих десятиліть

20 років тому поняття "інструментальні системи розробки програмного забезпечення" асоціювалося з пакетним компілятором для командної лінійки і (в кращому випадку) набором додаткових утиліт та деякими прикладами коду. Програміст при розробці нової системи повинен був виконати величезний об'єм роботи — від розробки структури алгоритму та його реалізації на відповідній мові до проектування інтерфейсу програми та її налагодження.

10 років тому інструментальні системи (ІС) вже являли собою потужні інтегровані середовища розробки програм, що включали спеціалізовані текстові редактори, вбудовані компілятори, пристойний комплект утиліт для тестування і налагодження коду програми, а також добавлення спеціальних можливостей (наприклад, робота з нестандартними зовнішніми пристроями), розвиненою системою допомоги. Крім того, програмісту пропонувалася велика кількість готових прикладів програмного коду для вирішення певних задач, і більше того — розвинені бібліотеки готових функцій, що можна було підключати залежно від потреби.

В цей час на ринку панували такі програмні пакети, як Turbo C, C++ Turbo Pascal від компаній Microsoft та Borland. Активно розвивався провідний на сьогодні напрямок — об'єктно-орієнтоване програмування — було розроблено популярні бібліотеки Object Pascal, TurboVision та інші, що давали можливість розробляти потужні програмні пакети з меншими затратами праці.

Наступний крок на ринку інструментальних систем настав з виходом систем програмування для операційної системи (ОС) Windows — Turbo Pascal for Windows та Turbo C for Windows від фірми Borland, а також Quick C Microsoft, що сталося на початку 90-х років.

Початок сучасного етапу розвитку інструментальних систем

Початок сучасного етапу розвитку ІС було покладено у 1992 р. — фірма BORLAND випустила першу розроблену повністю для програмування під Windows систему DELPHI 1.0, за допомогою якої можна було створювати Windows-додатки. Незабаром фірма Microsoft випустила пакет Microsoft Visual Studio, який включав системи розробки Windows-додатків Visual Basic for Application та Visual C++. Тепер програми можна було буквально "збирати" з готових елементів інтерфейсу та фрагментів коду.

Незважаючи на посилену конкуренцію, флагманами в галузі ІС-систем залишаються компанії Borland (тепер Inprise) та Microsoft. Регулярно виходять обновлені версії їх ІС DELPHI та Visual Studio, поповнюючись все новими і новими можливостями та функціями.

Єдиним класичним завданням до недавнього часу була розробка Windows-додатків для роботи з базами даних різного типу (локальні, клієнт-сервер). Всі традиційні інструментальні системи і були спрямовані на вирішення цього завдання.

В 1989 році почалася розробка мови програмування Python. Python — потужна інтерпретована, інтерактивна, об'єктно-орієнтована мова програмування з простим синтаксисом. Інтерпретатор Python можна легко вбудувати в будь-яку інструментальну систему. Наприклад існують реалізації для Delphi (PythonForDelphi) WinAPI (PythonWin), Java (JPython). Для розширення функціональних можливостей мови Python розроблено доволі багато стандартних бібліотек (для обробки графіки, набір математичних функцій, інтерфейс доступу до баз даних, платформонезалежний графічний інтерфейс, WinAPI-інтерфейс і т.п.). Це дозволяє легко створити додаток, що поєднує несумісні програмні продукти — наприклад, на основі бази даних виконує статистичний аналіз. Незважаючи на відносну молодість, дана мова програмування набуває все більшої популярності.

Орієнтація на універсальні технології доступу до даних

На початку 90-х рр. спеціалістами Microsoft був розроблений Open DataBase Connectivity (ODBC) — перший універсальний прикладний інтерфейс програмування доступу до баз даних клієнт/сервер, що базується на моделі Data Access Object (DAO — об'єкти доступу до даних). Сам інтерфейс написаний на мові C/C++. На сьогоднішній день він є де-факто стандартом доступу до баз даних різного типу. Всі сучасні інструментальні системи підтримують технологію ODBC.

Починаючи з 1998 року, компанія Microsoft представляє модель Component Object Model (COM) як основу для розробки всіх Windows-додатків, і особливо проектів клієнт-сервер. Дана технологія визначає набір інтерфейсів для створення об'єктів (компонентів) будь-якої складності з можливістю їх повторного використання. Операційні системи Windows 9x та Windows NT 4+ Windows 2000 підтримують розподілену модель DCOM (Distributed COM) — механізм взаємодії кількох об'єктів COM, розміщених на комп'ютерах, що об'єднані в мережу.

На моделі COM та ActiveX Data Objects (ADO) — об'єктах даних ActiveX — базується OLE DB — стратегічна технологія доступу до даних компанії Microsoft, призначена для локальних баз даних та баз даних клієнт/сервер. Основу даної технології складають три елементи: провайдери, споживачі та служби даних.

Головними конкурентами запропонованої Microsoft технології є:

- технологія Common Object Reqrest Broker Architecture (CORBA) — загальна архітектура брокерських запитів, що просувається постачальниками інструментальних систем для ОС Unix (зокрема, компанією IBM);
- технологія Enterprise JavaBeans (EJB) від компанії SunMicrosystems, написана на мові Java.

Інструментальні системи третього тисячоліття

У червні 2000 року компанія Microsoft випустила нову реалізацію мови C — Microsoft C#. В даному релізі велика увага

звертається на створення Web-сервісів для платформи Microsoft .NET, для чого введена підтримка процесоронезалежної мови Intermediate Language (IL). Завдяки цьому на Microsoft C# можна писати програми для будь якої .NET-платформи, від Windows 2000 до Windows CE.

В Microsoft C# поєднано простоту і виразність сучасних об'єктно-орієнтованих мов (Java) з широкими можливостями і потужністю C++. Суттєво покращено, порівняно з C++, механізм управління пам'яттю ПК. Введено підтримку інтерфейсів (як в Java). Дана мова програмування включає повноцінну підтримку технології COM+ Windows API, а також можливість взаємодіяти з іншими бібліотеками для платформи Win32. Залишається очікувати появи повноцінного інтегрованого середовища розробки додатків на основі Microsoft C#, що буде зроблено фірмою Microsoft у наступній версії комплекту ПЗ для розробників — Microsoft Visual Studio.NET.

Протягом 90-х рр. компанія Borland постійно модернізує інструментальну систему DELPHI, випускаючи все нові і нові версії, що включають все більші можливості щодо розробки Windows-додатків, активно використовуючи вказані найновіші технології Microsoft.

В травні 2001 року компанія Borland випустила нову, 6-ту версію середовища швидкої розробки додатків DELPHI. Серед нових можливостей в офіційному прес-релізі описуються бібліотеки компонентів: BizSnap — спрощує створення Web-сервісів для таких платформ, як Microsoft .NET BizTalk і SunONE від SunMicrosystems. Наступна нова бібліотека компоненів — WebSnap, що дозволяє проводити інтеграцію з сайтами, що створені за допомогою найбільш поширених Web-редакторів таких, як Dreamweaver та FrontPage. І ще одна важлива нова бібліотека — DataSnap, призначена для розробки масштабованих додатків типу клієнт-сервер з застосуванням основних промислових стандартів розподіленої обробки інформації — SOAP/XML, DCOM, TCP/IP CORBA.

Одночасно з анонсом DELPHI 6 була представлена аналогічна система Kylix для операційної системи Linux. Поєднання цих двох інструментальних систем дозволяє створювати крос-плат-

формові додатки, що можуть працювати під управлінням як ОС Windows, так і ОС Linux.

За останнє десятиліття об'єктно-орієнтована мова програмування Java набула великої популярності і зараз займає місце на рівні з Object C++ (Visual C++) та Object Pascal (Delphi). Це сталося завдяки високій надійності та захищеності програм, написаних на мові Java, що має велике значення для корпоративних клієнтів.

В червні 2001 компанія BORLAND анонсувала ще одну новинку — крос-платформове середовище розробки на мові Java для систем Windows, Linux і Solaris — інструментальну систему JAVABUILDER 5. В даний продукт додана підтримка сервера додатків IBM WebSphere, покращені засоби взаємодії з BEA WebLogic 6 і Borland App Server 4.5, а також багато інших дрібних модернізацій і додаткових бібліотек компонентів.

Інструментальні системи Internet-програмування

З середини 90-х рр. при розробці програмного забезпечення великого значення набула підтримка роботи в Internet. Сформувався цілий клас мов програмування, що призначені лише для розробки Web-сторінок та створення Web-серверів. В першу чергу, це HTML (HyperText Markup Language — мова гіпертекстової розмітки), DHTML (Dynamic HTML), XML (Extensible Markup Language — мова інтенсивної розмітки). Windows-додатки, написані за допомогою мов такого типу, виконуються броузером Internet (наприклад, Microsoft Internet Explorer).

Інструментальні системи для розробки ПЗ за допомогою вказаних мов вражають різноманіттям. Найпростіший (але не найслабший!) з них — редактор Notepad з комплекту Windows, за допомогою якого здійснюється написання коду "з нуля". Найбільш поширені зараз такі Web-редактори, як Dreamweaver, FrontPage, HomeSite. Ці системи являють собою класичні інтегровані середовища розробки — текстові редактори з можливістю попереднього перегляду Web-сторінки, перегляду її структури, а також додаткові утиліти для відлагодження коду.

Окремо слід виділити Java Script — мова Web-програмування на основі Java. Фрагменти коду Java Script можна встав-

ляти в будь-яку HTML-сторінку. Як правило, на Java Script описують окремі інтерфейсні елементи Web-сторінки.

Слід зазначити, що навіть виробники офісних програм все більше орієнтуються на Internet-технології. Починаючи з версії Microsoft Office 2000, пропонується єдиний стандарт зберігання документів — формат HTML.

Крім описаних вище інструментальних систем розробки ПЗ, існує велика кількість менших, які, однак, володіють потужною функціональністю.

Висновки

Підводячи підсумок, можна сказати, що за останнє десятиліття індустрія інструментальних систем набула великого піднесення. Причому за останні 5 років відбулося зміщення інтересів розробників у бік інструментальних систем, орієнтованих на створення додатків, що працюють, використовуючи Internet-технології для доступу та модифікації даних.

Запитання

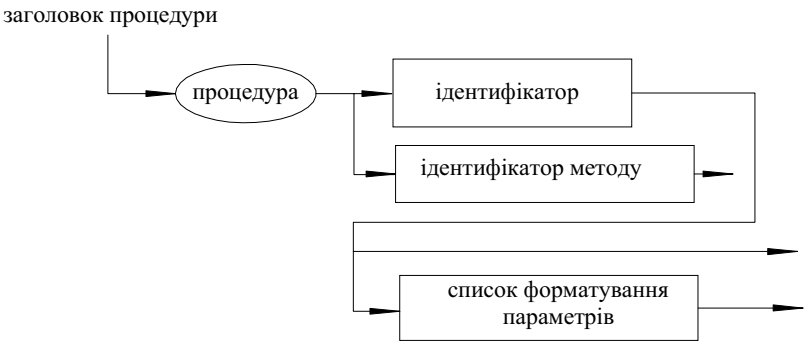
1. Поняття про інструментальні системи Internet-програмування.
2. Новинки індустрії інструментальних систем.
3. Які є основні універсальні технології доступу до даних?
4. Еволюція інструментальних систем за минулі десятиліття.

ОСНОВНІ СИНТАКСИЧНІ КОНСТРУКЦІЇ МОВИ ПРОГРАМУВАННЯ OBJECT PASCAL

Далі буде показано, що робота в популярній інструментальній системі Delphi — не лише налаштування властивостей візуальних компонент, а дещо більше. Для цього потрібно заглибитися в саме ядро Delphi — мову програмування Object Pascal.

Синтаксичні діаграми

Для опису мови програмування Object Pascal користуватимемося **синтаксичними діаграмами**. Наприклад:



Щоб прочитати синтаксичну діаграму, потрібно слідувати за стрілками. Дуже часто буває так, що можливий більше як один шлях. Подана вище діаграма показує, що список формальних параметрів необов'язковий у заголовку процедури. Можна йти від ідентифікатора до кінця заголовка процедури, або ж можна йти до списку формальних праметрів перед тим, як досягнути кінця діаграми.

Надписи у прямокутниках вказують на синтаксичні конструкції. Надписи в овалах — зарезервовані слова. Оператори та знаки пунктуації — це дійсні терми з програм; вони в діаграмах виділені напівтовстим шрифтом.

Токени

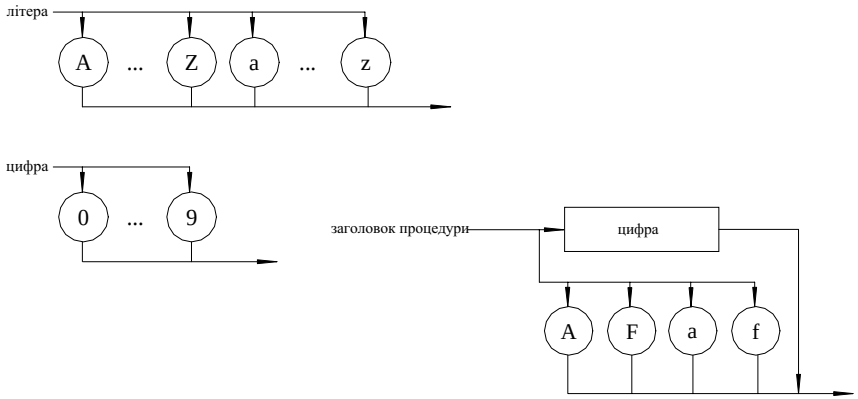
Токени — це найменші значащі одиниці тексту в програмі Object Pascal. Вони поділяються на символи, ідентифікатори, мітки, числа і рядкові константи. Програма на Object Pascal складається з токенів та сепараторів. **Сепаратор —** це або пропуск або коментар. Два суміжні токени повинні бути розділені одним або більше сепараторами, якщо кожен токен є зарезервованим словом, ідентифікатором, міткою або числом. Сепаратори не можуть бути частинами токенів, за винятком рядкових констант.

Спеціальні символи

Object Pascal використовує такі підмножини набору символів ASCII:

- ***Літери*** *— англійський алфавіт від* A *до* Z *і* a *до* z.
- ***Цифри —*** арабські від 0 до 9.
- ***Шістнадцяткові цифри*** *— арабські цифри від 0 до 9, букви від* A *до* F *або від* a *до* f.
- ***Пропуски —*** символ пропуску (ASCII 32) і всі керуючі символи ASCII (від ASCII 0 до 31), включаючи символи кінця рядка або повернення каретки (ASCII 13).

Далі наводяться синтаксичні діаграми букви, цифри та шістнадцятирічної цифри:



Спеціальні символи та зарезервовані слова — це символи, що мають одне або більше фіксованих значень. Наступні одиничні символи є спеціальними символами:

+ — * / = < > [ ] . , ( ) : ; ^ @ { } $ #

І ці пари символів є також спеціальними символами:

<= >= := .. (* *) (. .)

Ліва квадратна дужка (**[**) є еквівалентна парі символів лівої круглої дужки і крапки (**(.**, а символ (**]**) еквівалентний **.)**. Подібно символ (**{**) еквівалентний **(***, а (**}**) — ***)**.

Зарезервовані слова та стандартні директиви

Зарезервовані слова не можуть переозначуватися. Тут вони позначатимуться напівтовстим шрифтом і малими літерами. Object Pascal не є чутливим до регістру символів (великі-малі букви), отже, можна використовувати у ваших програмах як великі, так і малі букви.

Далі наводяться зарезервовані слова Obejct Pascal.

Таблиця 1-1. Зарезервовані слова Object Pascal

and	exports	library	set
array	file	mod	shl
as	finally	nil	shr
asm	for	not	string
begin	function	object	then
case	goto	of	to
class	if	on	try
const	implementation	or	type
constructor	in	packed	unit
destructor	inherited	procedure	until
div	initialization	program	uses
do	inline	property	var
downto	interface	raise	while
else	is	record	with
end	label	repeat	xor
except			

Далі наведені стандартні директиви Object Pascal. Директиви використовуються лише в місцях, де означені споживачем ідентифікатори не мають сили. На відміну від зарезервованих

слів стандартні директиви можна переозначати, хоча робити це не рекомендується.

Таблиця 1-2. Директиви Object Pascal

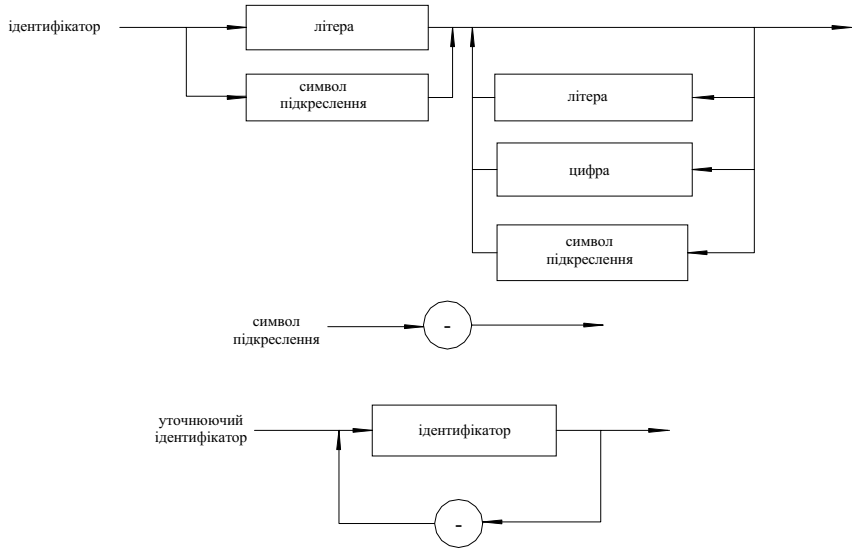
absolute	export	name
published	abstract	external
near	read	assembler
far	nodefault	resident
at	forward	override
stored	cdec	l
index	private	virtual
default	interrupt	protected
write	dynamic	message
public		

private, **protected**, **public** і **published** діють як зарезервовані слова в межах декларації типу об'єкта та порядок їх використання відрізняється від директив.

Ідентифікатори

Ідентифікатори позначають константи, типи, змінні, процедури, функції, модулі, програми та поля в записах.

Ідентифікатор може бути довільної довжини та лише перші 63 символи є значущими. Ідентифікатор повинен починатися з букви або символу підкреслення (_) і не може включати пропусків. Після першого символу дозволяється використовувати букви, цифри та символи підкреслення (ASCII $5F). Подібно до зарезервованих слів ідентифікатори не є залежними від регістру символу. Коли існує кілька інстанцій того ж ідентифікатора, буває потрібно кваліфікувати ідентифікатор іншим ідентифікатором, щоб вибрати певну інстанцію. Наприклад, щоб кваліфікувати ідентифікатор *Ident* ідентифікатором модуля *UnitName*, пишемо *UnitName.Ident*. Комбінований ідентифікатор називається **уточнюючим ідентифікатором**.



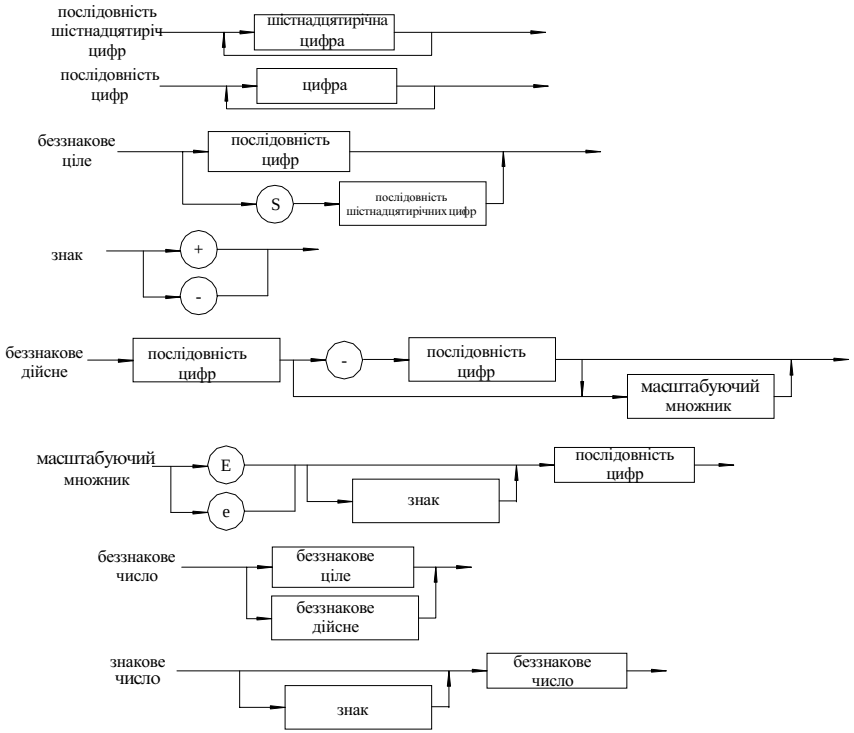
Приклади ідентифікаторів звичайних та уточнюючих:

Writeln
Exit
Real2String
System.MemAvail
SysUtils.StrLen
WinCrt.ReadText

Числа

Для чисел, що є константами цілочисельного або дійсного типів, використовується звичайне десяткове позначення. Шістнадцятирічні цілочисельні константи використовують в якості префікса знак долара ($). Інженерне позначення (E або e, за яким стоїть степінь) читається як "помножити на десять в степені" дійсного числа. Наприклад, 7E-2 означає 7×10^{-2}; 12.25e+6 або ж 12.25e6 означають 12.25×10^{6}.

Нижче наведено синтаксичні діаграми для написання чисел.



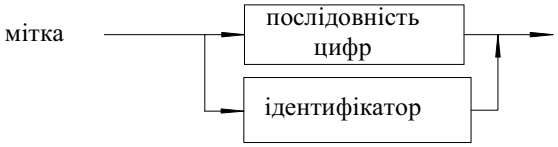
Числа з десятками або степенями позначають константи дійсного типу. Інші десяткові числа позначають константи цілочисельного типу; вони повинні бути в діапазоні від -2,147,483,648 до 2,147,483,647.

Шістнадцятирічні числа позначають цілочисельні константи; вони повинні бути в діапазоні від

$00000000 до $FFFFFFFF.

Мітки

Мітка — це послідовність цифр в діапазоні від 0 до 9999. Перші нулі не сприймаються. Мітки використовуються у реченнях **goto.**



В розширення до стандартного Pascal, Object Pascal допускає ідентифікатори функцій — мітки.

Рядки символів

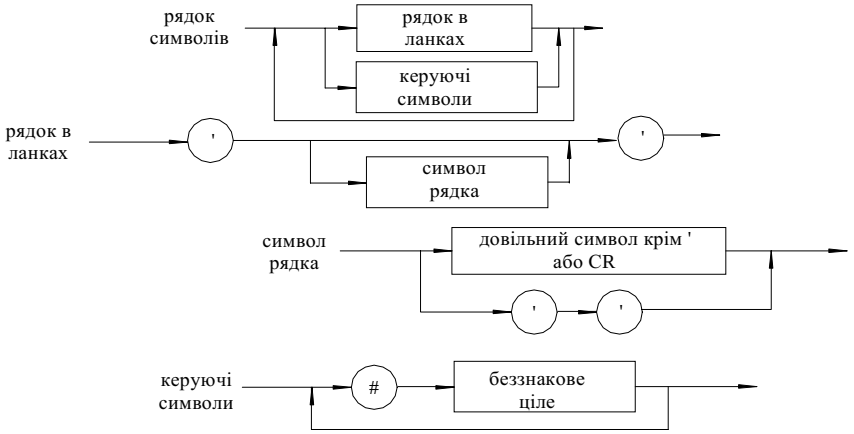
Рядок символів — це послідовність символів з розширеного набору символів ASCII, записана в одному рядку програми та заключена в апострофи.

Рядок символів, що нічого не містить між апострофами є **нульовим рядком**. Два послідовні апострофи в рядку символів позначають один символ апострофа. Наприклад:

```
'BORLAND' { BORLAND }
'You''ll see' { You'll see }
'''' { ' }
'' { null string }
' ' { a space }
```

Object Pascal дозволяє впроваджувати символи керування у рядок символів. Символ #, за яким слідує беззнакова цілочисельна константа в діапазоні від 0 до 255, позначає символ із відповідним значенням коду ASCII. Наприклад:

```
#13#10
'Line 1'#13'Line2'
#7#7'Wake up!'#7#7
```



Коментарі

Наступні конструкції називаються коментарями. Вони ігноруються програмою-компілятором:

```
{ Any text not containing right brace }
(* Any text not containing star/right parenthesis *)
```

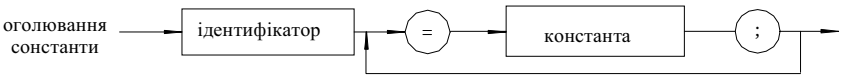
Коментар, що містить знак долара **($)** відразу після відкриваючих символів **{** або **(*** є **директивою компілятора**. Далі за символом **$** слідує мнемокод команди компілятора.

Рядки програми

Рядки програми на Object Pascal містять до 126 символів.

Константи

Константа — це ідентифікатор, що позначає значення, що не може змінюватися. **Декларація (оголошення) константи** оголошує константу в межах блоку, що включає декларацію. Ідентифікатор константи не може використовуватися у її ж власній декларації.

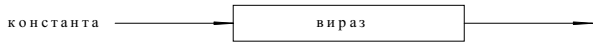


Object Pascal дозволяє використовувати **константні вирази**. Константний вираз — це вираз, що може бути оцінений, не виконуючи насправді програми. В той час, як стандартний Pascal дозволяє лише прості константи, Object Pascal дозволяє константні вирази.

Нижче показані приклади константних виразів:

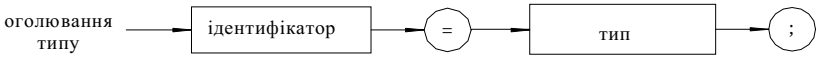
```
100
'A'x
256 — 1
(2.5 + 1) / (2.5 — 1)
'Borland' + ' ' + 'Pascal'
Chr(32)
Ord('Z') — Ord('A') + 1
```

Найпростіший приклад константного виразу — це проста константа, така як 100 або 'A'.

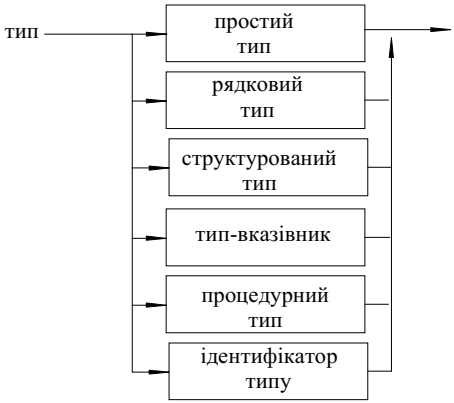


Типи

При оголошенні змінної необхідно вказувати її **тип**. Тип змінної визначає множину значень, що вона може приймати, та на операції, що може над нею виконуватися. Декларація (оголошення) типу визначає ідентифікатор, що позначає тип.



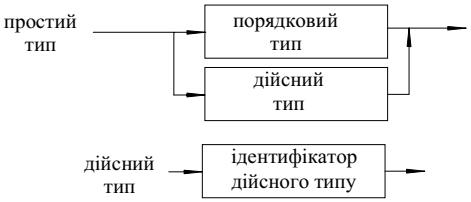
Ідентифікатор в лівій частині декларації типу є **ідентифікатором типу**. Ідентифікатор типу може використовуватися повторно в декларації лише для типів-покажчиків.



Існує шість головних класів типів. На них ми зупинимося далі.

Прості типи

Прості типи позначають впорядковані множини значень.



Ідентифікатор типу real є одним із стандартних ідентифікаторів: *Real, Single, Double, Extended* або *Comp*.

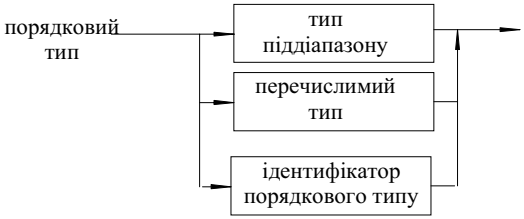
Порядкові типи

Порядкові типи є підмножиною простих типів. Усі прості типи, окрім типів дійсних чисел, є порядковими. Їм притаманні такі шість рис:

- Усі можливі значення заданого порядкового типу є впорядкованими множинами і кожне можливе значення має порядковий номер, що є цілочисельним значенням.

- Стандартна функція *Ord* може бути застосована до довільного значення порядкового типу з метою повернення порядкового номера значення.
- Стандартна функція *Pred* може бути застосована до довільного значення порядкового типу з метою отримання попередника цього значення.
- Стандартна функція *Succ* може бути застосована до довільного значення порядкового типу з метою отримання наступника цього значення.
- Стандартна функція *Low* може застосовуватися до ідентифікатора порядкового типу і до посилання змінної порядкового типу. Результатом є найнижче значення діапазону заданого порядкового типу.
- Стандартна функція *High* може застосовуватися до ідентифікатора порядкового типу і до посилання змінної порядкового типу. Результатом є найвище значення діапазону заданого порядкового типу.

Синтаксис порядкового типу такий:



Object Pascal має дванадцять попередньо означених порядкових типів: Integer, *Shortint, Smallint, Longint, Byte, Word, Cardinal, Boolean, ByteBool, WordBool, LongBool і Char*. Існує ще два інші класи порядкових типів, означених споживачем: перечислимі типи та типи піддіапазонів.

Цілочисельні типи

Наперед задані цілочисельні типи Object Pascal діляться на дві категорії: **фундаментальні** та **породжені** типи. Діапазон та формат фундаментальних типів не залежить від мікропроце-

сора та операційної системи і не відрізняється в різних реалізаціях Object Pascal. Діапазон та формат породжених типів, навпаки, залежить від мікропроцесора та операційної системи.

Фундаментальними типами є *Shortint, Smallint, Longint, Byte* і *Word*.

Таблиця 3-1. Фундаментальні цілочисельні типи

Тип	Діапазон	Формат
Shortint	-128..127	Signed 8-bit
Smallint	-32768..32767	Signed 16-bit
Longint	-2147483648..2147483647	Signed 32-bit
Byte	0..255	Unsigned 8-bit
Word	0..65535	Unsigned 16-bit

Породженими цілочисельними типами є *Integer* та *Cardinal*. Тип *Integer* представляє знакові цілі, а тип *Cardinal* — беззнакові цілі.

Таблиця 3-2. Породжені цілочисельні типи для 16-бітової реалізації Object Pascal

Тип	Діапазон	Формат
Integer	-32768..32767	Signed 16-bit
Cardinal	0..65535	Unsigned 16-bit

Таблиця 3-3. Породжені цілочисельні типи для 32-бітової реалізації Object Pascal

Тип	Діапазон	Формат
Integer	-2147483648..2147483647	Signed 32-bit
Cardinal	0..2147483647	Unsigned 32-bit

Додаткам слід використовувати породжені типи наскільки це можливо, оскільки це покращує продуктивність процесора та операційної системи. Фундаментальні типи слід використовувати лише там, де істотний дійсний діапазон та формат зберігання.

Булевські типи

Існує чотири наперед означені булевські типи: Boolean, ByteBool, WordBool *і* LongBool. *Булевські значення позначаються наперед означеними ідентифікаторами констант* False *та* True.

Оскільки булевські типи є перечислимими, то мають місце відношення:

- False < True;
- Ord(False) = *0;*
- Ord(True) = *1;*
- Succ(False) = True;
- Pred(True) = False.

Змінні типів *Boolean* та *ByteBool* займають один байт, *WordBool* — 2 байти (одне слово), *LongBool* — 4 байти (два слова). Необхідно віддавати перевагу типу *Boolean*. Типи *ByteBool, WordBool,* і *LongBool* існують, щоб гарантувати сумісність з іншими мовами та серидовищем *Windows*.

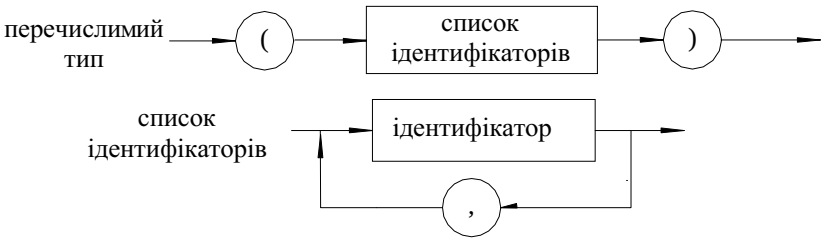
Символьний тип

Множина значень типу *Char* — це символи, впорядковані згідно з таблицею символів розширеного *ASCII*. Виклик функції *Ord(Ch)*, де *Ch* — це значення типу *Char*, повертає порядковий номер *Ch*.

Рядкова константа довжини 1 може позначати значення символьної константи. Довільне символьне значення може бути згенероване стандартною функцією *Chr*.

Перечислимі типи

Перечислимі типи позначають впорядковні набори значень, перераховуючи ідентифікатори, що позначають ці значення.

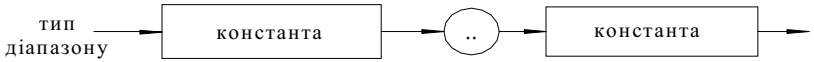


Приклад перечислимого типу:

```
type
Blood_pressure = (Low, Normal, High);
```

Типи піддіапазонів

Тип піддіапазону — це діапазон значень з порядкового типу, що називається **типом господаря**. Означення типу піддіапазону описує найменше та найбільше значення в піддіапазоні. Ось його синтаксис:



Приклади типів піддіапазонів:

```
0..99
-128..127
Low..High
```

Типи дійсних чисел

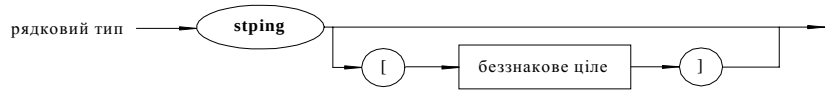
Тип дійсних чисел має множину значень, що є підмножиною дійсних чисел, що може бути представлена числами з плаваючою точкою з фіксованою кількістю цифр. Існує п'ять видів типів дійсних чисел: *Real, Single, Double, Extended* і *Comp*. Типи дійсних чисел розрізняються за діапазоном та точністю значень, як це показано у наступній таблиці.

Таблиця 3-4. Типи дійсних чисел

Тип	Діапазон	Значущі цифри	Розмір в байтах
Real	2.9×10^{-39}.. 1.7×10^{38}	11-12	6
Single	1.5×10^{-45}.. 3.4×10^{38}	7-8	4
Double	5.0×10^{-324}.. 1.7×10^{308}	15-16	8
Extended	3.4×10^{-4932}.. 1.1×10^{4932}	19-20	10
Comp	$-2^{63}+1$ .. $2^{63}-1$	19-20	8

Рядкові типи

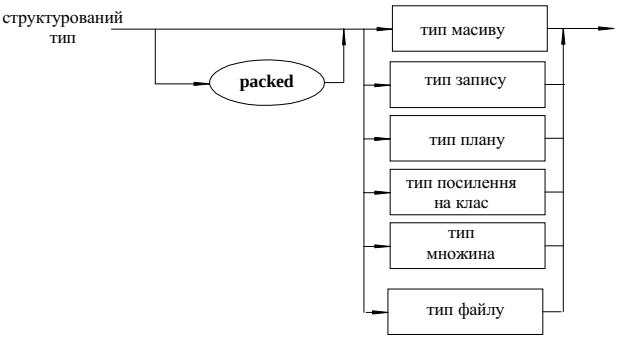
Значенням типу string (рядок) є послідовність символів з динамічною довжиною (залежить від фактичної кількості символів під час виконання програми) або із сталою кількістю символів (від 1 до 255).



Впорядкування між двома рядками встановлюється на основі впорядкування символів у відповідних позиціях.

Структуровані типи

Структурований тип, що характеризується методом його структурування та типом(ами) його компонент, зберігає більше як одне значення. Якщо тип компонента є структурним, то результуючий структурний тип матиме більше ніж один рівень структурування. Структурований тип може мати необмежене число рівнів структурування та максимально допустимий розмір структурованого типу є 65,520 байт.

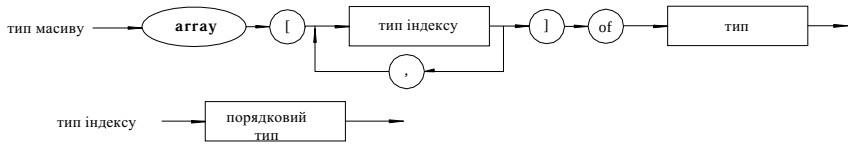


У стандартному Pascal слово **packed** вказує компілятору на ущільнення зберігання даних, незважаючи ні на що. У Object Pascal **packed** вказує на виконання автоматичне ущільнення там, де це можливо. Типи класів та посилання на типи класів є вер-

шинами об'єктно-орієнтованого програмування в Object Pascal. Про них піде мова далі.

Типи масивів (Array)

Масиви (Arrays) мають фіксоване число компонент одного типу — **типу компонента**. У наступних синтаксичних діаграмах тип компонента йде після слова **of**.



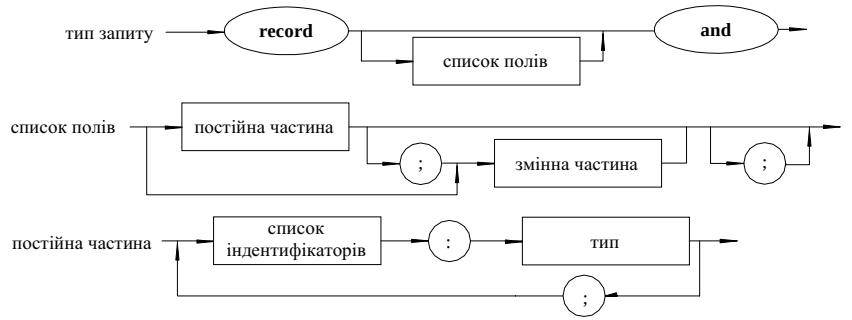
Приклади типу array:

array[1..100] **of Real**

Тип
array[Boolean] of array[1..10] of array[Size] of Real
інтерпретується так само, як:
array[Boolean,1..10,Size] of Real

Тип запису (Record)

Тип запису (record) охоплює задане число компонент або полів, що можуть бути різних типів. Декларація типу запису описує тип кожного поля та ідентифікатор, що іменує поле.



Приклад типу запису:

```
type
    TDateRec = record
      Year: Integer;
      Month: 1..12;
      Day: 1..31;
end;
```

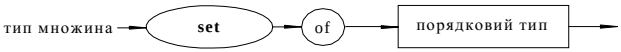
Більш складні приклади типів **записів із варіантами**

```
type
    TPerson = record
            FirstName, LastName: string[40];
            BirthDate: TDate;
            case Citizen: Boolean of
                    True: (BirthPlace: string[40]);
                    False: (Country: string[20];
                    EntryPort: string[20];
                    EntryDate: TDate;
                    ExitDate: TDate);
    end;

            TPolygon = record
            X, Y: Real;
            case Kind: Figure of
                    TRectangle: (Height, Width: Real);
                    TTriangle: (Side1, Side2, Angle: Real);
                    TCircle: (Radius: Real);
            end;
```

Тип множина (Set)

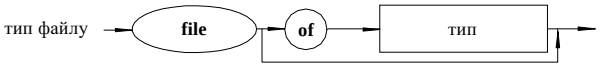
Область значень типу множина є декартовим добутком певного порядкового типу (базового типу). Декартовий добуток — це множина усіх можливих підмножин значень базового типу, включаючи порожню множину.



Кожен тип множина містить значення [ ], яке називається порожньою множиною.

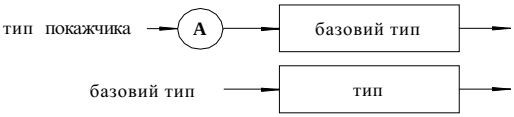
Тип файла (File)

Тип файла складається з лінійної послідовності компонент одного типу, який може бути довільним типом, окрім типу файла чи типу об'єкта. Число компонент у декларації типу файла не задається.



Типи покажчиків

Тип покажчика визначає множину значень, що вказують на динамічні змінні певного типу, що називається **базовим типом**. Змінна типу покажчик містить адресу пам'яті динамічної змінної.



Можна надавати значення змінним типу покажчик через процедуру *New*, оператор @, функцію *Ptr* або процедуру *GetMem*. New виділяє нову область пам'яті для динамічної змінної і зберігає адресу цієї області у змінній типу покажчик. Оператор @ срямовує змінну типу покажчик в область пам'яті, що включає довільну існуючу змінну або точку входу в процедуру чи функцію, включаючи змінні, що вже мають ідентифікатори. *Ptr* вказує змінній-покажчикові на певну адресу пам'яті. *GetMem* створює нову динамічну змінну певного розміру та розміщає адресу у змінну-покажчик.

Тип Pointer

Вмонтований тип *Pointer* позначає покажчик без типу, тобто, покажчик, що не вказує на певний заданий тип.

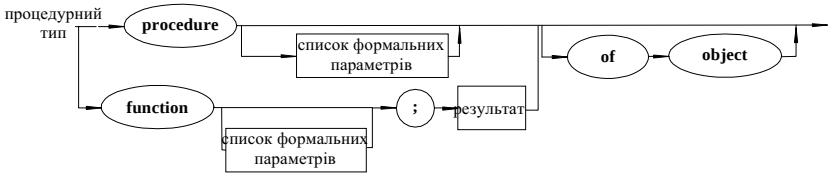
Type PChar

Object Pascal має вмонтований тип *PChar*, що служить для вказування на рядки. Модуль *System* оголошує *PChar* як

```
type PChar = ^Char;
```

Процедурні типи

Object Pascal дозволяє поводитися із процедурами та функціями, як із сутностями, що можуть присвоюватися змінним та передаватися в якості параметрів. Такі дії можна робити, використовуючи **процедурні типи**.



Існує дві категорії процедурних типів: покажчики на глобальні процедури та покажчики методів.

Покажчики глобальних процедур

Процедурний тип, що оголошується без речення **of object,** називається покажчиком глобальної процедури. Покажчик глобальної процедури може посилатися на глобальну процедуру чи функцію. Він зберігає адресу глобальної процедури чи функції. Приклади типів покажчиків глобальних процедур:

```
type
  TProcedure = procedure;
  TStrProc = procedure(const S: string);
  TMathFunc = function(X: Double): Double;
```

Покажчики методів

Процедурний тип, оголошений із реченням **of object,** називається покажчиком методу. Покажчик методу може посилатися на метод процедури або функції об'єкта. Внутрішньо він представлений як два покажчики: перший покажчик зберігає адресу методу, другий покажчик зберігає посилання на об'єкт, якому належить сам метод. Нижче наведені приклади типів покажчиків методів:

```
type
    TMethod = procedure of object;
    TNotifyEvent = procedure(Sender: TObject) of object;
```

Сумісність між типами

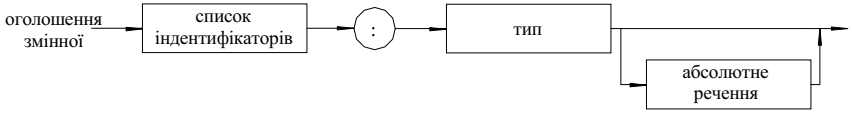
Сумісність між двома типами вимагається у виразах та операціях відношення. Сумісніть типів існує, коли виконується хоча б одна з умов (неповний перелік):

- Обидва типи є тотожними.
- Обидва типи є типами дійсних чисел.
- Обидва типи є цілочисельними типами.
- Один тип є підобластю іншого.
- Обидва типи є підобластями того ж типу господаря.
- Обидва типи є типами множина із сумісними базовими типами.
- Обидва типи є ущільненими рядковими типами з однаковою кількістю компонент.
- Один тип є типом рядка, а інший є або рядком, ущільненим рядком, або типом *Char* і т.д.

Змінні та константи типу

Декларації змінних

Змінна — це ідентифікатор, що позначає значення, яке може змінюватися. Декларація змінних показує список ідентифікаторів, що позначають нові змінні та їх тип.

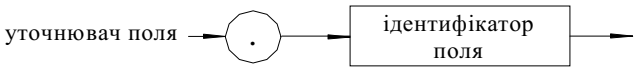


Приклад підрозділу декларації змінних:

```
    var
X, Y, Z: Double;
    I, J, K: Integer;
    Digit: 0..9;
    C: Color;
    Done, Error: Boolean;
    Operator: (Plus, Minus, Times);
    Hue1, Hue2: set of Color;
    Today: Date;
    Results: MeasureList;
    P1, P2: Person;
```

Визначальники полів у записах

Певне поле змінної-запису позначається посиланням на змінну, що посилається на змінну-запис, за яким слідує визначальник поля, що описує поле:



Приклади визначальників полів:

```
Today.Year Results[1].Count Results[1].When.Month
```

Визначальники компонента об'єкта

Формат визначальника компонента є таким же, що й для визначальника запису поля, тобто, він складається з інстанції (посилання на змінну), за якою слідує крапка та ідентифікатор компонента. Визначальник компонента, що визначає метод, називається визначальником методу.

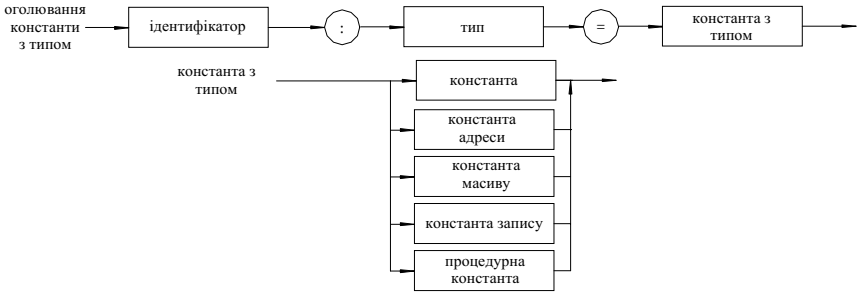
Покажчики та динамічні змінні

Значенням змінної-покажчика є або **nil** (не вказує ні на що), або адреса динамічної змінної.

Динамічна змінна, що вказується змінною-покажчиком, може бути отримана, написавши символ вказування (^) після змінної-покажчика.

Константи типу

Константи типу можна порівняти із ініціалізованими змінними — змінними, чиї значення означуються при входженні в їх блок. На відміну від констант без типу декларація констант типу визначає і тип, і значення константи.



Константи типу використовуються подібно до змінних того ж типу і вони можуть з'являтися в лівій частині речень присвоювання.

Константи простих типів

Декларація констант простих типів описує значення констант:

```
const
  Maximum: Integer = 9999;
  Factor: Real = -0.1;
  Breakchar: Char = #3;
```

Константи типу рядок

Декларація констант типу string описує максимальну довжину рядка та його початкове значення:

```
const
    Heading: string[7] = 'Section';
    NewLine: string[2] = #13#10;
    TrueStr: string[5] = 'Yes';
    FalseStr: string[5] = 'No';
```

Константи структурованих типів

Декларація константи структурованого типу описує значення кожного компонента структури. Object Pascal підтримує декларації констант таких типів, як масив, запис і множина. Константи типу файл або масив чи запис, що включає файл, не підтримуються.

Константи типу масив

Декларація констант типу масив описує значення елементів масиву. Приклад константи типу масив:

```
type
    TStatus = (Active, Passive, Waiting);
    TStatusMap = array[TStatus] of string[7];
const
    StatStr: TStatusMap = ('Active', 'Passive', 'Waiting');
```

Константи типу запис

Приклад констант типу запис:

```
type
    TPoint = record X, Y: Real;
    end;

    TVector = array[0..1] of Point;
    TMonth = (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct,
    Nov, Dec);
    TDate = record D: 1..31;
    M: Month;
    Y: 1900..1999;
    end;
```

```
const
    Origin: TPoint = (X: 0.0; Y: 0.0);
    Line: TVector = ((X: -3.1; Y: 1.5), (X: 5.8; Y: 3.0));
    SomeDay: TDate = (D: 2; M: Dec; Y: 1960);
```

Константи типу покажчик

Приклад:

```
type
    TDirection = (Left, Right, Up, Down);
    TStringPtr = ^String;
    TNodePtr = ^Node;
    TNode = record
        Next: TNodePtr;
        Symbol: TStringPtr;
        Value: TDirection;
    end;
const
    S1: string[4] = 'DOWN';
    S2: string[2] = 'UP';
    S3: string[5] = 'RIGHT';
    S4: string[4] = 'LEFT';
    N1: TNode = (Next: nil; Symbol: @S1; Value: Down);
    N2: TNode = (Next: @N1; Symbol: @S2; Value: Up);
    N3: TNode = (Next: @N2; Symbol: @S3; Value: Right);
    N4: TNode = (Next: @N3; Symbol: @S4; Value: Left);
    DirectionTable: TNodePtr = @N4;
```

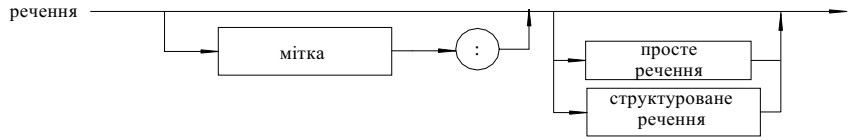
Константи типу процедура

Приклад:

```
type
    TErrorProc = procedure(ErrorCode: Integer);
procedure DefaultError(ErrorCode: Integer); far;
begin
    Writeln('Error ', ErrorCode, '.');
end;
const ErrorHandler: TErrorProc = DefaultError;
```

Програмні речення

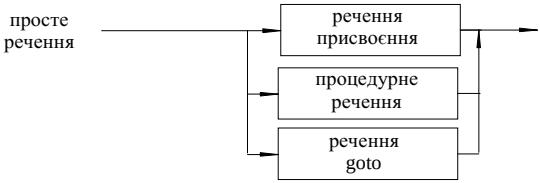
Речення описують алгоритмічні дії, що можуть бути виконані. Попереду речень можуть стояти мітки. На ці мітки можна посилатися, використовуючи речення **goto**.



Існує два головних види речень: прості та структуровані речення.

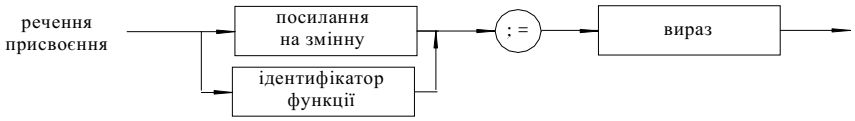
Прості речення

Просте речення — це речення, що не містить будь-яких інших речень.



Речення присвоєння

Речення присвоєння заміняють поточне значення змінної новим значенням, яке описане у виразі. Ці речення можна також використовувати для повернення значень функцій.

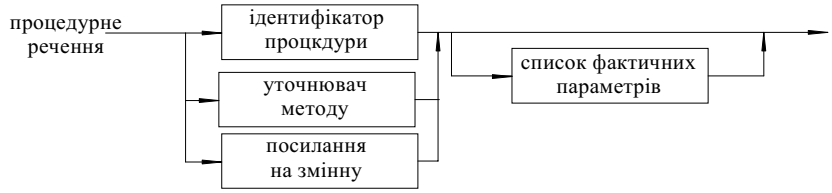


Деякі приклади речень присвоєння:

```
X := Y + Z;
Done := (I >= 1) and (I < 100);
Hue1 := [Blue, Succ(C)];
I := Sqr(J) — I * K;
```

Речення процедур

Речення процедур активізують процедуру, описану ідентифікатором процедури, визначником методу, визначником уточнюючого методу або посиланням на змінну процедурного типу.

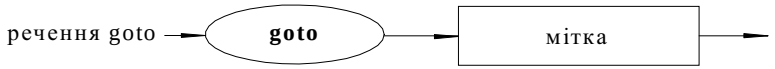


Приклади речень процедур:

```
PrintHeading;
Transpose(A, N, M);
Find(Name, Address);
```

Речення Goto

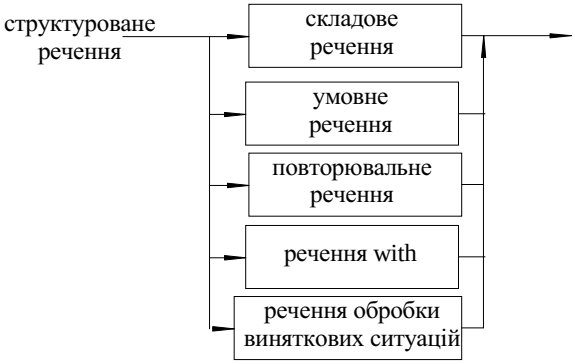
Речення **goto** переносить виконання програми на речення, позначене певною міткою. Синтаксична діаграма цього речення:



Хороше програмування використовує речення goto дуже рідко.

Структуровані речення

Структуровані речення — це конструкції, що складаються з інших речень, що повинні виконуватися послідовно (складові речення та речення **with**), за певних умов (умовні речення) або ж повторно (речення повторення).



Складові речення

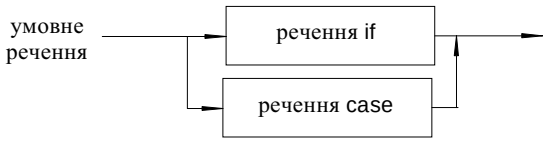
Складове речення показує, що вкладені у нього речення повинні виконуватися в тому порядку, в якому вони записані.

Приклад складового речення:

```
begin
    Z := X;
    X := Y;
    Y := Z;
end;
```

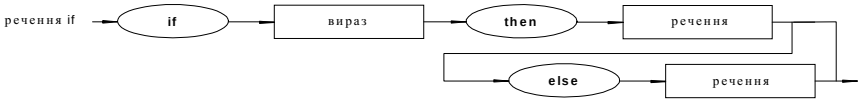
Умовні речення

Умовне речення вибирає для виконання одне речення (або ні одного) із вкладених у нього речень.



Речення If

Синтаксис речення **if**:



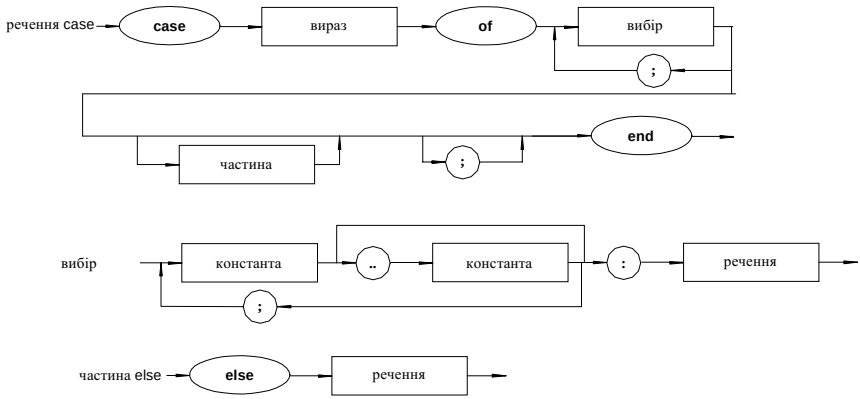
Два приклади речень **if**:

```
if X < 1.5 then
        Z := X + Y
else
        Z := 1.5;

if P1 <> nil then
        P1 := P1^.Father;
```

Речення Case

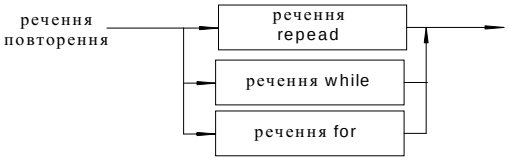
Речення **case** складається з виразу (селектора) і списку речень, попереду кожного з яких стоїть одна або більше констант (називаються **константами вибору**), або слово **else**.



```
case MySelector of
    5: Edit1.Text := 'Special case';
    1..10: Edit1.Text := 'General case';
end;
```

Речення повторення

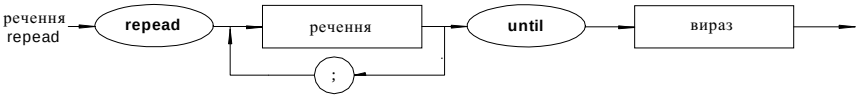
Речення повторення описує певні речення, що повинні виконуватися повторно.



Стандартні процедури *Break* та *Continue* можуть використовуватися для керування потоком повторюваних речень: *Break* закінчує повторюване речення; *Continue* продовжує виконання речення повторення з наступної ітерації повторюваного речення.

Речення Repeat

Речення **repeat** містить вираз, що керує повторенням виконання послідовності речень у цьому реченні **repeat**.



Речення між словами **repeat** та **until** виконуються послідовно доти, поки вираз в кінці послідовності не дорівнюватиме *True*.

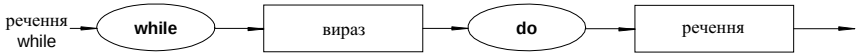
Приклади речень **repeat**:

```
repeat
    K := I mod J;
    I := J;
    J := K;
until J = 0;
```

```
repeat
    Write('Enter value (0..9): ');
    Readln(I);
until (I >= 0) and (I <= 9);
```

Речення While

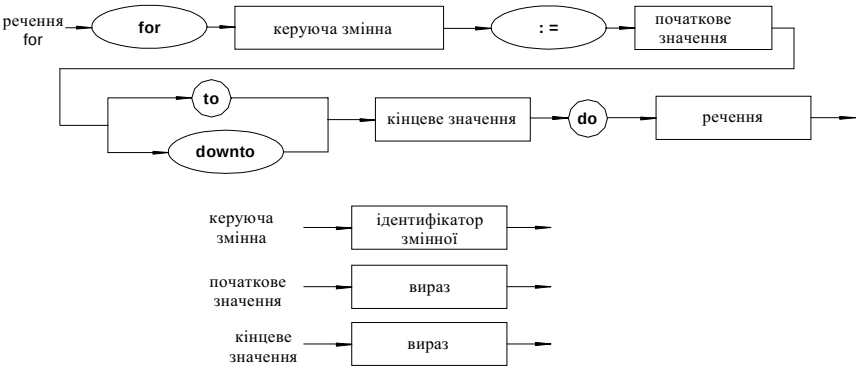
Речення **while** включає вираз, що керує повторним виконанням речення (яке може бути складовим).




```
while Data[I] <> X do I := I + 1;
while I > 0 do
begin
    if Odd(I) then Z := Z * X;
    I := I div 2;
    X := Sqr(X);
end;
```

Речення For

Речення **for** здійснює повторне виконання речення поряд із присвоєнням керуючій змінній прогресії значень.

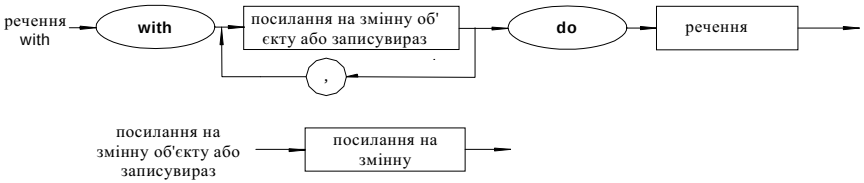


Керуюча змінна повинна бути порядкового типу. Її початкове та кінцеве значення повинні бути типу, сумісного з порядковим.

```
for I := 1 to 10 do
for J := 1 to 10 do
begin
        X := 0;
        for K := 1 to 10 do
                X := X + Mat1[I, K] * Mat2[K, J];
      Mat[I, J] := X;
end;
```

Речення With

Речення **with** — це спрощення посилання на поля записів та поля і методи об'єкта. В межах речення **with** посилання на поля однієї або більше змінних-записів можуть здійснюватися лише з використанням ідентифікаторів полів.



Нехай задано декларацію типу:

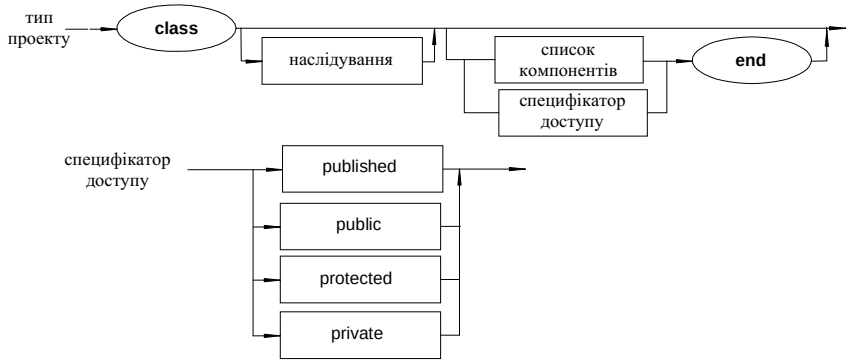
```
type
      TDate = record
      Day : Integer;
      Month: Integer;
      Year : Integer;
end;
var OrderDate: TDate;
```

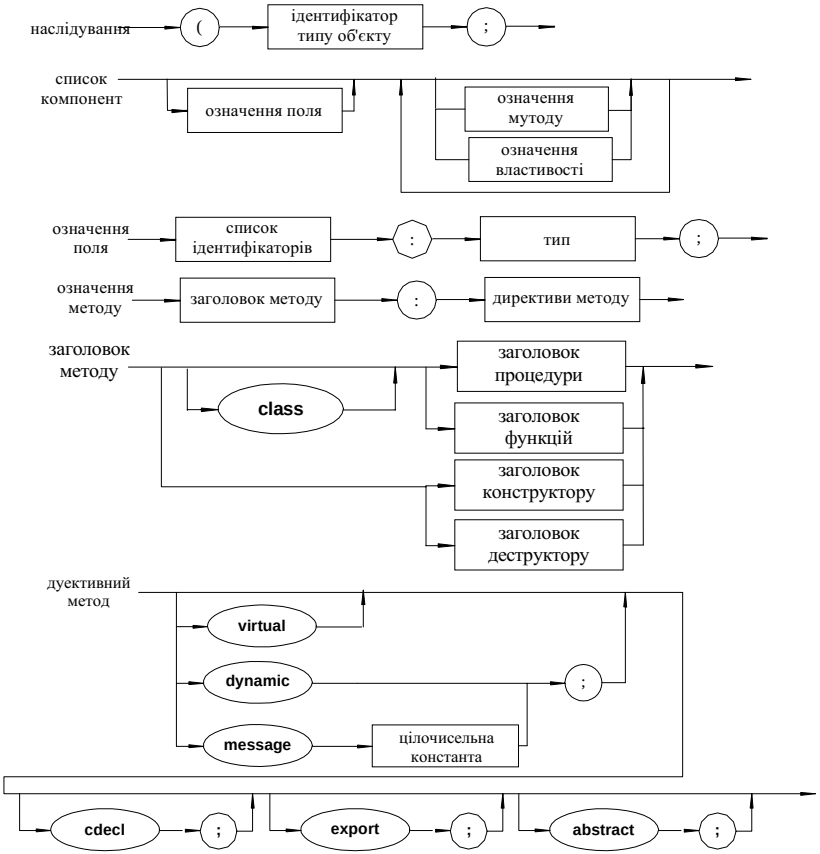
Тоді можна використати таке речення **with**:

```
with OrderDate do
    if Month = 12 then
            begin
                Month := 1;
                Year := Year + 1
            end
    else
            Month := Month + 1;
```

Типи класів

Тип класу — це структура, що складається з фіксованого числа елементів. Можливими елементами класу є поля, методи та властивості. На відміну від інших типів тип класу може бути декларований лише в частині декларацій типів файла програми чи модуля. Отже, тип класу не може декларуватися в частині декларацій змінних або в блоках процедур, функцій чи методів.





Інстанції та посилання

Інстанція типу класу — це динамічно виділений блок пам'яті, розбиття якого визначається типом класу. Інстанції типу класу часто називають **об'єктами**. Об'єкти створюються конструкторами та руйнуються деструкторами. Кожен об'єкт типу класу має єдину копію полів, оголошених в типі класу, але всі вони використовують ті ж самі методи.

Змінна типу класу включає **посилання** на об'єкт типу класу. Змінна не містить об'єкта самого, а швидше є покажчиком на блок пам'яті, виділений під об'єкт. Подібно до змінних-покажчи-

ків змінні багатьох типів класів можуть посилатися на той же об'єкт. Більше того, змінна типу класу може містити значення **nil**, що свідчить про те, що вона не посилається на жоден об'єкт.

Примітка. На відміну від змінних-покажчиків не має потреби "перепосилати" змінну типу класу, щоб отримати доступ до об'єкта, на який посилаємося. Іншими словами, тоді як потрібно писати *Ptr^.Field*, щоб отримати доступ до поля в динамічно розміщеному записі, то оператор ^ вживається, коли ми доступаємося до елементу об'єкта. Використовується наступний синтаксис: *Instance.Field*.

Домовимося, що термін **посилання на об'єкт** використовується, щоб позначити значення типу класу. Наприклад, змінна типу класу містить посилання на об'єкт і конструктор повертає посилання на об'єкт.

До того ж , термін посилання на клас використовується, щоб позначити значення типу посилання на клас. Наприклад, ідентифікатор типу класу — це посилання на клас, а змінна типу посилання на клас містить посилання на клас.

Елементи класу

Елементи класу — це поля, методи і властивості. Елементи класу часто називають його членами.

Поля

Декларація поля в класі визначає пункт даних, що існує в кожній інстанції класу. Це дуже нагадує поле в запису.

Методи

Метод — це процедура або функція, що виконує операцію над об'єктом. Перша частина виклику методу описує об'єкт, над яким метод повинен оперувати. Декларація методу у типі класу відповідає декларації **forward** цього методу. Це означає, що десь слідом за декларацією типу класу і протягом того ж модуля метод повинен бути реалізований декларацією означення.

Протягом реалізації методу ідентифікатор *Self* представляє параметр, що посилається на об'єкт, для якого було викликано метод.

Консруктори та **деструктори** — це спеціальні методи, що керують побудовою та руйнуванням об'єктів.

Конструктор визначає дії, пов'язані із створенням об'єкта. Коли він викликається, він діє як функція, що повертає посилання на заново розміщену та ініціалізовану інстанцію типу класу.

Деструктор визначає дії, пов'язані із руйнуванням об'єкта. Коли він викликається, деструктор звільнить пам'ять, що була виділена під об'єкт.

Метод класу — це процедура або функція, що оперує над посиланням на клас замість посилання на об'єкт.

Властивості

Декларація властивості у класі визначає поіменований атрибут для об'єктів класу та дії, пов'язані з читанням та записом атрибуту. Глибше про властивості буде подано далі.

Наслідування

Тип класу може наслідувати елементи з іншого типу класу. Якщо *T2* **наслідує** *T1*, тоді *T2* — це **нащадок** *T1*, а *T1* — це **предок** *T2*. Наслідування є транзитивним; тобто, якщо *T3* наслідує *T2*, а *T2* наслідує *T1*, то *T3* також наслідує *T1*. **Домен** типу класу складається із себе та усіх його нащадків. Клас нащадка включає усі елементи, означені в класах його предків. Клас нащадка може додати нові елементи до тих, що він успадковує. Однак він не може викинути означення елементу, визначеного у класі предка.

Наперед заданий тип класу *TObject* — є основний і остаточний предок усіх типів класів. Якщо декларація типу класу не описує тип предка (тобто, якщо успадкована частина декларації класу опущена), тип класу походитиме із *TObject*.

TObject декларований в модулі *System*; він визначає методи, що застосовуються до всіх класів. Опис цих методів можна знайти далі.

Елементи та сфера їх дії

Сфера дії ідентифікатора елементу, декларованого у типі класу, простягається від місця декларації до кінця декларації типу

класу та розповсюджується на усі нащадки типу класу та блоки декларації усіх методів класу. Також сфера дії ідентифікаторів компонента включає визначники полів, методів та властивостей та речення **with**, що оперують над змінними заданого типу класу.

Ідентифікатор елементу, який декларовано у типі класу, може бути передекларований у блоці декларації методу типу класу. В цьому випадку для доступу до елементу використовується параметр *Self*.

Ідентифікатор елементу у типі класу предка може бути переоголошений у типі класу нащадка. Таке переоголошення приховує успадкований елемент, хоча ключове слово **inherited** може бути використане, щоб повернути успадкованому елементу його сферу дії.

Майбутня декларація

Для використання можливості **майбутньої декларації** використовується зарезервоване слово **class**, за яким нічого далі не слідує. Майбутня декларація повинна прийти до звичайної декларації в тому ж підрозділі декларації. Майбутня декларація дозволяє декларуватися одночасно залежним класам. Наприклад:

```
type
      TFigure = class;
      TDrawing = class
      Figure: TFigure;
      :
end;
TFigure = class
      Drawing: TDrawing;
:
end;
```

Правила сумісності типів класів

Тип класу є сумісним з типами класів довільного предка; отже, під час виконання програми змінна типу класу може поси-

латися на інстанцію цього типу або ж інстанцію типу нащадка. Наприклад, для декларацій:

```
type
      TFigure = class
      :
end;
TRectangle = class(TFigure)
      :
end;
TRoundRect = class(TRectangle)
      :
end;
TEllipse = class(TFigure)
      :
end;
```

значення типу *TRectangle* може бути присвоєним змінним типів *TRectangle, TFigure* і *Tobject*, а під час виконання програми змінна типу *TFigure* може бути або **nil,** або посиланням на інстанції типів *TFigure, TRectangle, TRoundRect, Tellipse* або довільну іншу інстанцію типу — нащадка *TFigure*.

Видимість компонент

Видимість ідентифікатора елементу керується атрибутом видимості секції елементів, що декларує ідентифікатор. Існує таких чотири атрибути видимості: **published**, **public**, **protected**, and **private**.

Елементи секції Public

Ідентифікатори елементів, декларованих у секціях **public,** не мають спеціальних обмежень на їх видимість.

Елементи секції Published

Правила видимості елементів цієї секції подібні до секції public. Єдина різниця між елементами секцій published та public полягає в тому, що **інформація часу виконання** генерується для полів та властивостей, декларованих у секції **published.** Ця

інформація часу виконання дозволяє додатку динамічно запитувати поля та властивості в іншому випадку невідомого типу класу.

Поля, означені у секції published, повинні бути типу класу. Поля усіх інших типів заключаються у секції **public**, **protected** і **private**.

Властивості, визначені в секції **published,** не можуть бути властивостями-масивами. Більше того, тип властивості, означений у секції **published,** повинен бути порядкового типу, дійсних типів (*Single, Double, Extended, Comp,* але не *Real*), рядкового типу, типу малої множини, типу класу або типу покажчика методу. **Тип малої множини** — це тип множини з базовим типом, нижня і верхня межі якого мають порядкові значення між 0 та 15. Іншими словами, тип малої множини — це множина, що вміщається у байті або слові.

Елементи секції Protected

При здійсненні доступу через тип класу, декларований в поточному модулі, ідентифікатори елементів із секції protected є видимими. У всіх інших випадках вони є прихованими. Доступ до елементів класу секції protected обмежується згідно з реалізацією методів класу та його нащадків. Отже, елементи класу, що створені лише для використання в підрозділах реалізацій породжених класів, переважно оголошуються як захищені (protected).

Елементи секції Private

Видимість ідентифікатора елементу, декларованого у секції **private,** обмежена модулем, що містить декларацію типу класу.

ОСНОВНІ СИНТАКСИЧНІ КОНСТРУКЦІЇ С (JAVA)

Зовсім не применшуючи усіх можливостей та переваг синтаксису Pascal, слід визнати, що на сьогоднійшній день більшість програм розробляється, дотримуючись інших "граматичних правил" — синтаксису С (читається "Сі"). Даний синтаксис з'явився в однойменній мові програмування С і тепер активно використовується в специфікаціях сучасних мов програмування — C++, Java. Особливий інтерес викликає остання, завдяки безпосередній інтегрованості у протоколи Internet, про що піде мова далі.

Мова С багато перейняла у свого попередника — BCPL. С була розроблена в США співробітниками фірми Bell Laboratories на межі 70-х років. Приблизно в той же час вона була використана для розробки операційної системи UNIX. Питання підвищення здатності перенесення відігравали велике значення з самого початку розробки як мови С, так і операційної системи UNIX, тому їх розповсюдження на нові комп'ютери відбувалося дуже швидко.

Перший опис мови С був даний її авторами — Б. Керніганом та Д. Рітчі (є російський переклад) [1]. Досить показовим фактом про оригінальну вдачу авторів є назва мови С (за однією з версій — просто третя буква англійського алфавіту). На жаль, опис не був суворим і повним і містив ряд моментів, що допускали неоднозначне тлумачення. Це призвело до того, що різні розробники систем програмування у своїх реалізаціях мови С дотримувались різних діалектів. Протягом тривалого часу фактичним стандартом мови С була її реалізація у сьомій версії операційної системи UNIX, яка практично і була визнана Американським Національним Інститутом Стандартів (ANSI) в 1989 р.

Наступним нащадком С стала мова програмування C++. Ранні версії цієї мови використовувалися з 1980 р. під узагальненою назвою "С з класами". Перший імпульс створення C++ виник тому, що автор хотів написати деякі дії (емуляції), які б керувалися подіями, для чого б ефективно підійшла Simula-67, якби не міркування ефективності. Тому в мові C++ було повністю перейнято синтаксис С, а також реалізовано такі важливі поняття із Simula-67, як клас, похідний клас, віртуальна функція, перейнято

деякі "вільності" програмування з Algol-68. Вперше за межами дослідницької групи C++ було використано у червні 1983 р.

Назву C++ придумав Рік Маскітті. Ця назва відображає еволюційний характер змін мови C. "++" — це оператор інкрементації в мові C. Більш коротка назва — C+ — це синтаксична помилка. Знавці семантики C вважають, що C++ гірша назва, ніж ++C. Мова не називається D, тому що вона — розширення C, і в ній не робляться спроби розв'язання проблем шляхом виключення будь-яких засобів C.

90-ті роки XX століття були ознаменовані появою WWW. Прагнучи розробити універсальний засіб для надання Web-сторінкам можливості інтерактивності, фірмою Sun було створено специфікації мови Internet-програм, яка отримала назву Java. За однією з версій назва Java походить від сорту популярної кави, яку так смакували автори мови. Тому на пікторамах вікон виконання програм Java часто можна зустріти горнятко паруючої кави.

Завдяки своїй популярності і, можливо, вдалості синтаксис мови C у своїй основі залишився у Java. Хоча розробники Java вирішили відмовитися від сумісності із C заради простоти мови. Для порівняння, розробники C++ добивалися сумісності із мовою C, розробленою майже 30 років тому. Як результат, мова C++ вийшла винятково складною.

Наступний огляд слугуватиме екскурсією у синтаксис C в своїй реалізації у Java. Її мета — не стільки дати повний і точний виклад мови, як вказати на особливості синтаксичних конструкцій, на які опирається більшість програмного забезпечення сьогодення і сподіватимемося — майбуття.

Вирази

Оператором Java може бути коментар, блок, опис, вираз або оператор керування. Більшість операторів в Java закінчуються символом; (крапка з комою). Лише блоки та коментарі не підлягають цьому правилу.

Коментарі

В Java означено три типи коментарів. Найрозповсюдженіші коментарі — дві косі риски (//) в рядку. Все, що розміщується між // і кінцем рядка, вважається коментарем.

Другий тип коментаря починається із символів /*, а закінчується символами */. Символ закінчення рядка в цьому випадку ігнорується.

Третій тип коментаря — починається /** і закінчується */ — використовується спеціальними утилітами для документування програми.

Блоки

Переважно програма розбивається на логічні блоки, код яких обмежується фігурними дужками { та }.

Оголошення змінних

Усі локальні змінні повинні бути оголошені до першого їх використання у програмі. Локальна змінна — це змінна, оголошена всередині функції; область її видимості обмежена цією функцією. Такі описи можуть міститися в довільному місці всередині функції. При оголошенні змінної вказується її тип, ім'я змінної; може також бути знак рівності (=) і вираз, що задає початкове значення змінної.

```
Int nAnInt; //оголошення змінної
int nASecondInt = 10; //оголошення іншої змінної і присвоєння їй початкового значення.
```

Вмонтовані типи даних

У таблиці наведено вмонтовані типи Java. Кожна змінна при оголошенні отримує деяке значення за припущенням.

Таблиця. Вмонтовані в Java типи даних

Тип даних	Розмір, розр.	Значення за припущенням
boolean	8	false
byte	8	0
char	16	‘x0’
short	16	0
int	32	0
long	64	0
float	32	0.0F
double	64	0.0D

Імена змінних. Ідентифікатори

Ім’я змінної може бути довільним дозволеним в Java ідентифікатором. Ідентифікатор може починатися з букви, символу підкреслення (_) або знаку долара ($). Решта частина ідентифікатора може містити, крім цього, і цифри.

Константи

Звичайні константи цілого типу — 32-розрядні десяткові числа із знаком. З нуля починаються восьмирічні константи, а з 0x або 0X — шістнадцятирічні. Символи від A до F в шістнадцятирічних числах можуть бути як великими, так і малими. Константи типу long можуть закінчуватися символами l або L, double — d або D, float — f або F.

Логічні операції

Логічні оператори &, | і ^ можна застосовувати до аргументів типу boolean для виконання логічних операцій AND, OR і XOR відповідно. Зверніть увагу, що лівий і правий аргументи операторів & і | обчислюються навіть тоді, коли в цьому немає потреби. Два спеціальних оператори — && і || — виконують ті самі операцій AND і OR, але в короткому варіанті (тобто, якщо на основі значення першого аргументу немає сенсу обчислювати значення другого аргументу, то воно не обчислюється).

Спеціальні оператори

Незвичними для цілочисельних змінних є оператори інкрементації (++) та декрементації (- -), обидва з яких можуть приймати префіксну ++nA та постфіксну nA++ форми. Відмінності у результатах їх виконання можна прослідкувати з прикладу:

```
int nA;
int nB;
int nC;
nA = 1;
nB = ++nA; // префіксна операція — після неї nB і nA дорівнюють 2
nA = 1;
nC = nA++; // постфіксна операція — після неї nC дорівнює 1, nA дорівнює 2.
```

Дивними є оператори, що поєднують бінарну операцію та присвоєння. Якщо op — бінарна операція, то можливий такий вираз:

```
a op= b; // це еквівалентно до виразу a = a op b.
```

Цей запис застосований до всіх бінарних операцій. Та найцікавішим є так званий **тернарний** оператор (тернарний означає потрійний і вказує, що оператор має три аргументи):

```
(boolexpr) ? expr1 : expr2;
```

Спочатку обчислюється логічне значення boolexpr. Якщо воно дорівнює true, то значенням оператора буде значення expr1, в іншому випадку — expr2.

Оператори керування

Оператор розгалуження if:

```
if (boolexpr)
{
        // довільне число операторів або виразів
}
else
{
        // довільне число операторів або виразів
}
```

Частина з else, звісно, необов'язкова.

Оператор циклу while

Є два види циклу while:

```
while (boolexpr)
{
        // довільне число операторів або виразів
}

i

do
{
        // довільне число операторів або виразів
}
while (boolexpr);
```

Ключове слово break дозволяє програмі перервати виконання циклу і передати керування за його межі. Причому, вказавши у break ім'я відповідної мітки, можна відразу вийти з кількох вкладених циклів.

Ключове слово continue відразу передає керування на дужку, що закривається, викликаючи початок наступної ітерації циклу.

Оператор циклу for:

```
for (expr1; boolexpr2; expr3)
{
        // довільне число операторів або виразів
}
```

Тут expr1 — вираз, що використовується для ініціалізації індексної змінної, boolexpr2 — умова виконання циклу, expr3 — вираз для зміни індексної змінної.

Оператор вибору switch використовується для вибору з кількох варіантів:

```
switch (expr)
{
        case cexpr1:
                // довільне число операторів або виразів
                break;
        case cexpr2:
                // довільне число операторів або виразів
        default:
                // оператори, що виконуються за припущенням
}
```

Спочатку обчислюється значення виразу expr. Далі його значення порівнюється із константними виразами після ключових слів case. При співпаданні виконуються відповідні командні блоки.

КОНЦЕПЦІЇ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПІДХОДУ

Потреба нових концепцій створення програмного забезпечення

З початку 60-х років XX століття завдяки появі високопродуктивних компіляторів мов високого рівня розробка програм стає індустрією. Залишається позаду час, коли програми розроблялися вченими з університетів самотужки або в невеликих лабораторіях. З'являються потужні корпорації із випуску програмного забезпечення, в штаті яких тисячі висококласних програмістів. Для налагодження співпраці між ними при розробці серйозних програмних продуктів виникає потреба у встановленні певних механізмів взаємодії між різними програмними модулями. Така взаємодія може бути досягнута лише при наданні модулям властивості функціональності (тобто, здатності поводити себе згідно з обставинами та командами).

З іншої сторони, потреба в існуванні певних важелів керування програмними модулями виникає також з бажання використовувати одні й ті ж програмні модулі в різних проектах — при розробці різного програмного забезпечення. Така властивість називається **переносимістю програмного коду**. Вона має неабияке значення на ринку розробників програмного забезпечення, за яким, віриться, великі перспективи.

Появу і технічну реалізацію нових концепцій програмування активно стимулювала робота багатьох фірм над створенням принципово нових програмних інтерфейсів (наприклад, операційні системи-оболонки з графічним інтерфейсом).

Об'єктно-орієнтований підхід традиційно використовується в науці і техніці при розв'язуванні масштабних задач. Застосування його до програмування отримало назву об'єктно-орієнтованого програмування (ООП).

Java як остаточне втілення ООП

Далі розглянемо реалізацію головних понять ООП на прикладі мови програмування Java.

Java вибрана тому, що, крім унікальної інтегровності з Internet, вона ще й розроблялася як повністю об'єктно-орієнтована мова програмування (скажімо, C++, Delphi мають деякі відступи від ООП в своєму синтаксисі, зумовлені, в першу чергу, прагненням прийнятності синтаксису своїх необ'єктно-орієнтованих попередників). Будь-які типи даних Java-програми, починаючи від базових і закінчуючи самою програмою, розглядаються з точки зору ООП. Отже, короткий екскурс в ООП.

Означення класу

Для того, щоб зрозуміти поняття класу, необхідно мати уявлення про ООП. "ООП — це більше філософія програмування, ніж набір інструкцій або ключових слів мови Java" [1]. Спробуємо продемонструвати доречність ООП на практиці медичного працівника. При призначенні пацієнту в ході лікування певного хіміопрепарату лікар враховує такі фактори:

- протипоказання при певних видах захворювань;
- застереження для певної категорії пацієнтів (наприклад, вагітність);
- реакції різних систем людського організму (центральна нервова система, серцево-судинна система, дерматологічні захворювання, ендокринна система, гематологічна система, респіраторна система і інші, зумовлені, наприклад, певними дозуваннями);
- можливість передозування, токсикологія;
- взаємодія з іншими хіміопрепаратами;
- фармакодинаміка та фармакокінетика (тривалість і порядок дії);
- механізм дії.

Але, з іншої сторони, він не враховує наступне (принаймні, це його цікавить набагато менше):

- торговельні фірми, які займаються продажем даного хіміопрепарату;
- технологія промислового виробництва даного хіміопрепарату;
- ім'я вченого — автора хіміопрепарату.

І взагалі, в уяві лікаря хіміопрепарат — не сукупність хімічних речовин та ланцюжок потрібних реакцій, а зручно упакований предмет з детальною документацією щодо застосування.

Тобто, людина в своєму житті використовує цілком реальні об'єкти-одиниці, не заглиблюючись у їх внутрішню будову і це набагато простіше.

Далі наведемо дещо розширене означення класу.

Означення. Класом називається деяка абстрактна сутність, що задовольняє наступні вимоги:

- клас повинен володіти добре продуманим інтерфейсом, тобто, способом взаємодії. Інтерфейс повинен надавати споживачу можливість виконати будь-яку операцію, що підтримує клас;
- клас повинен символізувати концепцію в проблемній області, тобто, він повинен грунтуватися на чомусь реальному;
- клас повинен містити в собі все, що може вимагатися від нього іншими класами;
- клас повинен бути винятково надійним. Клас повинен відсіювати некоректні команди. Наприклад, неможливо виконати команду — призначити аспірин для лікування астматичного пацієнта.

Тепер про те, як такі міркування ООП виглядають в Java. Отже, слідуючи ООП, клас — це єдина можливість створити новий тип даних. Означення типу даних (класу) виглядає таким чином:

```
class MyClass
{
    // тут розміщуються елементи класу
}
```

Елементами класу можуть бути змінні класу та методи.

Змінні класу

Змінні класу (називаються ще властивостями, полями, членами) використовуються для опису властивостей класу.

Розглянемо клас, що описує хіміопрепарат:

```
class Drug
{
    // далі означуються змінні класу

    // назва хіміопрепарату
    String m_sTitle;

    // концентрація хіміопрепарату в плазмі (мг/мл)
    double m_dPlasmaDrugConcentration;

    // неперервне введення
    boolean m_bIntravenousInfusion;

    // швидкість неперервного введення
    double m_dInfusionRate;

    // доза при введенні з перервами
    double m_dDose;

    // час між введеннями хіміопрепарату у випадку з перервами
    double m_dDosingInterval;
}
```

Зрозуміло, що назва — важлива характеристика хіміопрепарату, також важливим є порядок дозування.

Методи класу

Елементи класу, які є функціями, називаються методами класу. Як доказ повного дотримання концепцій ООП в Java є те, що усі функції повинні бути методами якогось класу (на-

приклад, в Delphi такого немає). Розглянемо означення методу на прикладі

```
import java.lang.String;

class Drug
{
      // далі означуються змінні класу

      // назва хіміопрепарату
      String m_sTitle;

      // концентрація хіміопрепарату в плазмі (мг/мл)
      double m_dPlasmaDrugConcentration;

      // неперервне введення
      boolean m_bIntravenousInfusion;

      // швидкість неперервного введення
      double m_dInfusionRate;

      // доза при введенні з перервами
      double m_dDose;

      // час між введеннями хіміопрепарату у випадку з перервами
      double m_dDosingInterval;

      // далі — методи класу

      // призначене дозування (мг/хв)
      double DosingRate( )
      {
            if (m_bIntravenousInfusion)
            {
                  return m_dInfusionRate;
            }
            else return m_dDose / m_dDosingInterval;
```

```
    }

    // AverageDrugConcentration — середня концентрація
    // хіміопрепарату (мг/мл)
    double AverageDrugConcentration(double dClearance)
    {
        return DosingRate( ) / dClearance;
    }
}

class Patient
{
    String m_sName; // прізвище та ім'я пацієнта
    boolean m_bMale; // чоловік чи
    boolean m_bFemale; // жінка
    int m_nAge; // вік
    double m_dHeight; // зріст (см)
    double m_dPlasmaCreatinine; // (мг/дл)
    double m_dRenalFunctionClearance; // значення очищення
з таблиці ренальної функції
    Drug m_Drug; // хіміопрепарат, що призначається

    // методи класу

// вага (кг)
    double BodyWeight( )
    {
        if (m_bMale)
        {
            return 0.73 * m_dHeight — 59.42;
        }
        else
        {
            // для жінок
            return 0.65 * m_dHeight — 50.74;
        }
    }
```

```
        double BodySurfaceArea( )
        {
                return Math.sqrt((m_dHeight * BodyWeight( )) / 3600);
        }

        // CreatinineClearance — очищення креатиніну (мл/хв)
      double CreatinineClearance()
      {
        if (m_bMale)
        {
                return ((140 — m_nAge) * BodyWeight( ) ) /
(m_dPlasmaCreatinine * 72);
        }
        else
        {
                return 0.85 * ((140 — m_nAge) * BodyWeight( ) )
/ (m_dPlasmaCreatinine * 72);
                }
      }

        // очищення (мл/хв)
        double Clearance( )
        {
                return m_dRenalFunctionClearance * BodyWeight( );
        }

      }
```

Опис методу починається з означення типу об'єкта, що повертається методом. Якщо метод не повертає об'єктів, то тип об'єкта, що повертається, буде void. Тип об'єкта, що повертається, вказується перед іменем методу. За іменем методу в круглих дужках міститься список так званих формальних параметрів (аргументів) методу. Якщо метод не містить аргументів, то список повинен бути порожнім. Якщо метод повертає щось (на відміну від void), то його тіло повинно містити ключове слово return, за яким — об'єкт, що повертається.

Об'єкти

Поняття об'єкта і класу досить близькі. Клас вказує на тип предмета. Наприклад, аспірин — це речовина, що належить до класу хіміопрепаратів. З іншої сторони, кожен конкретний аспірин, призначений певному пацієнту — це **об'єкт**. Він може бути представлений змінною patient_aspirin, описаною, як показано нижче:

```
Drug patient_aspirin = new Drug( );
```

Створювати об'єкти одного класу можна по-різному багато разів.

Доступ до елементів об'єкта

Доступ до елементів об'єкта здійснюється за допомогою оператора крапка (.). Розглянемо наступну функцію:

```
void Example( )
{
    Patient currentPtnt = new Patient( );
    currentPtnt.m_sName = 'Петренко Тарас';
    currentPtnt.m_bMale = true;
    currentPtnt.m_nAge = 30;
    currentPtnt.m_dHeight = 180.5;
    currentPtnt.m_dRenalFunctionClearance = 30.0;
    currentPtnt.m_Drug = new Drug ( );
    currentPtnt.m_Drug.m_sTitle = 'Aspirin';
    currentPtnt.m_Drug.m_bIntravenousInfusion = false;
    currentPtnt.m_Drug.m_dDose = 325.0;
    currentPtnt.m_Drug.m_dDosingInterval = 300.0;
    doubledrugConcentration=
currentPtnt.m_Drug.AverageDrugConcentration(Clearance( ));

    // у змінній drugConcentration — середня концентрація
хіміопрепарату
    }
```

Тут, наприклад, конструкція currentPtnt.m_sName здійснює доступ до елемента m_sName об'єкта currentPtnt, а currentPtnt.m_Drug.AverageDrugConcentration(double) викликає метод-функцію класу Drug.

Наслідування

У розумінні об'єктів реального світу, що нас оточують, важливою є властивість наслідувати властивості своїх об'єктів-батьків. Наприклад, на запитання "Що таке аспірин?" студент-медик відповість приблизно так: "Це хіміопрепарат, що застосовується у стоматології для лікування післяопераційного болю, у медицині — для лікування болю і лихоманки; може використовуватися для профілактики інфаркту міокарда та ішемічної хвороби серця; управління ревматоїдним артритом, ревматичною лихоманкою, остеоартритом та подагрою (висока доза)". Подробиці тут не істотні — важливо те, що будь-яка відповідь починатиметься із слів "Аспірин — це хіміопрепарат ..." Використання таких абстрактних термінів, як хіміопрепарат (про що піде мова далі) сприяє систематизації інформації. Систематизація, в свою чергу, допомагає істотно скоротити кількість інформації, що потрібно запам'ятати. Наприклад, наступна модифікація класу Drug зменшує обсяг інформації, що потрібно зберігати, та робить класи більш гнучкими для розв'язання різноманітних задач за допомогою введення наступних підкласів:

```
public class Drug
{
    // далі означуються елементи класу
}

public class Aspirin extends Drug
{
    String sBrandName;

    // ... решта означень класу
}
```

```
public class Abacavir extends Drug
{
      String sBrandName;

      // ... решта означень класу
}

public class Abciximab extends Drug
{
      String sBrandName;

      // ... решта означень класу
}
```

Тобто, з кожним конкретним хіміопрепаратом пов'язується власний клас.

Абстрактні класи. Абстрактні методи

Немає нічого дивного в тому, що наш приклад з хіміопрепаратами ми розпочали з класу Drug, який не може існувати сам по собі. Дійсно, немає хіміопрепарату, який би не належав до підкласу конкретних хіміопрепаратів (наприклад, Abacavir, Abciximab, Absorbable gelatin, Aspirin і т.д.). Тобто, кожен хіміопрепарат є або Acarbose, або Acebutolol, або Acetaminophen. Клас, такий як Drug, є більш абстрактним поняттям, ніж конкретний фізичний об'єкт, який безпосередньо можна взяти. Такий клас називається **абстрактним класом**. Головна особливість абстрактного класу — не існує об'єктів — екземплярів абстрактного класу. Тобто, не існує екземплярів класу Drug, існують лише екземпляри його підкласів. Виникає питання про доцільність використання абстрактних класів. Відповідь продемонструємо модифікованим класом Drug:

```
import java.lang.String;

abstract public class Drug
```

```
{
      // далі означуються змінні класу

      // концентрація хіміопрепарату в плазмі (мг/мл)
      double m_dPlasmaDrugConcentration;

      // далі — методи класу

      // призначене дозування (мг/хв)
      abstract public double DosingRate( )

      // AverageDrugConcentration — середня концентрація
      // хіміопрепарату (мг/мл)
      double AverageDrugConcentration(double dClearance)
      {
            return DosingRate( ) / dClearance;
      }

}

public class IntravenousInfusionDrug extends Drug
{
      // швидкість неперервного введення
      double m_dInfusionRate;

      double DosingRate( )
      {
            return m_dInfusionRate;
      }
}

public class IntermittentlyGivenDrug extends Drug
{
      // доза при введенні з перервами
      double m_dDose;

      // час між введеннями хіміопрепарату у випадку з перервами
```

```
    double m_dDosingInterval;

    double DosingRate( )
    {
            return m_dDose / m_dDosingInterval;
    }
}

public class Aspirin extends IntermittentlyGivenDrug
{
    String m_sBrandName;

    // ... решта означень класу
}
```

Тут, крім виділення нового класу Aspirin для представлення конкретного хіміопрепарату, надана можливість враховувати спосіб введення хіміопрепарату (неперервно — IntravenousInfusionDrug або ж з перервами — IntermittentlyGivenDrug) (рис. 1). Метод DosingRate( ) класу Drug є абстрактним, оскільки неможливо розрахувати швидкість дозування для хіміопрепарату взагалі. В подальшому DosingRate( ) реалізовано у конкретних підкласах IntravenousInfusionDrug та IntermittentlyGivenDrug.

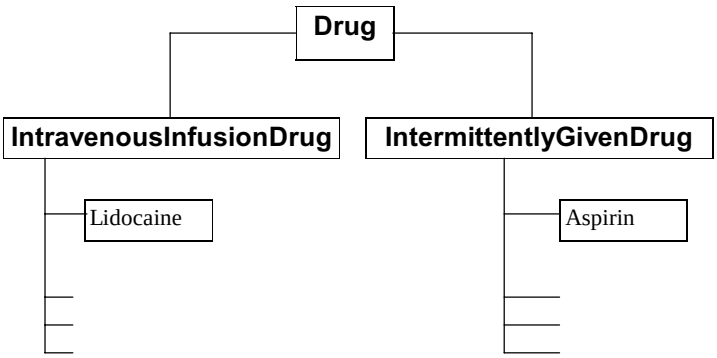


Рис. 1. Ієрархія класів для представлення хіміопрепаратів.

Статичні елементи класу

Більшість властивостей, таких як m_dDose, m_dDosingInterval — унікальні для об'єктів. Але деякі властивості є спільними для усіх об'єктів даного класу. Наприклад, уявимо, що весь аспірин, який використовується в даній клініці, має єдину торговельну марку, наприклад, Regular Strength Bayer® Enteric 500 Aspirin [OTC]. Крім того, ми бажаємо вести облік усіх використань аспірину. Це можна реалізувати за рахунок обліку усіх об'єктів класу Aspirin. Введемо означення.

Елементами класу називаються властивості, що спільно використовуються всіма об'єктами даного класу.

Елементами об'єкта називаються властивості, що не є спільними для всіх об'єктів класу.

Елементи класу означуються за допомогою ключового слова static. З цієї причини вони називаються **статичними** елементами, на відміну від елементів об'єкта, які називаються **нестатичними**.

Використовуючи поняття статичних елементів, маємо наступне оголошення класу Aspirin:

```
class Aspirin extends IntermittentlyGivenDrug
{
    static String m_sBrandName; // торговельна назва

    // ... решта означень класу
}
```

Якщо тепер ми змінимо значення торговельної назви для одного об'єкта, то вона зміниться для усіх об'єктів цього ж класу:

```
void SomeFunc( )
{
    Aspirin asDrug1 = new Aspirin( );
    Aspirin asDrug2 = new Aspirin( );
    asDrug1.m_sBrandName = "Extra Strength Bayer® Enteric
500 Aspirin [OTC]";
```

```
    String str = asDrug2.m_sBrandName;
    // значення змінної str — "Extra Strength Bayer® Enteric
500 Aspirin [OTC]"
}
```

Конструктори класу

Конструктор — це спеціальний метод класу, який автоматично виконується кожного разу, коли створюється об'єкт класу. Його мета — ініціалізація об'єкта в початковий стан. Конструктор відрізняється від усіх інших методів тим, що він має таке ж ім'я, що й клас. Розглянемо приклад:

```
class Aspirin extends IntermittentlyGivenDrug
{
    static String m_sBrandName; // торговельна назва
    static int m_nCounter; // лічильник використань аспірину

    Aspirin( )
    {
            m_nCounter++; // інкрементація
            // ... решта ініціалізації
    }
    // ... решта означень класу
}
```

Зверніть увагу, що на відміну від інших методів, конструктор не має типу значення, що повертається.

Специфікатори керування доступом

Класи, елементи класів, елементи об'єктів можуть бути доступними або недоступними у різних файлах, класах, пакетах завдяки спеціальним ключовим словам (специфікаторам), що вказуються перед ними:

Специфікатор	Призначення
не вказується або ж friendly	елементи доступні іншим класам в тому ж самому пакеті (застосовний і до класів)
public	елементи доступні для інших класів в довільному пакеті (застосовний і до класів)
private	змінні і методи доступні лише всередині власного класу (цей специфікатор незастосовний до класів)
protected	елементи доступні в методах того ж класу або усіх його підкласів. Непов'язані класи не мають до них доступу, навіть якщо вони з того ж пакета

ОСНОВИ ВІЗУАЛЬНОГО ПРОЕКТУВАННЯ В СЕРЕДОВИЩІ DELPHI

Замість передмови до Delphi

Для багатьох людей мови програмування, такі як C++, є такими ж недоступними, як класична грецька мова. Чи існує менш виснажливий спосіб писати якісне програмне забезпечення для Windows?

Так. Саме разом з Delphi ви відчуєте себе впевненими у програмуванні для Windows. Названа як система швидкої розробки додатків, або ж RAD, Delphi є генератором ефективного коду, конструктором візуальних додатків і інструментом для баз даних, який є легким в опануванні і водночас потужним у використанні.

Для опанування Delphi у наших рамках зовсім не потрібно бути фахівцем в галузі Computer Science, та якщо ви знаєте Pascal, C, C++, або Visual Basic, це Вам стане у пригоді. Чисельні покрокові інструкції демонструватимуть впровадження візуальних компонент у мову Object Pascal.

Вимоги до комп'ютера

Для опанування даного розділу вам необхідна інсталяція на комп'ютері редакцій Delphi, таких як Desktop або ж Client/Server. Звісна річ, вимагається операційна система Windows NT або ж Windows 95. Редакція Delphi Desktop вимагає мінімум 6MB оперативної пам'яті. Редакція Client/Server вимагає 8MB. В даному викладі можна користуватися довільною редакцією. Повна інсталяція Delphi та пов'язаних із нею інструментальних засобів охоплює приблизно від 60MB до 80MB простору жорсткого диска. Вам не потрібні файли вхідного коду візуальних компонент, що надаються редакцією Client/Server (також можна отримати незалежно); однак досвідчені розробники завжди потребують ці файли.

Довільний персональний комп'ютер з процесором 80386, 80486 або Pentium, що має, принаймні, від 8MB до 12MB оперативної пам'яті, дисплей VGA і 200MB або більше жорсткого диска створює чудові умови для інструментальної системи Delphi.

Хоча Windows 3.1 може функціонувати на персональних комп'ютерах з процесорами 80286, Delphi вимагає процесора 80386 або пізнішої моделі. З приводу процесорів можна додати ще таке: Delphi автоматично генерує підправлений код для процесорів Pentium, щоб уникнути помилок з числами з плаваючою точкою. Щоб уможливити таку властивість, виберіть команду Project — Options, клацніть закладку сторінки Compiler у діалоговому вікні, що з'явилося, і встановіть прапорець Pentium-safe FDIV.

Візуальне середовище Delphi

Як і для більшості програмного забезпечення, найкращий шлях для вивчення Delphi є шлях його використання. Зараз ми спробуємо проаналізувати візуальне середовище Delphi. Запустимо або ж перейдемо у Delphi. Рисунок 1 показує Delphi, що працює у Windows 95. Якщо ж Ваш екран не нагадує рисунка, виберіть команду File|New Project.

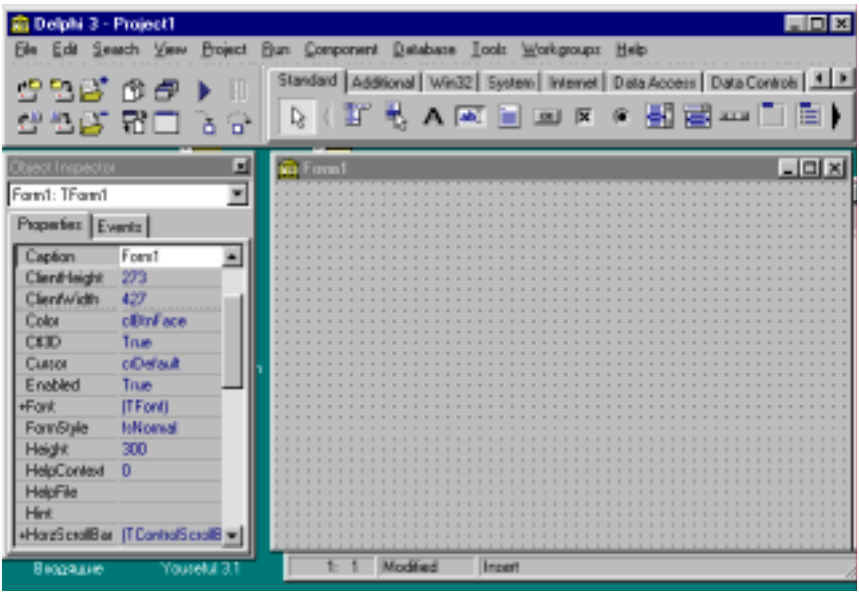


Рис. 1. Delphi в операційній системі Windows 95.

Існує кілька головних елементів інтегровного середовища розробки Delphi (integrated development environment, IDE), починаючи із верхнього лівого кута на малюнку і продовжуючи за годинниковою стрілкою, цими елементами є:

головне меню — це є стандартне меню у стилі Windows.

Палітра компонент — ця палітра включає позначки (іконки), що представляють компоненти у VCL. Клацніть кнопку на Палітрі компонент, щоб вибрати візуальний компонент, а потім клацніть мишкою всередині вікна форми, щоб вставити об'єкт компонента. Перемістіть курсор мишки над довільною кнопкою компонента, але не клацайте кнопку мишки. Через деякий час Delphi виведе підказку з іменем компонента. Перемістіть мишку, щоб переглянути підказки інших компонент. Цим методом можна скористатися для знаходження довільних компонент.

Закладки сторінок палітри — щоб переглянути інші категорії компонент, клацніть одну із закладок на Палітрі VCL. Наприклад, клацніть закладку сторінки Dialogs, щоб вивести компоненти Delphi типу діалогове вікно.

Форма — у більшості додатків форма є візуальним представленням головного вікна програми. Однак форма може також представляти інші вікна — наприклад, діалогове вікно або ж вікно нащадка у програмному забезпеченні, що грунтується на багатодокументному інтерфейсі (multiple document interface, MDI). Прості програмки можуть мати лише одну форму; складні додатки можуть мати десятки. Сітка на формі допоможе Вам вирівняти компоненти, вставлені у вікно форми. Ця сітка не з'явиться у кінцевому додатку.

Вікно модуля — це вікно показує текст програмування, виконаного на Pascal, що пов'язаний із кожною формою додатка. Delphi автоматично створює це програмування, до якого ви можете додавати Pascal-твердження, що виконують дії для подій, таких, як клацання по командах меню або кнопках. Це вікно ви можете використовувати також для редагування інших Pascal-модулів і текстових файлів. Для вибору відкритого файла клацніть відповідну закладку сторінки зверху вікна модуля.

Закладки сторінок Property та event — клацніть одну з цих закладок сторінки зверху вікна Object Inspector, щоб пере-

йти до властивостей чи подій компонента або форми. Закладка property представляє атрибут компонента, такий як розмір кнопки або ж шрифт текстової мітки. Закладка event представляє дію, таку як клацання мишкою або ж натискання клавіші.

Інспектор об'єктів (Object Inspector) — це вікно показує усі властивості і події для одного або більше вибраних компонент або форм. Вікно Object Inspector є одним з найважливіших інструментальних знарядь Delphi.

Швидкі кнопки — це є кнопки типу "вкажи-і-клацни" для вибору команд меню. Так, як і на палітрі компонент, ви можете вивчити, що кнопка робить, викликаючи підказку з іменем кнопки. Використання цих кнопок призводить до економного витрачання часу. Наприклад, хоча існують інші шляхи для запуску програм, найшвидшим методом є клацання швидкої кнопки Run (трикутник, що вказує праворуч).

В подальшому ви дізнаєтеся більше про кожне з вікон Delphi, команди та інші можливості. Ви також познайомитеся з багатьма іншими вікнами, такими як диспетчер проекту, огляд об'єктів (object browser), інтегрований відлагоджувач, редактор коду, редактор рисунків, конструктор меню.

Приготування для нового додатка

Перший крок після початку нового додатка — це надати йому ім'я. Найкраще це зробити безпосередньо після виконання File — New Project. Найменування нового проекту настільки швидко, як це можливо, запобігає від збереження проекту у власних каталогах Delphi під іменами файлів за припущенням, що призводить до нерозумного витрачання дискового простору і може спричинити проблеми через спосіб, в який Delphi використовує ім'я файла модуля, щоб створити програмні твердження та оголошення.

Типовий проект Delphi складається з файлів кількох типів. Деякі файли містять текст, інші містять двійкові значення, "бітмапи" і виконуваний код. Оскільки кожен додаток складається з багатьох файлів, розумно створити окремий каталог на вашому жорсткому диску для кожного нового проекту. В такому разі ви зможете легко копіювати файли додатка на гнучкий диск або

мережевий пристрій для резервного зберігання, а також знищувати старі додатки, просто знищивши їх каталоги.

Виконавши наступні дії, ви створите ваш перший додаток в Delphi:

1. Використовуючи Explorer для Windows 95, створіть новий каталог C:\Projects для зберігання проектів. Виберіть цей же каталог і створіть його підкаталог на ім'я Hello. Цим ви створите шлях C:\Projects\Hello для зберігання файлів проекту.
2. Виберіть File|New Project. Якщо в цей час вже завантажено і змінено інший проект, Delphi надасть вам можливість зберегти проект або ж усунути внесені зміни. Якщо ви побачите діалог Confirm, виберіть Yes, щоб зберегти попередній проект, або ж No, щоб усунути внесені зміни.
3. Виберіть File — Save Project. У діалозі типу file-save, що з'явиться, змініть каталог на C:\Projects\Hello, що ви створили на першому кроці, і введіть main у поле File Name. Щоб зберегти форму у текстовому файлі під іменем Main.Pas, клацніть кнопку OK або натисніть Enter. Delphi автоматично дописує розширення імені файла. Pas, що вказує на файл, що містить програмування, здійснене на Pascal.
4. Далі Delphi запросить ім'я проекту в іншому діалозі типу file-save. Введіть hello у поле File Name і виберіть OK або натисніть Enter. Цим ви створите головний файл проекту Hello.Dpr. Delphi автоматично надає розширення імені файла .Dpr, яке вказує на проект Delphi.

Якщо ж Delphi не запросить ім'я проекту на 4-му кроці, найімовірніше, що ви помилково вибрали команду File — Save File, яка зберігає лише один файл. Якщо це трапилося, повторіть кроки 3 і 4 та наочно виберіть команду File|Save Project.

Збереження проекту в окремому каталозі перед додаванням будь-якого програмного коду зробить модулі вашого додатка добре організованими. Щоб досягти кращих результатів, ніколи не зберігайте кілька проектів в одному каталозі.

Задання надпису вікна

За припущенням, Delphi надасть формі-вікну проекту Hello надпис Form1. Надпис з'являється у рядку заголовка під верх-

ньою рамкою вікна. За допомогою наступних кроків можна змінити надпис на заголовок програми.

1. Зверху вікна Object Inspector клацніть закладку сторінки Properties, щоб вивести властивості форми програми.
2. Виберіть властивість Caption (вона вже може бути вибрана за припущенням).
3. Щоб змінити властивість, просто введіть або виберіть нове значення у стовпчику праворуч від назви властивості. Наприклад, змініть значення Caption з Form1 на Hello Delphi Programmer! Просто розпочніть введення — вам не потрібно помічати текст перед цим. Вам також не потрібно натискати Enter після введення. Якщо ви все ж натиснули Enter, поле просто помітиться ще раз. Зверніть увагу, що надпис вікна форми змінюється одночасно із введенням нового значення.
4. Збережіть проект, вибравши команду File — Save Project. Оскільки ви вже надали імена файлів, Delphi виконає цю команду безпосередньо — тобто, не виведе діалогу типу file-save (щоразу, коли ви додаєте нову форму до проекту, ви будете бачити цей діалог знову, коли ви зберігатимете проект).

Зауваження. Ви можете зберегти проект, натиснувши "швидку" кнопку Save project (це швидше, ніж вибирати команду File|Save Project).

Запуск додатка

Вірите в це ви чи ні, але ви вже закінчили програмування вашого першого додатка Delphi. Щоб закінчити процес розробки і запустити програму, виконайте такі кроки:

1. Натисніть F9 або виберіть команду Run з меню Run. Або ж клацніть швидку кнопку Run. Пробуйте запускати ваш додаток багаторазово під час його розробки, щоб випробувати нові програмні твердження. Вам не слід завершити проект до того, як можна буде його запускати.
2. Після кількох секунд на екрані з'явиться вікно вашого першого додатка.
3. Щоб вийти із додатка, клацніть кнопку закриття вікна у правому верхньому куті. Якщо вам більше до вподоби клавіатура, натисніть Alt+F4, щоб закрити додаток і повернутися у Delphi.

Після запуску додатків із середовища завжди слід закривати їх, щоб повернутися в режим програмування. Якщо ви помилково залишите додаток відкритим, ви не зможете вибрати багато команд Delphi. Отже, якщо Delphi починає не відповідати на команди, одна з причин, що може бути — працюючий додаток, що ви забули закрити.

Натисніть F9, щоб перезапустити Hello. Зверніть увагу, що на цей раз вікно додатка з'явиться швидше, ніж ви вперше запускали програму. Delphi знає, що ви не зробили жодних змін у проекті, отже, вона просто перезавантажить файл коду додатка. Закрийте зараз вікно програми Hello.

Ви також можете використовувати Windows 95 Explorer, щоб запустити додаток Delphi. Наприклад, відкрийте Explorer і виберіть каталог C:\Projects\Hello. Двічі клацніть Hello.Exe або помітьте це ім'я файла і натисніть Enter. Не забудьте закрити програму, щоб продовжити роботу.

Використайте Windows 95 Explorer, щоб запустити кілька екземплярів (інстанцій) даного додатка. Наприклад, спробуйте запустити дві або більше копій Hello. Ви можете запускати лише один екземпляр програми з Delphi. Щоб запустити кілька екземплярів, ви повинні скористатися Explorer. Закрийте усі примірники Hello, щоб продовжити роботу далі.

Додаток Delphi це повнофункціональна програма для Windows — ви запускаєте його таким же чином, як ви запускаєте інші додатки. Ви можете клацати і перетягувати файл коду додатка на робочий стіл і тоді двічі клацати одержану іконку. Або ж ви можете запускати додаток, вказавши ім'я файла його коду у команді Run системного меню Windows 95.

Компіляція та зв'язування (лінкування) коду

Коли ви запускаєте додаток, натискаючи F9, Delphi компілює і зв'язує програму, щоб створити файл виконуваного коду. Тут відбуваються дві ключові події. По-перше, компілятор Delphi переводить текст програми у двійковий код. Далі лінкувальник (програма, що з'єднує) комбінує цей код з іншими модулями, які вимагаються для дій запуску програми та інших задач. Результа-

том компіляцій та зв'язування проекту Delphi є завершений файл виконуваного коду, який має те ж ім'я, що й проект, але розширення імені цього файла — .Exe.

На відміну від інших систем візуального програмування, Delphi генерує дійсно природний код, що означає те, що кінцева програма не вимагає інтерпретатора під час виконання. .Exe-файл коду завершений на 100 %. Лише цей файл ви повинні надавати споживачам вашої програми.

Щоб зкомпілювати та зв'язати додаток, але не запускати його, натисніть Ctrl+F9 або виберіть команду Compile — Syntax Check. Ви можете скористатися цими методами, щоб перевірити, що програма не містить помилок, таких як "описка".

Програмування за допомогою компонент

Перед тим, як іти далі, давайте зупинимося і зробимо короткий огляд. Зараз ми вже навчилися робити в Delphi три важливі речі.

1. Створювати та зберігати новий проект.
2. Змінювати властивість, таку як Caption для форми Delphi.
3. Компілювати. Зв'язувати і запускати додаток, натискаючи F9.

Ми використовуватимемо ці методики при написанні кожного нового додатка. Звичайно, програми, які не роблять нічого, окрім виведення заголовка, не приносять жодної користі. Для того, щоб додаток Hello міг щось виконувати, ви можете вставити візуальний компонент у вікно програми. Про це піде мова далі.

Вставка візуальних компонент

За допомогою наступних кроків здійснюється вставка об'єкта візуального компонента у вікно програми Hello:

1. Закрийте працюючу програму Hello. Якщо слід повернутися у Delphi. Виберіть File — Open Project або клацніть "швидку" кнопку Open project. (Зробіть це для набуття досвіду, навіть, якщо Hello вже відкрито). Коли з'явиться діалог Open Project, змініть при потребі каталоги і виберіть Hello.Dpr. Натисніть Enter або клацніть OK, щоб вибрати проект (за допомогою цього методу будуть відкриватися будь-які проекти. На цьому

місці ви можете припинити роботу, зберегти проект і потім перезавантажити його, коли будете готовими продовжити).

2. Клацніть закладку сторінки Standard на палітрі компонент і потім виберіть компонент Button, на якому знаходиться позначка Ok.
3. Перемістіть вказівник мишки у форму, а потім клацніть один раз, щоб вставити об'єкт Button у вікно. Точне розміщення не має значення.
4. Переконайтеся, що вибрано об'єкт Button — ви повинні бачити довкола нього квадратні керма (хендли). Якщо об'єкт не вибрано, клацніть його ще раз. За припущенням, Delphi надасть об'єкта ім'я Button1. Щоб надати більш опосередковану назву, виберіть властивість Name у вікні Object Inspector (вона може вже бути вибрана) і наберіть CloseButton. Це одне слово — в імені об'єкта не можна набирати пропусків. Натисніть Enter.
5. Зверніть увагу, що властивість Caption та текст кнопки також зміниться у CloseButton. Це нормально, але в багатьох випадках ви бажатимете, щоб властивості об'єкта компонента Name та Caption відрізнялися. У цьому випадку, наприклад, виберіть властивість Caption, і наберіть Close.
6. При потребі один раз клацніть об'єкт CloseButton. Виберіть закладку сторінки Events зверху вікна Object Inspector, яка наводить список усіх дій, що може виконати кнопка. Двічі натисніть порожнє місце праворуч події OnClick. У вікні модуля з'явиться Main.Pas і Delphi помістить мерехтливий курсор між ключовими словами begin та end. Наберіть наступне твердження: Close;
7. Щойно ви запрограмували процедуру, що називається обробником події для події кнопки OnClick. Переконайтеся, що завершили це твердження символом ";". Порівняйте текст у Вашому вікні із лістингом у додатку.
8. Збережіть проект і потім запустіть його, натиснувши F9. Зараз ви можете натиснути кнопку OK, щоб закрити вікно. Оскільки це головне вікно програми, то його закриття також завершує додаток (рис. 2).

Рис. 2. Головне вікно програми Hello.

Команда File — Open File відкриває індивідуальні файли. Щоб завантажити завершений додаток Delphi, завжди використовуйте File — Open Project або клацніть "швидку" кнопку Open project.

Попередні кроки продемонстрували три важливих аспекти програмування з Delphi.

1. Ви вставляли об'єкт компонента Button у вікно форми.

2. Ви модифікували властивості об'єкта, щоб змінити їх внутрішні значення Name та Caption.

3. Ви запрограмували одну з подій об'єкта, щоб виконати дію під час виконання у відповідь на вибір кнопки.

Інтерфейс Delphi і усі її компоненти, окрім елементів керування VBX (компоненти, вперше розроблені для Microsoft Visual Basic і написані на мові C++), були написані з використанням Delphi. Це справжній доказ складності задач, які дана інструментальна система дозволяє розв'язати.

Відлагодження програмних речень

Відлагодження — виявлення помилок кодування.

Відлагодження — це, напевно, найскладніший період розробки додатка. Програмісти витрачають багато часу на виявлення так званих bugs ("жучків"). Звичайно, що Delphi не може запобігти появі таких "жучків", але вона може допомогти уникнути багатьох типових помилок, надаючи вам потужне візуальне середовище, що автоматично створює більшість вашого додатка. На жаль, все ще залишається багато простору для помилок і тому нечесно сказати, що процес розробки додатків в Delphi повністю гладкий.

Команди відлагодження Delphi також корисні для дослідження того, як програма працює. За допомогою наступних кроків ви дізнаєтеся, як використати деякі можливості відлагодження у Delphi.

1. При потребі відкрийте проект Hello.
2. Якщо ви ще не вставили кнопку Close у форму, зробіть це зараз за допомогою попередніх інструкцій.
3. Знайдіть процедуру TForm1.CloseButtonClick у вікні модуля Main.Pas. Вставте речення вище Close, щоб вивести повідомлення. Дві стрічки між begin та end повинні виглядати так:
   ShowMessage('На цьому робота з додатком завершується');
   Close;
4. За звичайних обставин ви натискаєте F9, щоб відкомпілювати, злінкувати та запустити програму. Цього разу натисніть F8, щоб вибрати команду Delphi Step Over. Коли з'явиться головне вікно, клацніть кнопку Close. Замість завершення програми Delphi робить у ній паузу всередині процедури CloseButtonClick. Це приклад покрокового проходження програми, одного з найважливіших засобів дослідження, що ви можете використати під час розробки програми.
5. Натисніть F8 ще два рази, щоб перейти через речення ShowMessage і виконати його. Віконце з кнопкою OK з'явиться на екрані. Клацніть цю кнопку. Знову, оскільки ви натиснули F8, Delphi зробить паузу у програмі на наступному реченні, а саме Close.
6. Натисніть F9, щоб виконати речення Close і виконати програму до кінця. Після натискання F8 для покрокового проходження одного або більше тверджень ви, переважно, завершуватимете програму, натиснувши F9.

Інший спосіб крокування програмою полягає у встановленні точки зупинки. Коли програма досягає поміченого речення, Delphi зупиняє додаток перед виконанням цього твердження. За допомогою наступних дій ви ставитимете точку зупинки і проходимете через програму.

1. Перемістіть мерехтливий курсор у речення ShowMessage у Main.Pas і виберіть Run — Add Breakpoint. Клацніть кнопку New у діалозі, що з'явиться або просто натисніть Enter. Помічене речення зараз помітиться червоним кольором.

2. Натисніть F9, щоб запустити програму. Коли ви натиснете кнопку Close, Delphi призупинить код на поміченому реченні. Натисніть F8, щоб далі іти покроково і вивести повідомлення. Закрийте діалог повідомлення, клацнувши кнопку OK і натисніть F9, щоб запустити програму на завершення.
3. Спробуйте 2-й крок ще раз але цього разу натисніть для завершення програми Alt+F4. Оскільки цим ви не виконуєте процедуру CloseButtonClick, то програма завершується без паузи на точці зупинки.
4. Завжди намагайтеся запустити програму для завершення. Якщо вам це не вдається, виберіть Run — Program Reset і запустіть знову. Це може спричинити переривання процедур, знищення різних ресурсів з пам'яті, що, можливо, заставить вас вийти і перезапустити Windows.
5. Щоб встановлювати чи знімати точки зупинки, помістіть вказівник мишки у крайню ліву позицію напроти поміченого речення і клацніть один раз ліву кнопку. Це — найлегший спосіб встановлення точки зупинки. Ви можете також використовувати команду View — Breakpoints, щоб додавати, знищувати, відміняти і виконувати інші дії над точками зупинки. Виберіть цю команду і клацніть правою кнопкою мишки у вікні Breakpoint list, щоб отримати список наявних команд.

Лістинг: Обробник події OnClick об'єкта CloseButton.

```
procedure TForm1.CloseButtonClick(Sender: TObject);
begin
  Close;
end;
```

РОЗРОБКА МУЛЬТИМЕДІА-ДОДАТКІВ В МЕДИЦИНІ. МЕДІАПЛЕЄР

Мультимедіа. Вимоги до апаратного та програмного забезпечення

Сучасний комп'ютер отримав ще одну властивість — мультимедіа. Вона вказує на здатність використовувати засоби обробки високоємнісної інформації. На сьогодні сюди належать — графіка, звук та відео. Windows-додатки, орієнтовані на використання засобів мультимедіа, отримали назву мультимедіа-додатків.

Даючи таке загальне визначення, слід сказати, що в професійній діяльності лікар найчастіше може мати справу з підмножиною мультимедіа, що включає:

1. Показ відео у форматі Microsoft Video for Windows (AVI).
2. Показ стиснутого відео у форматі MPEG (MPG).
3. Відтворення звуків з MIDI і WAVE файлів.

Обробка аудіо- та відеоінформації вимагає великих обчислювальних можливостей комп'ютера та спеціального програмного забезпечення. Тому для програвання файлів мультимедіа може знадобитися наявність деякого устаткування і програмного забезпечення, а саме:

1. Для обробки потокової аудіо- та відеоінформації ПК повинен мати потужний процесор (наприклад, Pentium II, III, IV, Athlon) та збільшений об'єм оперативної пам'яті (64-128 Mb).
2. Для відтворення звуків потрібна звукова карта.
3. Для відтворення відео потрібна відеокарта обсягом 4 Mb.
4. Для відтворення AVI у Windows потрібно встановити ПЗ Microsoft Video.
5. Для відтворення MPEG потрібно встановити відповідні відео- та аудіодекодери.

Слід зауважити, що формат MPEG має кілька рівнів компресіїї аудіо-відеоінформації (Layer I-III). Для кожного з них є окремий декодер. Декодери аудіо- та відеопотоків також розділені по окремих бібліотеках.

Все необхідне ПЗ вже є в комплекті з ОС Windows 98. Однак слід перевірити налаштування драйверів мультимедіа в

Windows, що здійснюється за допомогою Панелі Керування (Control Panel).

Робота з мультимедіа в Delphi

Чи не найбагатші можливості для розробки мультимедіа-додатків має інструментальна система Delphi. Delphi дозволяє легко і просто включати в програму такі мультимедійні об'єкти, як звуки, відео і музику. Це можна зробити двома шляхами:

- використовуючи вбудовані у Delphi компоненти TAnimate або TMediaPlayer, що дають доступ до всіх основних можливостей програмування мультимедіа;
- використовуючи зовнішній Windows-додаток для відтворення даного типу мультимедіа-файла.

Ці два способи і будуть розглянуті далі.

Використання компонента TAnimate для відтворення AVI-файла

Компонент TAnimate знаходиться на сторінці Win32 Палітри Компонентів Delphi. Даний компонент призначений для відтворення відеозаписів у форматі Audio Video Interleaved (AVI). Однак даний компонент має кілька суттєвих обмежень:

- відсутня можливість відтворення звуку;
- AVI файл для відтворення повинен бути нестисненим або стисненим методом run-length encoding — групового кодування (RLE).

Компонент TAnimate може відтворювати AVI-відеозаписи з AVI-файлів, AVI-ресурсів, а під управлінням ОС Windows 95 або NT4 — з бібліотеки Shell32.dll.

Помістивши компонент на форму, Ви побачите, що Інспектор Об'єктів містить властивість "FileName". Клацніть двічі на цій властивості і виберіть ім'я файла з розширенням AVI. Якщо формат AVI-вірний, у формі буде відображено перший кадр анімації. Встановіть значення властивості Open у True і можна побачити анімаційну послідовність.

Крім вибору зовнішнього AVI-файла, компонент TAnimate дає можливість вибрати один з готових AVI-записів. Це здійсню-

ється за допомогою властивості Common AVI, яка містить список AVI-записів.

Це і є основним призначенням компонента TAnimate — створення форми з елементами анімації без необхідності керувати процесом відтворення. Наприклад: візуалізацій процесів копіювання, перенесення, знищення, пошуку файлів в Windows, створення спеціалізованих елементів управління (кнопок з анімацією) і інших подібних елементів інтерфейсу.

Медіаплеєр на основі компонента TMediaPlayer

Компонент TMediaPlayer знаходиться на сторінці System Палітри Компонентів Delphi. В даний компонент інкапсульовані методи динамічної бібліотеки Microsoft Multimedia Extensions для Windows (MMSYSTEM.DLL), що призначена для відтворення мультимедіа-файлів. Компонент TMediaPlayer дуже простий у використанні. Усе, що потрібно знати — це те, що компонент називається TMediaPlayer, і що він дає доступ до набору підпрограм, створених Microsoft і названих Media Control Interface (MCI). Ці підпрограми дають програмісту простий доступ до широкого кола пристроїв мультимедіа.

Власне робота з TMediaPlayer інтуїтивно зрозуміла й очевидна. З одного боку, це дає можливість кожному створювати мультимедіа додатки. З іншого боку, можна знайти, що в компоненті реалізовані не всі можливості. Якщо необхідно використати низькорівневі функції, то доведеться використати доволі складні конструкції на мові Delphi.

Для початку давайте створимо новий проект, потім помістимо компонент TMediaPlayer (с. System Палітри) на форму, як показано на рис. 1.

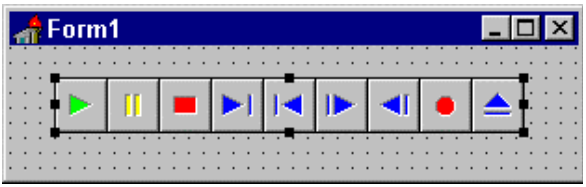


Рис.1. Компонент TMediaPlayer на формі.

Компонент TMediaPlayer оформлений як панель керування пристроєм із кнопками. Як і на магнітофоні, тут є кнопки "відтворення", "перемотування", "запис" і ін.

Помістивши компонент на форму, Ви побачите, що Інспектор Об'єктів містить властивість "FileName" (рис. 2). Клацніть двічі на цій властивості і виберіть ім'я файла з розширенням AVI, WAV чи MID. На рис. 2 обраний AVI файл 01.MPG, що знаходиться в папці C:\Video. Далі потрібно встановити властивість AutoOpen у True.

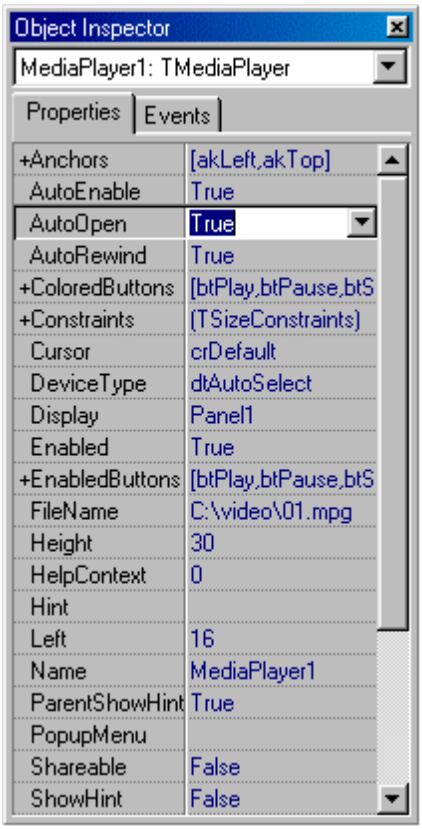


Рис. 2. Властивості TMediaPlayer в Інспекторі Об'єктів.

Після виконання цих кроків програма готова до запуску. Запустивши програму, натисніть зелену кнопку "відтворення" (крайня ліворуч) і Ви побачите відеоролик (якщо вибрали AVI або MPG) чи почуєте звук (якщо вибрали WAV чи MID). Якщо цього не відбулося, і з'явилося повідомлення про помилку, то можливі два варіанти:

- введено неправильне ім'я файла;
- не налаштовано правильним чином мультимедіа в Windows. Це означає, що або немає відповідного "заліза", або не встановлені потрібні драйвери. Установка і настроювання драйверів здійснюється в Control Panel, вимоги до "заліза" наведено вище, або можна скористатися будь-якою книгою з мультимедіа.

Отже, є можливість програвати AVI, MPG, MIDI і WAVE файли, просто вказуючи ім'я файла.

Ще одна важлива властивість компонента TMediaPlayer — Display. По замовчуванні воно не заповнене і відео відтворюється в окремому вікні. Однак, як екран для показу ролика, можна використовувати, наприклад, панель. На форму потрібно помістити компонент TPanel, забрати текст з властивості Caption. Далі, для TMediaPlayer, у властивості Display вибрати зі списку Panel1. Після цього треба запустити програму і натиснути кнопку "відтворення" (рис. 3).

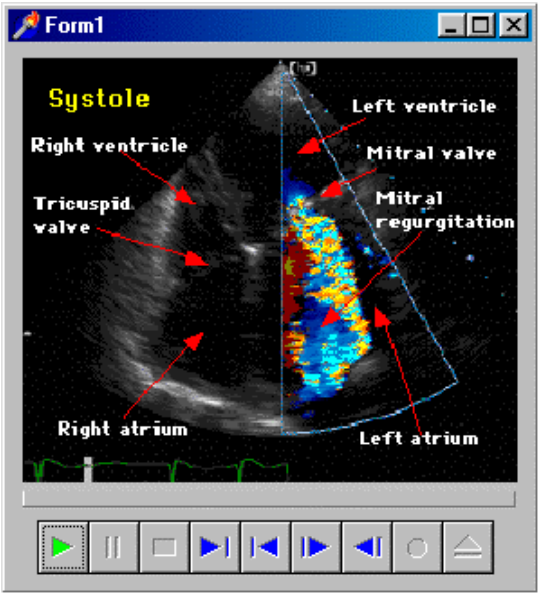


Рис. 3. Відтворення MPG на панелі.

Два способи використання компонента TMediaPlayer

- Іноді доводиться показувати користувачам простий шлях для програвання максимально широкого кола файлів. Це означає, що користувачу потрібно буде дати доступ до жорсткого диска чи CD-ROM, і потім дозволити йому вибрати і відтворити належний файл. У цьому випадку на формі звичайно розташовується TMediaPlayer, що дає можливість керування відтворенням, а також елементи інтерфейсу для доступу до дисків, вибору папок та файлів.
- Іноді програміст може захотіти сховати від користувача існування компонента TmediaPlayer, тобто, відтворити звук чи відео без того, щоб користувач піклувався про їхнє джерело. Зокрема, звук може бути частиною презентації. Наприклад, показ якого-небудь графіка на екрані може супроводжуватися поясненням, записаним у WAV файл. При відтворенні презен-

тації користувач навіть не знає про існування TMediaPlayer. Він працює у фоновому режимі. Цей компонент робиться невидимим (Visible = False) і керується програмно.

Використання асоційованих програм для відтворення мультимедіа-файлів

Другий спосіб відтворення аудіо-, відеозаписів — використання асоційованих з ними Windows-додатків. В складі ОС Windows 98 є в наявності стандартні програми для відтворення аудіо- та відеозаписів. Серед них Универсальный Проигрыватель, Windows MediaPlayer. Ці програми автоматично запускаються при подвійному натисненні на файл з розширенням імені відповідного типу.

Такий шлях дещо складніший, ніж при використанні компонента TMediaPlayer, однак дає більш широкі можливості. Зокрема, можна використати для відтворення файлів аудіо- та відеозаписів як асоційовані з ними програми, так і деякі спеціалізовані.

Для реалізації такого підходу було розроблено новий компонент TSurgeryOperationsViewer. Даний компонент дає можливість переглянути список описів відеозаписів у вигляді дерева та запустити вибраний файл. Як приклад, відео використовуються при відеозаписі хірургічних операцій.

Нижче наведено опис компонента TSurgeryOperationsViewer. Повністю лістинг даного модуля представлено у додатку.

```
TSurgeryOperationsView = class(TTreeView)
private
  { Private declarations }
  ini: TIniFile;
  strs, first_substrs, second_substrs, third_substrs: TStrings;
protected
  { Protected declarations }
public
  { Public declarations }
  procedure DblClick; override;
```

```
    procedure ShowItems;
  published
    { Published declarations }
  end;
```

Як видно з лістингу, даний компонент є наслідувачем компонента TreeView, який призначений для відображення деревоподібних структур. Використовуються всі функції об'єкта-предка і додаються нові можливості — робота з ini-файлами, та запуск файла за вибраним елементом списку.

Метод ShowItems призначений для зчитування структури відеозаписів з ini-файла та відображення їх описів в вигляді дерева. Для відображення описів використовуються засоби об'єкта-предка, тоді як для роботи з ini-файлом написано спеціальні процедури.

Метод DblClick призначений для запуску вибраного файла відеозапису за вибраним в списком опису. Головну роль в цьому методі відіграє функція FileExecute, за допомогою якої запускається на виконання файл відеозапису.

Нижче наведено лістинг файла **Video.ini**, який містить назви операцій та відповідні їм відеофайли.

```
[Лапароскопічні операції]
Video1=Введення другого основного маніпуляційного троакара діаметром 11 мм||01.mpg
Video2=Введення допоміжних троакарів||02.mpg
Video3=Інтраопераційна доплеросонографія печінки||03.mpg
Video4=Захоплення жовчного міхура та виведення його в поле зору||04.mpg
Video5=Пункція жовчного міхура||05.mpg
```

В даний файл можна додати опис відеороликів будь-якого змісту (при наявності відповідного .mpg-файла), достатньо лише дотримуватися вказаних правил запису.

Порядок розробки проекту “ Медіаплеєр”

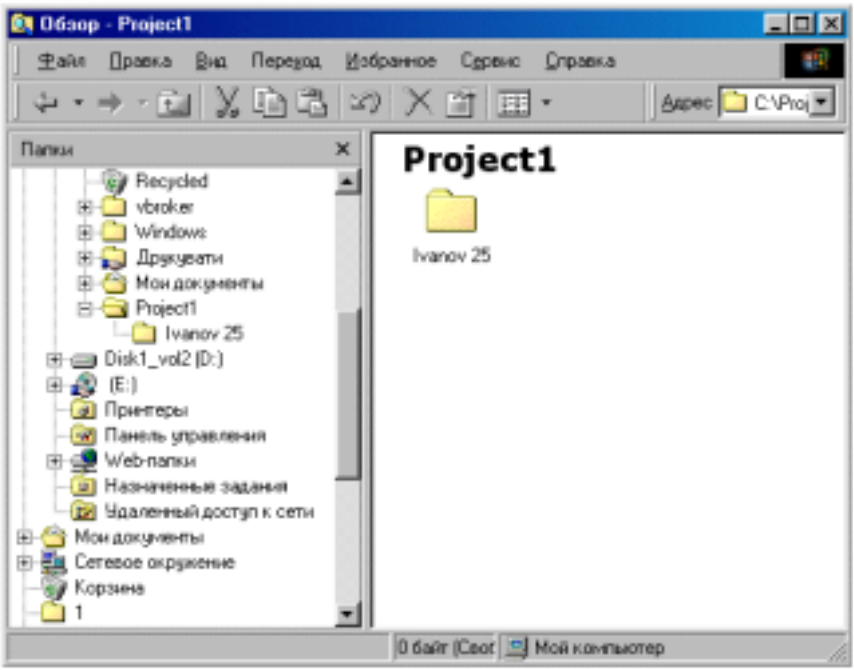
Необхідне програмне забезпечення (ПЗ):

1. Інструментальна система Delphi не нижче версії 3.
2. Інстальована бібліотека FMXUtils (знаходиться в папці Docs інсталяційного пакета Delphi).
3. Інстальований компонент SurgeryOperationsViewer (з питань інсталяції компонентів див. додаток).

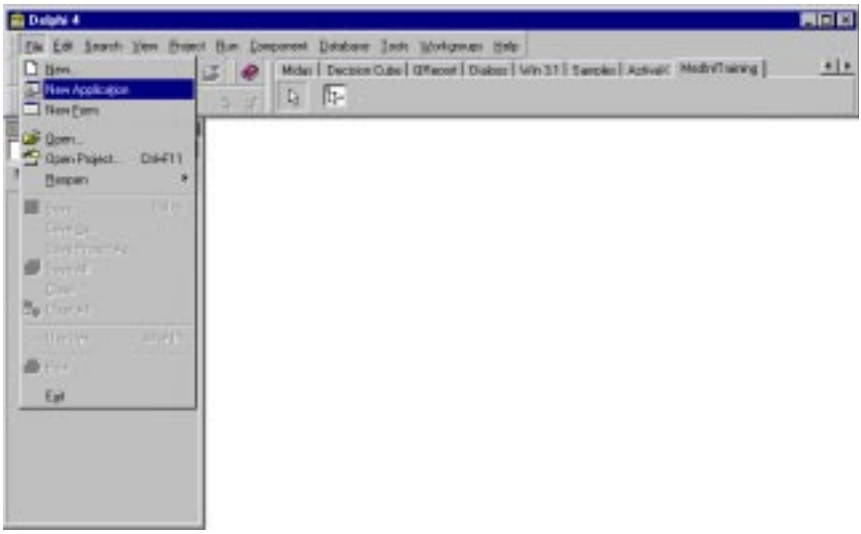
Розробка демонстраційного проекту SurgeryOperationsViewer здійснюється таким чином:

1. Створіть каталог, заданий шляхом

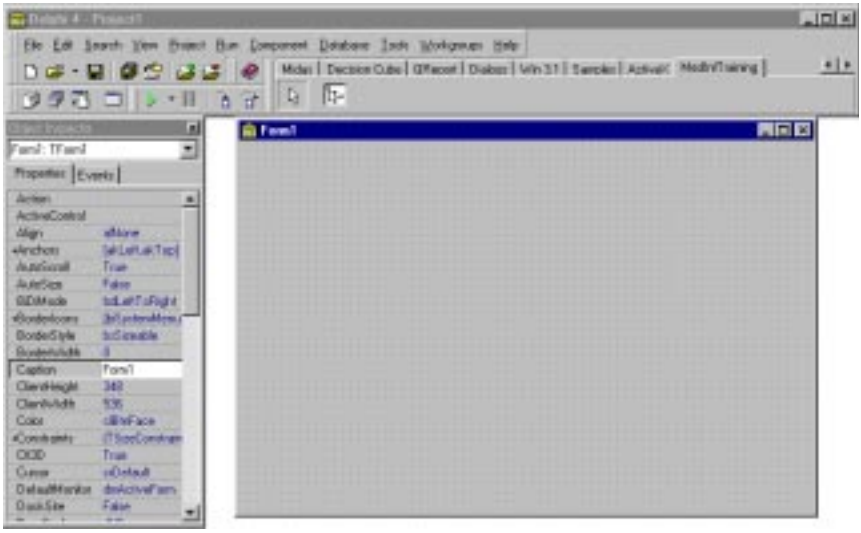
C:\Projects1\Прізвище25



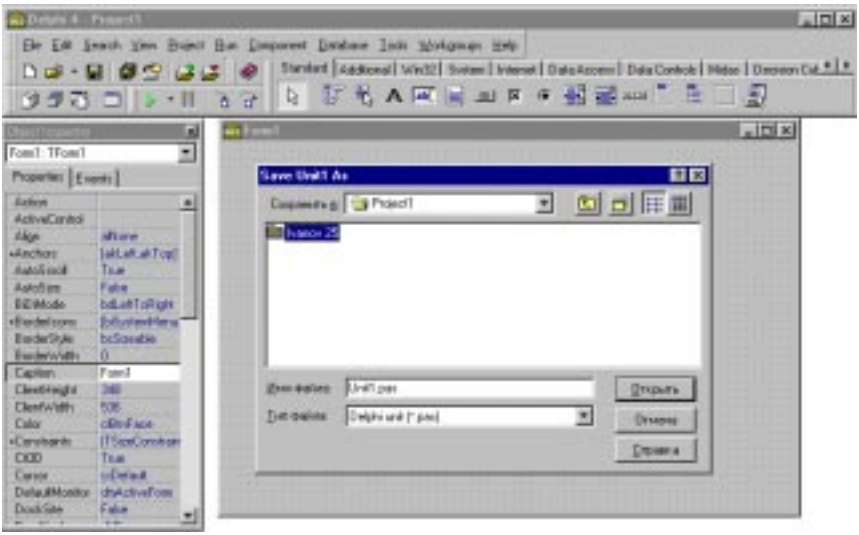
2. Скопіюйте каталог Video у створений у пункті 1 каталог. По замовчуванню каталог Video знаходиться на робочому столі.
3. Запустіть інструментальну систему Delphi.



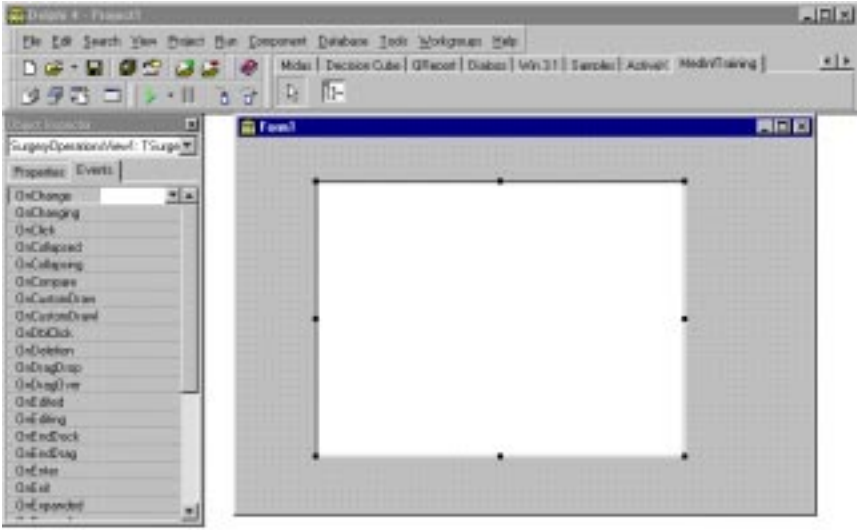
4. Створіть новий проект командою File — New Application



5. Збережіть проект у каталозі, створеному у пункті 1 командою File — Save Project as.

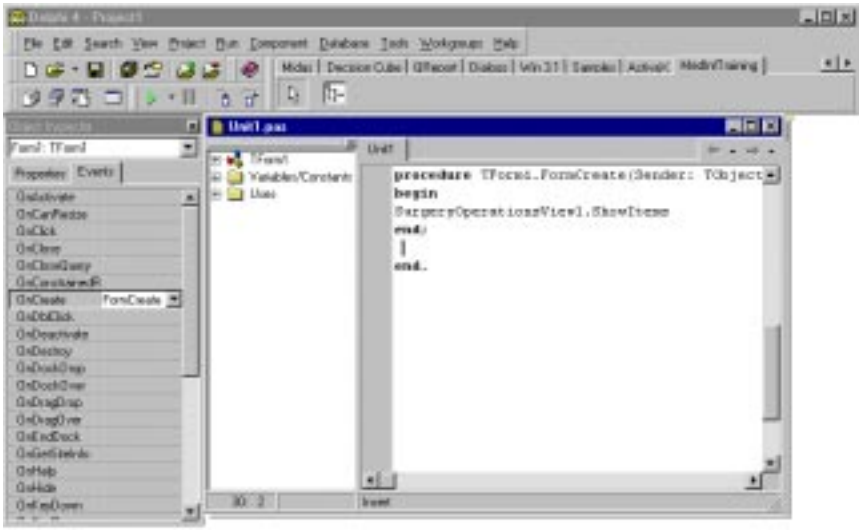


6. Знайдіть на сторінці MedInfTraining Палітри компонент компонент SurgeryOperationsViewer і розмістіть у вікні форми.

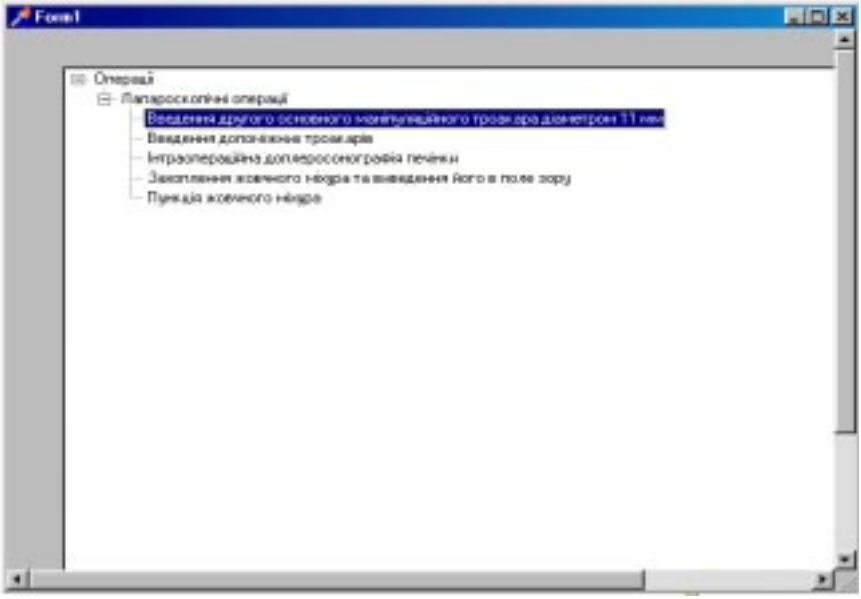


7. Перейдіть на сторінку Events в інспекторі об'єктів для форми. Знайдіть обробник події OnCreate. Двічі натисніть лівою клавішею мишки по полю правіше OnCreate. У вікно редактора команд, що з'явилося, введіть:

SurgeryOperationsView1.ShowItems;



8. Натиснувши клавішу F9, запустіть програму на виконання. За відсутності помилок, можна переглянути список відеофрагментів хірургічних лапароскопічних операцій.



Для того, щоб переглянути наявні відеозаписи, слід двічі клацнути лівою кнопкою мишки на назві відповідної операції (її фрагмента). При цьому відтворення запису буде відбуватися в окремому вікні за допомогою асоційованої з даним типом файла програми. У випадку, що показаний на рис. 4 — це "Универсальный Проигрыватель".

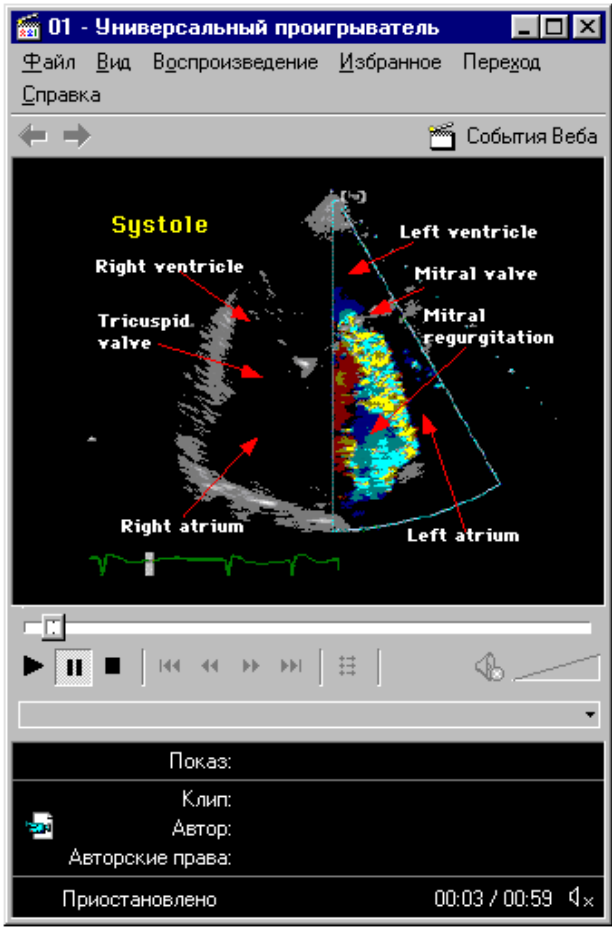


Рис. 4. Відтворення файла відеозапису хірургічної операції за допомогою асоційованої програми.

ВСТУП ДО АНАЛІЗУ МЕДИЧНИХ ЗОБРАЖЕНЬ. ПЕРЕГЛЯДАЧ МЕДИЧНИХ СЛАЙДІВ

Медична інформація має два основних способи представлення:

- у вигляді тексту;
- у вигляді зображення.

Робота з графічною інформацією традиційно є однією з найцікавіших галузей застосування комп'ютера в медицині. Таке застосування комп'ютера сьогодні вивчає спеціальний підрозділ медичної інформатики, що отримав назву аналіз медичних зображень. Це порівняно досить молодий напрямок, якому вже було приурочено роботу ряду конференцій (щорічні конгреси AMIA, див. www.amia.org), видаються журнали (див. www.elsevier.com/locate/media) та монографії.

Зображення з точки зору пам'яті комп'ютера можна трактувати просто як масив чисел, на зразок неструктурованого медичного запису (скажімо, про пацієнта) — це просто масив символів. Медичні зображення вирізняються тим, що вони несуть великий вміст інформації, даних (як і будь-яке три вимірне зображення).

При цьому без виділення певних типів структур (якими для медичних зображень є, наприклад, різні органи, ділянки органів) дані можуть бути відображені, але подальша їх обробка неможлива. Оцінюючи зображення, можна виділити ще більше абстрактної інформації, що є корисною для діагностики та терапії. Оцінювання зображення може здійснюватися як завдяки візуалізації, так і за допомогою кількісних аналітичних методів.

Аналіз медичних зображень розв'язує дві головні проблеми:

- реєстрація зображень;
- візуалізація зображень.

Проблема реєстрації зображень

Однією з найскладніших задач, яка ще чекає остаточного розв'язання в аналізі медичних зображень, є реєстрація зображень, які є, як правило, три вимірними. Реєстрація медичного зображення є винятково важливою для подальшого його аналізу. Прийняте наступне означення [1] реєстрації для видозмін A і B того ж об'єму.

Реєстрація для двох видозмін A і B — це оцінка відображення між системами координат Ref_A та Ref_B, пов'язаних з кожною видозміною:

$$\vec{x}_B = T(\vec{x}_A),$$

де $\vec{x}_A = (x_A, y_A, z_A)$, $\vec{x}_B = (x_B, y_B, z_B)$ — точки в системах координат Ref_A та Ref_B відповідно, які відповідають тій же анатомічній точці.

Реєстрація поверхні може бути розділена на три стадії, як показано на малюнку 1: вибір перетворення, представлення поверхні та критерій подібності, узгодження та глобальна оптимізація. Перший етап використовує припущення, зроблені стосовно природи взаємозв'язків між двома видозмінами. Другий етап визначає, який тип інформації ми отримуємо з три вимірних поверхонь, які характеризують їх локальну та глобальну поверхні і, як ми організуємо цю інформацію для представлення поверхні, що призведе до покращення ефективності на останньому етапі. Останній етап дає відповідь на запитання, як ми досліджуємо цю інформацію, щоб оцінити перетворення, яке максимізує міру подібності глобальної поверхні цих двох поверхонь.

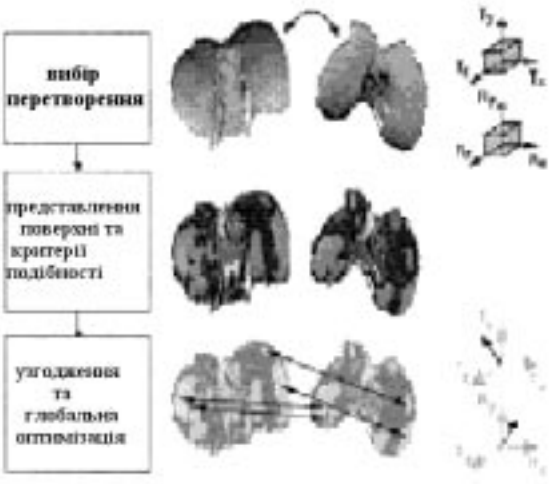


Рис. 1. Етапи реєстрації 3-вимірних поверхонь.

Проблема візуалізації зображень

На сьогодні зустрічаються наступні проекції зображень.

дво вимірна проекція зображень

При рентгенологічному чи флюорографічному дослідженні промені проходять через внутрішні структури тіла. На вході такої системи береться три вимірний об'єм. На виході отримуємо лише єдине дво вимірне зображення. Таке дво вимірне зображення має багато корисної інформації, та отримати її складно. Багато структур залишаються незрозумілими (наприклад, ребра, що затемнюють легені). Дійсні три вимірні структури не виявляються.

Дво вимірні томографічні зображення

В УЗД, комп'ютерній томографії (КТ) береться об'ємний перереіз. Тобто на вході системи ми маємо дво вимірний об'ємний перереіз. На виході маємо також дво вимірне зображення. Хоча помітно усі структури, все ж можна втратити цікаві частини об'єму в цілому. І знову ж три вимірна структура — невідома.

три вимірне об'ємне зображення

Використовуючи УЗД або КТ та ряд томографічних перерізів отримується об'єм. Отже, на вході такої системи маємо три вимірне зображення, на виході — три вимірний об'єм. При цьому об'єм розглядається повністю, а отже, ніщо не втрачається чи викликає сумнів. Однак тут маємо справу із набагато більшою кількістю даних. Можна навіть "сфотографувати" послідовність об'ємів в часі.

Порівняння дво вимірної та три вимірної візуалізацій

Проекція томографічної візуалізації проста; дво вимірне зображення відображається на дво вимірний дисплей (світлину або монітор). Об'ємна візуалізація складніша: три вимірний об'єм

повинен бути якимось чином перевпорядкований на дво вимірний пристрій (монітор комп'ютера).

Способи дво вимірної візуалізації

Режим фільму передбачає огляд осьових площин, як це робиться в анімації. Режим багато-площинного переформатування передбачає огляд осьових та довільних похилих площин.

Способи дійсної три вимірної візуалізації

При виборі способу три вимірної візуалізації повинні враховуватися такі обставини. Наші очі та мозок добре адаптовані до інтерпретації три вимірних, а не дво вимірних сцен. Методи візуалізації повинні візуалізувати елементи усього об'єму. Інтуїтивна візуалізація повинна відображати інформацію в природнішій формі. На сьогодні використовуються такі способи три вимірної візуалізації.

Проекція максимальної інтенсивності знаходить значення максимальної інтенсивності вздовж променя, що проходить через об'єм. Перевагою такого методу є те, що три вимірна структура може бути легко візуалізована при поворотах точки зору. Недоліками є:

- багато інформації втрачається (наприклад, коли всі значення — максимальні);
- деталі відносно рівних поверхонь втрачаються.

Відображення затіненої поверхні передбачає визначення затіненої поверхні на основі об'ємних даних і тоді її відображення. Перевагою є те, що він дає реальний три вимірний вигляд з хорошою візуалізацією морфології поверхні. Недоліками є:

- багато даних втрачається (наприклад, все поза поверхнею);
- метод вимагає визначення поверхні (це є складним завданням сегментації).

Об'ємне виконання (volume rendering). Таблиця непрозорості робить деякі інтенсивності прозорими (наприклад,

повітря), деякі — непрозорими (наприклад, тканина). Перевагами є реальний три вимірний вигляд без потреби сегментації та надзвичайна якість зображення. Недоліком може бути сповільненість. Адже більшість спеціалізованого графічного апаратного забезпечення сконструйовано і оптимізовано для відображення поверхонь, а не об'ємного виконання.

Застосування три вимірної візуалізації

Віртуальна колоноскопія. При цьому дані три вимірних зображень отримуються спіральним КТ; віртуальна камера "пілотується" вздовж кишечника; віртуальні ендоскопічні зображення візуалізуються. Перевагами над справжньою колоноскопією є:

- усунення ризику перфорації, комфортність для пацієнта;
- навігація, обмежене поле зору.

Віртуальна колоноскопія включає автоматичну навігацію, віртуальний розтин та картографічну проекцію. *Автоматична навігація* розроблена з метою "пілотування" віртуальною камерою, уникаючи зіткнень із стінками і стабілізуючи камеру. *Віртуальний розтин* (autopsy) спочатку математично випрямляє і розкручує кишечник, тоді візуалізує об'єм. Тобто, можна візуалізувати об'єм, як єдине статичне зображення. *Картографічна проекція* — це циліндрична проекція з рівними віддалями.

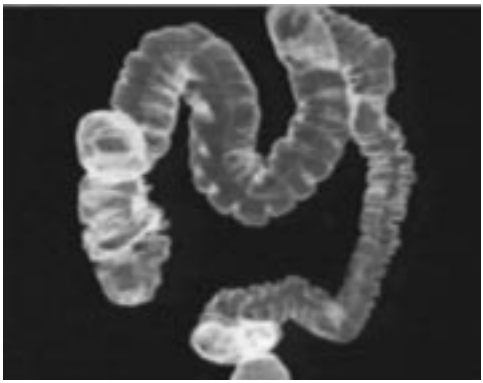


Рис. 2. Віртуальна колоноскопія.

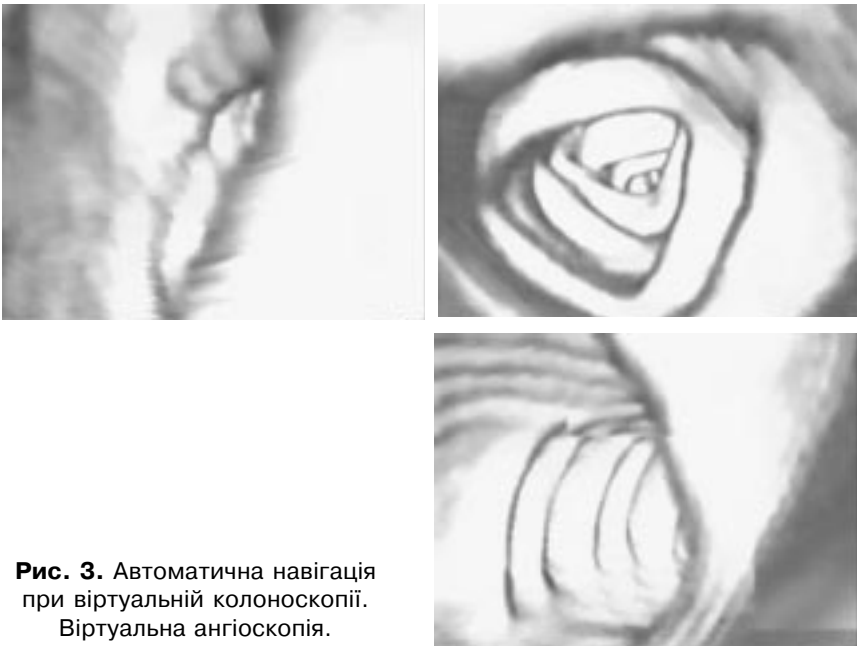


Рис. 3. Автоматична навігація при віртуальній колоноскопії. Віртуальна ангіоскопія.

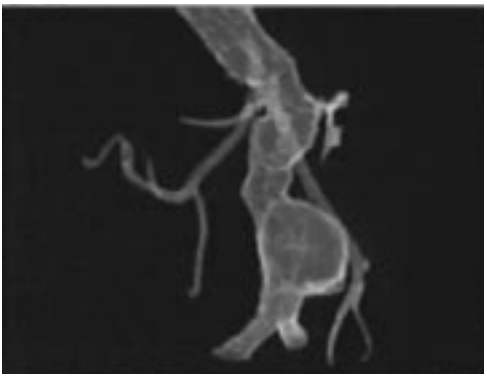


Рис. 4. Віртуальна ангіоскопія.

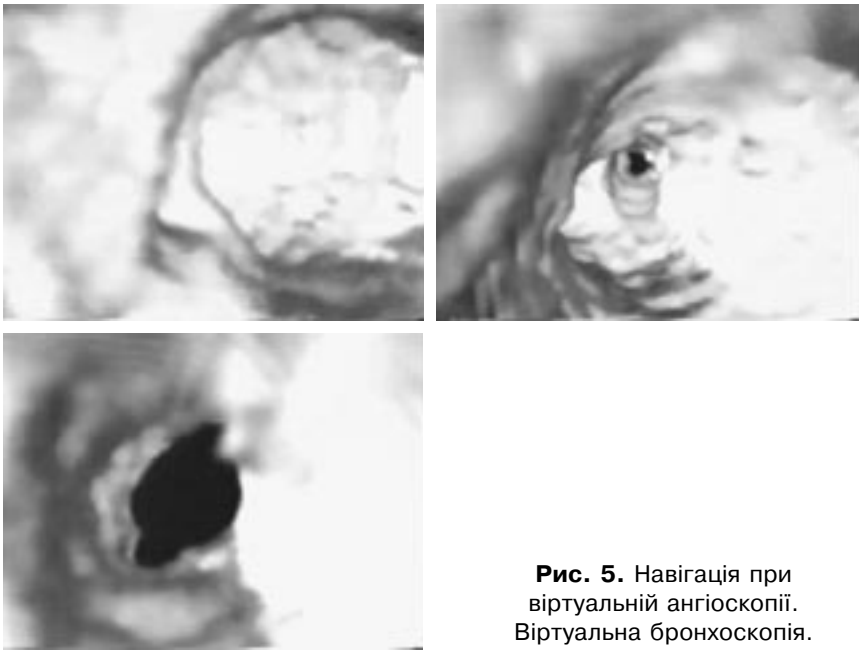


Рис. 5. Навігація при віртуальній ангіоскопії. Віртуальна бронхоскопія.

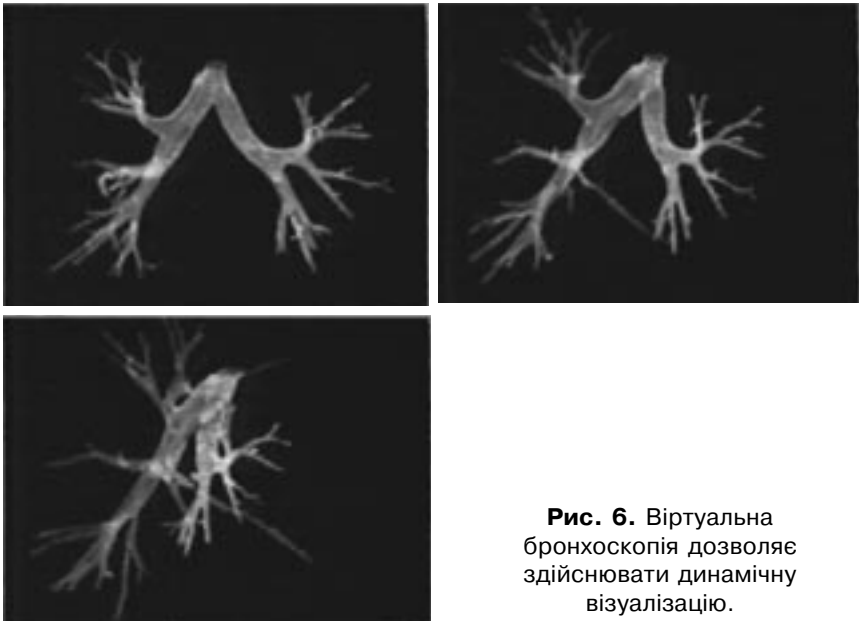


Рис. 6. Віртуальна бронхоскопія дозволяє здійснювати динамічну візуалізацію.

Структурна квонтифікація. Багато характеристик поверхонь (як функції від розмірів) є важливими для прийняття діагностичних та терапевтичних рішень. Сюди належать площа поперечного перерізу, середній діаметр, довжина, кривизна. Структурна квонтифікація застосовується в оцінці судинної, респіраторної та інших функцій.

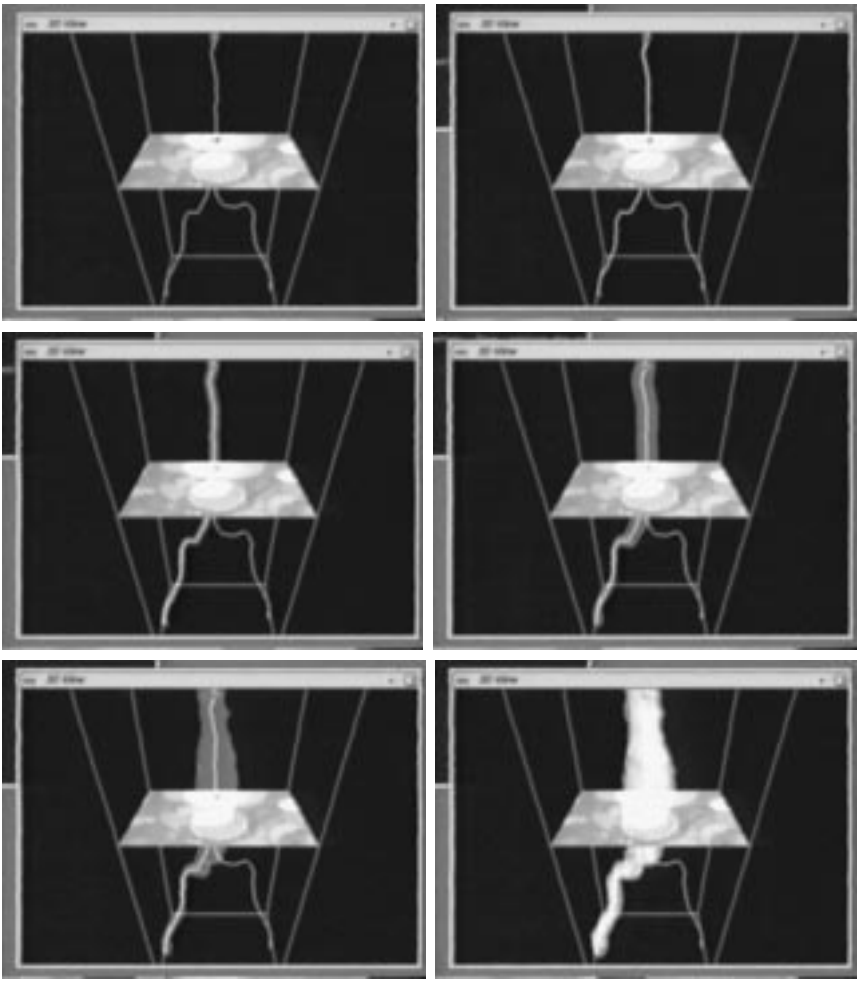


Рис. 7. Етапи структурної квантифікації. Сегментація тромбів (virtual angioplasty).

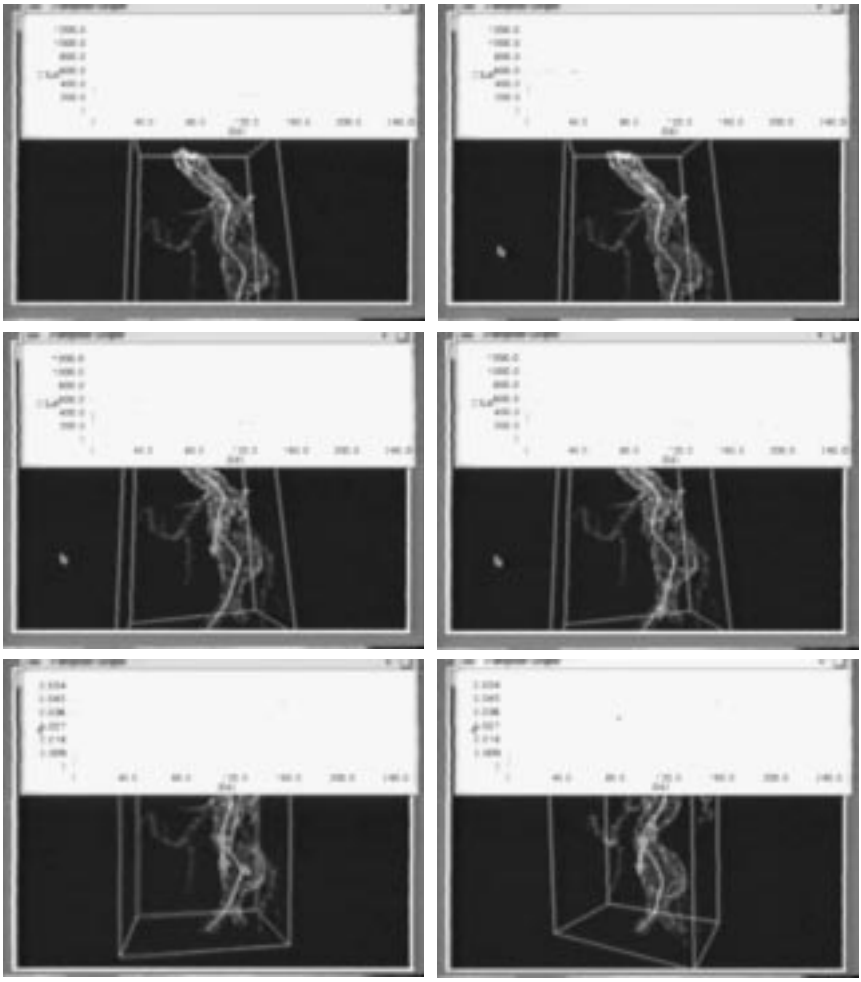


Рис. 8. Виконання сегментації тромбів.

Отже, оскільки з'являються нові медичні зображення, то є потреба в нових способах для їх аналізу.

Проект: переглядач медичних слайдів

Зазначимо, що проблеми аналізу медичних зображень досить близькі до загальних проблем розпізнавання образів. На

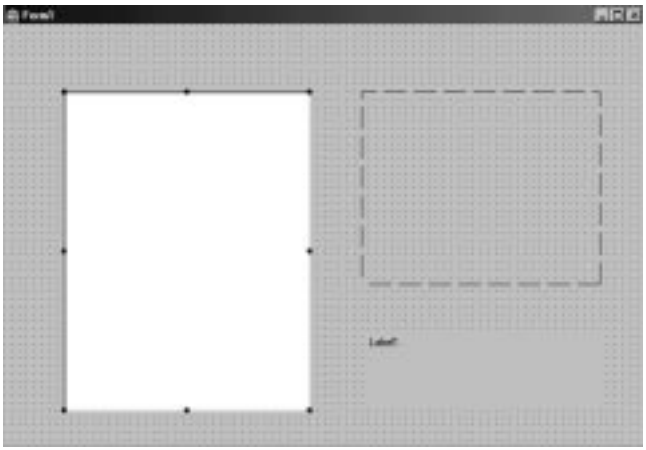
сьогодні ще немає загальновизнаного програмного забезпечення для розв'язання такого класу задач. Виконуються окремі дослідницькі проекти в провідних медичних університетах світу. Для розв'язування такого класу задач користуються сучасними інструментальними системами. Delphi має одні з найкращих можливостей для роботи з графічною інформацією. Наступний додаток — переглядач медичних слайдів цьому підтвердження.

Необхідне програмне забезпечення:

інструментальна система Delphi, в якій інстальовано компонент SlidesViewer, каталог Slides із файлами слайдів, опис яких міститься у файлі Slides.ini.

Порядок розробки проекту "Переглядач медичних слайдів":

1. Створіть каталог для збереження проекту.
2. Розмістіть каталог Slides із зображеннями у каталозі проекту, створеному на першому кроці.
3. Запустіть Delphi і створіть новий проект командою File — New Application
4. Виберіть на сторінці MedInfTraining Палітри Компонент компонент MedicalSlidesViewer  і розмістіть його у вікні форми.
5. Знайдіть на сторінці Standard Палітри Компонент компонент Label, на сторінці Additional компонент Image і розмістіть їх у вікні форми. За припущенням вони отримають імена Label1 і Image1 відповідно.



6. Встановіть властивості у наступні значення для компонента MedicalSlidesViewer1

Display	Image1
Explanation	Label1

для компонента Image1

Stretch	True

для Form1

Position	poScreenCenter

7. На сторінці Events компонента Form1 Інспектора Об'єктів знайдіть обробник події OnCreate.

У вікні редактора коду надрукуйте команду MedicalSlidesViewer1.ShowItems;

8. Збережіть проект у каталозі, створеному на кроці 1.
9. Скомпілюйте і запустіть проект. Змініть розміри компонент MedicalSlidesViewer1, Image1, Label1 з метою отримання найкращого результату при відображенні зображень та їх описів.

Компонент TMedicalSlidesViewer можна використовувати для створення інших додатків-броузерів медичних зображень, їх документування. Декларація компонента має такий вигляд (повний лістинг представлений у додатку):

```
TMedicalSlidesViewer = class(TTreeView)
  private
    { Private declarations }
    ini: TIniFile;
    strs, first_substrs, second_substrs, third_substrs: TStrings;
    FDisplay: TImage;
    FExplanation: TLabel;
    procedure SetDisplay (Value: TImage);
    procedure SetExplanation (Value: TLabel);
  protected
    { Protected declarations }
  public
    { Public declarations }
    procedure DblClick; override;
    procedure ShowItems;
    constructor Create(AOwner: TComponent); override;
  published
    { Published declarations }
```

```
    property Display: TImage read FDisplay write SetDisplay;
    property Explanation: TLabel read FExplanation write
    SetExplanation;
  end;
```

Тут ini — це компонент-файл для зберігання файла опису зображень (з іменем slides.ini, зберігається в каталозі додатка); Display — компонент-дисплей, на якому безпосередньо і розміщується зображення; Explanation — компонент, в якому знаходиться опис поточного зображення; метод ShowItems здійснює початкове "під'єднання" файлів зображень до компонента.

Файл **slides.ini** повинен мати спеціальний формат, показаний нижче

```
[Title of images group]
Title_of_image                                          =
Title_of_image_that_appears_in_MedicalSlidesViewer  |
Description_of_image_that_appears_in_Explanation    |
real_file_name_of_image
```

Тут символи [,], = і | це сепаратори.

Запитання та вправи

1. Вкажіть головні задачі аналізу медичних зображень.
2. Види проекцій медичних зображень.
3. Вкажіть проблеми графічного представлення 3-вимірних графічних зображень.
4. Переваги віртуальної колоноскопії.
5. Розробіть додаток з можливістю одночасного перегляду кількох груп медичних зображень в одному вікні. Скористайтеся компонентом TMedicalSlidesViewer.

ПРОВЕДЕННЯ НАУКОВИХ ОБЧИСЛЕНЬ У МЕДИЦИНІ

Однією з важливих галузей застосування інструментальних систем у медицині залишається медична наука, особливо такий її розділ, як математичне моделювання медичних процесів. Більшість з них має складний нелінійний характер.

Для представлення закономірностей розвитку медичних процесів слід використовувати системи з післядією. Динамічні системи з післядією відіграють важливу роль в економіці, біології, медицині, техніці. Чисельні застосування ередитарних систем показані в роботах [1-3]. У той же час широке практичне використання вказаного класу систем пов'язане з певними труднощами, що виникають при чисельному інтегруванні (розв'язуванні) відповідних диференціальних рівнянь. Однією з причин є, мабуть, нескінченновимірність задач. Нескінченновимірність задачі означає, що не існує скінченного числа деяких базових розв'язків, через які буде виражатися будь-який інший розв'язок системи. У цьому, можливо, і причина недостатнього розвитку аналітичної теорії (теорії явних розв'язків) вказаного класу систем. Тому слід вдатися до чисельно-комп'ютерного розв'язування.

Постановка задачі

Розглядається клас динамічних систем, що описуються диференціальними рівняннями із запізненням:

$$y'(x) = F(x, y(x), y(x-\tau_1(x)), y(x-\tau_2(x),..., y(x-\tau_m(x)), \; x>0. \quad (1)$$

Тут $x \in R$ — незалежна змінна, $y(x) \in R^n$ — фазова траєкторія системи, $\tau_1(x), \tau_2(x),..., \tau_m(x)$ — змінні запізнення в системі, $F(\bullet) \in R^n$ — функціонал, який може бути навіть кусково-неперервним.

Припускається, що функції $\tau_i(x), \;\; i = \overline{1,m}$ приймають на додатній півосі невід'ємні значення і обмежені знизу і зверху. По-

ведінка системи на початковому проміжку часу вважається заданою:

$$y(x) = \phi(x), \qquad x \in \left[\max_{x>0, i=\overline{1,m}} \tau_i(x), 0 \right]. \qquad (2)$$

Тут $\phi(x)$ — функція, що допускає розв'язок початкової задачі (1), (2).

Розглянемо метод, придатний для комп'ютерного розв'язування початкових задач загального вигляду (1), (2).

Методи і алгоритми розв'язування

При чисельному інтегруванні звичайних диференціальних рівнянь найбільший розвиток отримав метод Рунге-Кутти і численні його модифікації. Стосовно задач (1), (2) метод Рунге-Кутти можна застосувати лише у випадку одного сталого запізнення t.

Причому це може бути лише метод з постійною довжиною кроку, що призводить до значних похибок і відзначається поганою стійкістю. При розгляді більш складних систем і використанні більш досконалих методів із змінною довжиною кроку виникають наступні труднощі:

- при розгляді декількох змінних запізнень не завжди відомо наскільки великий діапазон попередніх станів системи потрібно зберігати для обчислення подальших;
- не завжди зберігаються саме ті значення, які будуть вимагатися на майбутніх кроках;
- невідомо, яка повинна бути щільність значень, що зберігаються.

Всі вищеперелічені труднощі дозволяє уникнути глобальна апроксимація розв'язку. Відповідними методами для щільної видачі розв'язків є методи типу Адамса, неперервні методи Рунге-Кутти. У зв'язку з легкістю програмування і переваги над іншими інтерполяційними методами [3] в даній роботі вибраний метод Дормана і Прінса порядку 5(4).

Програмна побудова розв'язку задачі (1), (2) будується на використанні ООП в середовищі Java.

На малюнку 1 наведена схема класів, включених в проект (Workspace) для розв'язування задач (1), (2). Тут Delay SystemSolution — клас, в якому описується початкова поведінка і праві частини системи (1), а також здійснюється чисельне інтегрування методом Дормана-Принса; GraphConstruction — клас-аплет, що здійснює візуальне відображення графіків розв'язків, їх ретельніше відображення і вивчення; FunctionList — клас — "пов'язаний список", призначений для розв'язування задачі одночасного виведення кількох компонент розв'язку в одній площині; ColorList — клас, призначений для "різнокольорового" відображення графіків різних компонент розв'язку; LinkedList — клас — "пов'язаний список", призначений для відображення розв'язку задачі (1), (2) в різних межах зміни; BoundsLocation — клас для збереження меж зміни.

Стрілками позначені напрямки взаємодії класів. Клас GraphConstruction є типовою бібліотекою машинної графіки, що має багаті візуальні можливості щодо вивчення поведінки системи на найдрібніших інтервалах зміни, одночасного виведення кількох графіків в одній площині.

Клас використовує багатопоточність, а з метою уникнення мерехтіння використовується метод подвійної буферизації. Нижче вказані головні змінні і методи класу GraphConstruction.

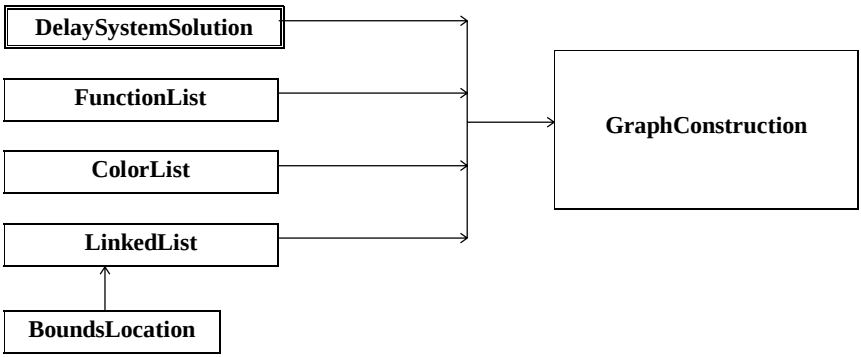


Рис. 1. Схема класів проекту розв'язування задач (1), (2).

```
public class GraphConstruction extends java.applet.Applet
implements Runnable
{
    DelaySystemSolution SampleSystemSolution;
    Thread m_Graphics = null; // потік класу
    private double m_x0;// ліва межа x
    private double m_x1;// права межа x
    // другий буфер зображення
    private Image m_image; // позаекранне зображення
    private Graphics m_g; // графічний об'єкт позаекранного
зображення
    double f(int i, double x) // метод класу, що вказує на функцію, графік якої // потрібно відобразити
    {
            return SampleSystemSolution.ylag(i, x);
     }
```

Об'єкт класу DelaySystemSolution, асоційований із системою із запізненням, що досліджується, створюється в класі GraphConstruction і є його змінною.

Далі зупинимося на будові класу DelaySystemSolution. Нижче вказані його головні змінні.

```
public class DelaySystemSolution
{
    double[] y; // змінна-масив для зберігання початкового
значення і шуканого
    // розв'язку в точці xend
    double x0; // початкова точка для x
    private double xstor[]; // масив точок, в яких проводилося
інтегрування
    private double ystor[][]; // масив знайдених розв'язків
    private double c1[][]; // масиви коефіцієнтів апроксимуючих багаточленів,
    private double c2[][]; // в яких зберігаються коефіцієнти
при доданках
    private double c3[][]; // відповідно першої, другої, третьої,
четвертої
    private double c4[][]; // степені
```

int n; // розмірність системи; може бути <= 51

Метод fcn, оголошений як

public double[] fcn(double x, double y[])

повертає масив значень правих частин системи (1). Нижче буде показана його реалізація на прикладі розв'язування задачі медичної інформатики.

Метод phi, оголошений як

public double phi(int i, double x)

повертає значення початкової функції $\phi(x)$, обчислене для 1-ї компоненти розв'язку.

Знаходження наближеного розв'язку задачі (1), (2) здійснюється в класі DelaySystemSolution таким чином. У конструктор класу DelaySystemSolution включений виклик методу retard, оголошеного як:

public void retard(int n, double x, double y[], double xend, double eps, double hmax, double h).

Далі, після успішного кроку інтегрування метод retard викликає метод store, який записує у змінні xstor, c1, c2, c3, c4 коефіцієнти многочленів з відповідними значеннями .

Після цього при виклику методу ylag(i, x) здійснюється пошук потрібного інтервалу для x і обчислюється відповідний многочлен для 1-го розв'язку. Отже, за допомогою методу ylag можна отримати 1-й розв'язок в довільній точці x. Ця властивість використовується для задання запізнення в методі fcn.

Результати

Як приклад, розглянемо задачу про реакцію клітки на рентгенівське опромінення. Припустимо, що опромінення триває протягом часу T (і починається при $t = 0$), а потім припиняється.

Припускаємо, що клітини мають здатність поповнювати нестачу або усувати надлишок певної речовини, але їх реакція відбувається із запізненням, що дорівнює t. Нехай $x(t)$ — концентрація речовини в клітині, що піддається опроміненню, a — стала опромінення, що залежить від степені опромінення, b — стала, що показує реакцію клітини на відхилення від рівноважної концентрації x_0. Очевидно, що $x(t) \equiv x_0, \quad t \le 0$. Виходячи

із зроблених припущень, концентрація описується рівняннями:

$$\frac{dx(t)}{dt} = ax(t) + b(x(t-\tau) - x_0), \quad 0 \le t < T,$$
$$\frac{dx(t)}{dt} = b(x(t-\tau) - x_0), \quad t \ge T \qquad (3)$$

Розв'язки рівнянь можуть бути знайдені методом, запропонованим в даній роботі. Для цього потрібно створити такий метод fcn:

```
public double[] fcn(double x, double y[])
{
        //this method describes "right side" of system
        double T = 20.;
        double tau = 1.;
        double[] dRight_side = new double[Math.max(n, nn)
+ 1];
        if((x>0) & (x<T))
        {
                dRight_side[1] = m_a * y[1] + m_b * (ylag(1,
x — tau) — phi(1,0));
        }
        else
        {
                dRight_side[1] = m_b * (ylag(1, x — tau) —
phi(1,0));
        }
        return dRight_side;
}
```

Припускаємо, що m_a і m_b є змінними класу DelaySystemSolution, призначеними для зберігання значень *a, b*. На рисунку 2 представлено розв'язок вказаної задачі для випадку $a = -1,\ b = -1,\ T = 20,\ \tau = 1,\ x_0 = 1$.

В якості наступного прикладу розглядається модель запального процесу інфекційної природи. У загальному вигляді математична модель імунітету описана в роботі [2]. Вона універсаль-

на і справедлива не тільки для запального процесу, але і для інфекційного зараження організму. У моделі враховуються наступні визначальні для перебігу процесу чинники:

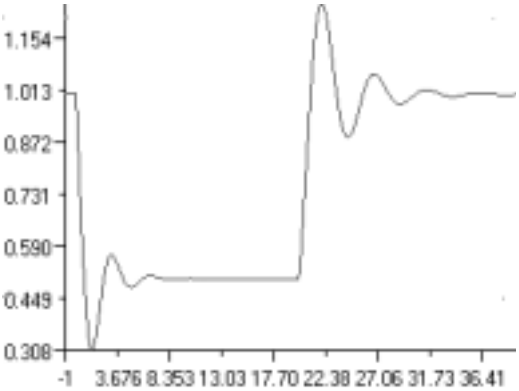


Рис. 2. Зміна концентрації речовини в результаті рентгенівського опромінення.

I. Популяція антигенів V, що розмножуються в організмі;

II. Популяція антитілоутворювальних клітин (плазмоклітин) C;

III. Кількість антитіл (імуноглобулінів) F в організмі;

IV. Ступінь пошкодження органа m.

Рівняння, що визначають динаміку перерахованих чинників, мають вигляд:

$$\frac{dV}{dt} = (\beta - \gamma F)V \ ,$$

$$\frac{dC}{dt} = \xi(m)\alpha\ V(t-\tau)F(t-\tau) - \mu_c(C - C_0) \ ,$$

$$\frac{dF}{dt} = \rho\, C - (\mu_f + \eta\,\gamma\, V)F \ ,$$

$$\frac{dm}{dt} = \sigma\, V - \mu_m m$$

з початковими умовами при $t \in [-\tau, 0]$;

$$V(t) = V_0, F(t) = F_0, C(t) = C_0, m(t) = 0 \ .$$

Тут β — коефіцієнт розмноження антигену; γ — коефіцієнт, що визначає ймовірність нейтралізації антигену антитілом; α — коефіцієнт, що зумовлює ймовірність зустрічі антиген-антити-

ло; μ_c — коефіцієнт, обернений до часу життя плазмоклітин; ρ — швидкість виробництва антитіл однією плазмоклітиною; μ_f — коефіцієнт, обернено пропорційний до часу розпаду антитіл; η — число антитіл, що вимагаються на нейтралізацію одного антигену; σ — коефіцієнт, що визначає швидкість загибелі клітин за рахунок пошкоджуючої дії антигену; μ_m — коефіцієнт, що враховує швидкість відновлення пошкодженого органа; τ — фаза запізнення (час, за який здійснюється формування каскаду плазмоклітин); $\xi(m)$ — неперервна незростаюча функція $(0 \le \xi(m) \le 1)$, що характеризує порушення нормального функціонування імунної системи через значне пошкодження органа-мішені.

Перераховані параметри додатні і є специфічними як для виду антигену, так і для органа і конкретного організму. За допомогою розробленої комп'ютерної моделі здійснене якісне дослідження запального процесу у випадку, коли:

$$\beta = 2, \gamma = 0.8, \alpha = 10^4, \mu_c = 0.5, \rho = 0.17, \mu_f = 0.17, \eta = 10, \mu_m = 0.12, \tau = 0.5$$

$$\xi(m) = \begin{cases} 1, m \le 0.1 \\ (1-m)/(10/9), 0.1 \le m \le 1 \end{cases}.$$

При $t \in [-\tau, 0]$ справедливі такі початкові умови:

$$V(t) = \max(0, x + 10^{-6}), C(t) = 1, F(t) = 1, m(t) = 0.$$

Здійснене моделювання показує, що час повторної появи запального процесу і ступінь його активності залежать від коефіцієнта σ, що узгоджується з експериментальними даними (рис. 3, 4).

Порядок розробки Internet-проекту для моделювання системи імунного захисту

1. Створіть новий каталог для збереження усіх файлів проекту.
2. Розмістіть у каталозі файли Java-класів: BoundsLocation, ColorList, DelaySystemSolution, GraphConstruction, LinkedList, ListOfSolutions, xyPoint.

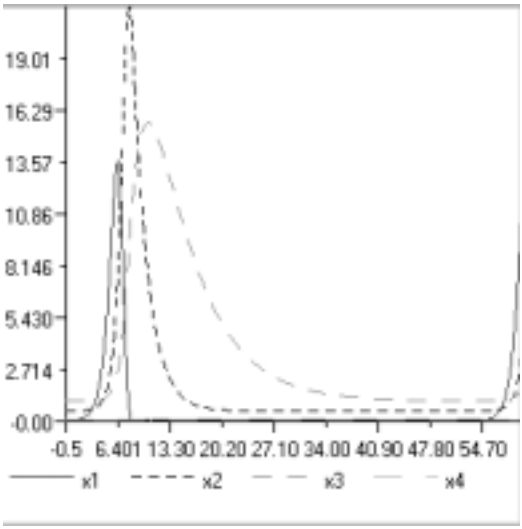


Рис. 3. Модель запального процесу інфекційної природи при s=2. Тут x1 , x2 , x3 , x4=10m.

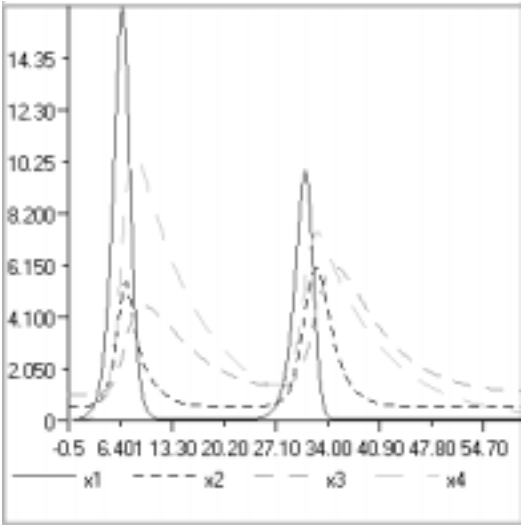
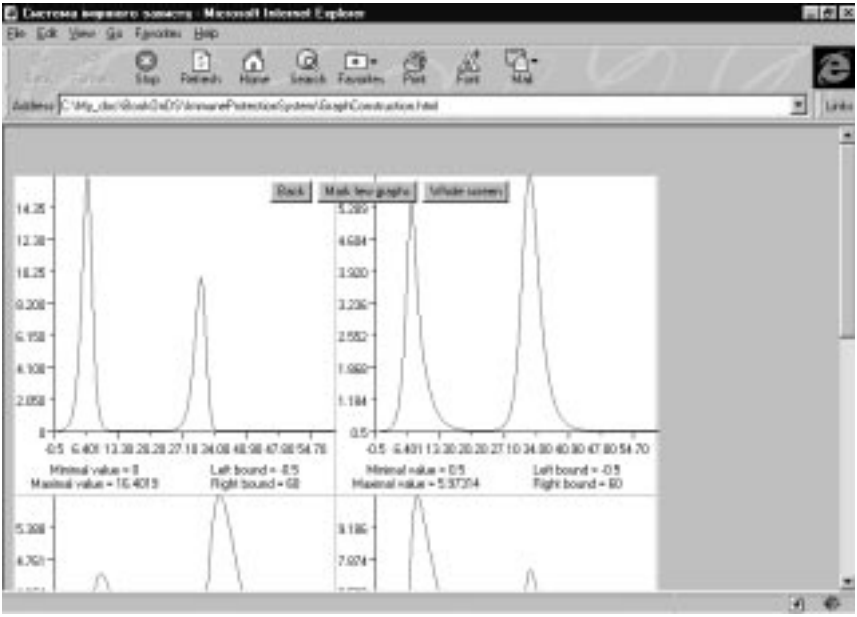


Рис. 4. Модель запального процесу інфекційної природи при s=300. Тут x1 , x2 , x3 , x4=10m.

3. Створіть текстовий файл вигляду:

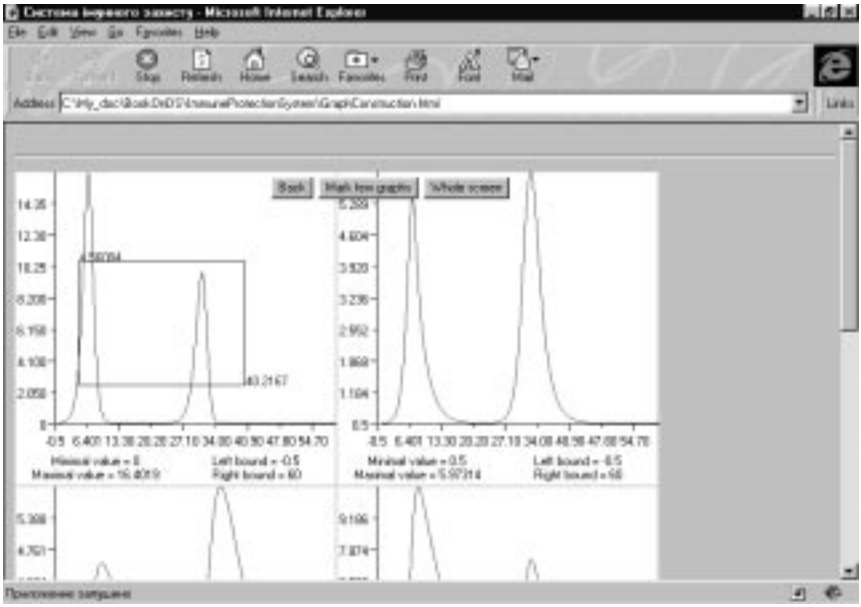
```
<html>
<head>
<title>Система імунного захисту</title>
</head>
<body>
<hr>
<applet
   code=GraphConstruction.class
   id=GraphConstruction
   width=700
   height=900 >
<param name=x0 value=0>
<param name=x1 value=60>
<param name=Scale value=30>
<param name=Legend value=true>
<param name=GraphWidth value=300>
<param name=GraphHeight value=300>
<param name=GraphCount value=4>
<param name=PunctureLine value=true>
<param name=Delay value=.5>
<param name=Hmax value=.2>
<param name=Beta value=2.>
<param name=Gamma value=.8>
<param name=Alpha value=10000>
<param name=Mu_c value=.5>
<param name=Ro value=.17>
<param name=Mu_f value=.17>
<param name=Eta value=10.>
<param name=Sigma value=300.>
<param name=Mu_m value=.12>
<param name=Tau value=.5>
</applet>
</body>
</html>
```

4. Збережіть файл, надавши йому розширення .html.
5. Запустіть файл через програму Web-броузер, встановлену в системі. Результат виконання аплету показано на рисунку.

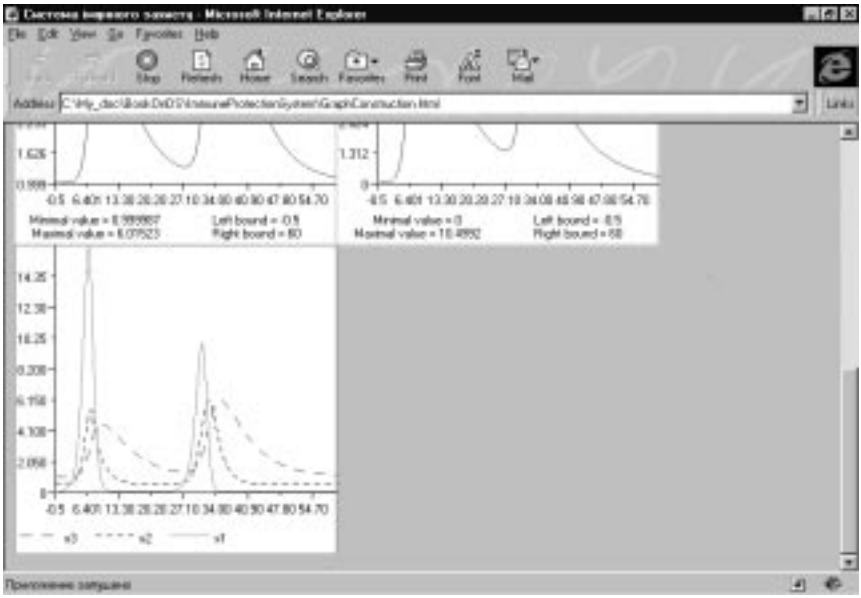


Запитання та завдання

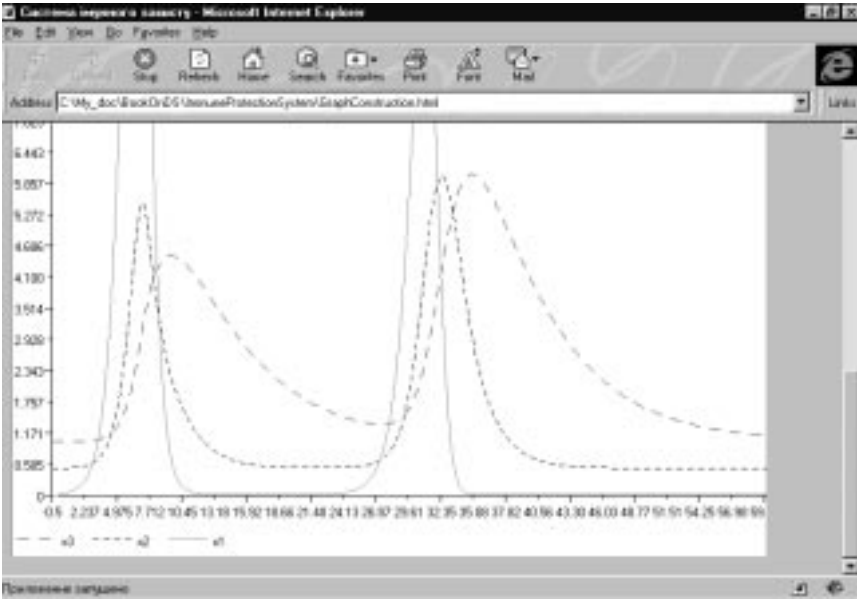
1. Дослідіть зміну різних характеристик імунної системи у різних часових межах:



2. Здійсніть одночасне виведення кількох графіків у одній площині за допомогою кнопки Mark few graphs:



3. Здійсніть повноекранний показ графіка за допомогою кнопки Whole screen:



4. Змоделюйте графіки для імунної системи для наступних значень параметрів моделі:

№ варіанта	β	γ	α	μ_c	ρ	μ_f	η	σ	μ_m	τ
1	1.	.5	8000	.3	.1	.1	8.	200.	.08	.4
2	2.	.3	5000	.2	.05	.1	5.	200.	.1	.8

Примітка. Для цього вносяться відповідні зміни у речення типу

<param name=Beta value=2.>

файла. html.

5. Створіть Web-сторінку, яка б одночасно моделювала імунні системи для двох груп параметрів, поданих у завданні 4.

ТЕОРЕТИЧНІ ПРИНЦИПИ ПРЕДСТАВЛЕННЯ МЕДИЧНИХ ЗНАНЬ

Загальні положення

Довільна діагностична система здійснює процес розпізнавання захворювань на основі конкретних алгоритмів, з яких найпоширенішими є такі варіанти розв'язування діагностичних задач.

Ймовірнісні алгоритми

Побудовані на припущенні, що поява у хворого тих чи інших симптомів є статистично незалежними подіями. Кожен із симптомів має свою умовну ймовірність появи при кожному із захворювань. На основі обчислення ймовірностей визначається найімовірніше при даному наборі симптомів захворювання.

Послідовний статистичний аналіз

Послідовний статистичний аналіз симптомів за частотою їх появи при захворюваннях, які потрібно діагностувати. Відношення ймовірностей двох захворювань, що називається відношенням правдоподібності, для одного симптому, що вивчається, повинно бути або більше певної величини (верхня межа), щоб можна було прийняти рішення на користь одного з діагнозів, або менше певної величини (нижня межа), щоб можна було прийняти рішення на користь іншого діагнозу. У випадку, якщо величина відношення правдоподібності займає проміжкове значення, проводиться додаткове вивчення наступного симптому і т.д. При цьому відношення ймовірностей сумуються і залежно від одержаних результатів приймається рішення на користь одного з діагнозів.

Методи розпізнавання образів

Подальшим розвитком цих методів є геометрична інтерпретація задачі автоматичного розпізнавання. Результат обстеження хворого за наявності n ознак може бути представлений у

вигляді точки в n-вимірному просторі. Діагностична задача зводиться до вибору такої лінії або площини, яка досить надійно би відділила простір ознак, характерних для одного захворювання, від простору ознак іншого захворювання.

Метод клінічного прецеденту

Грунтується на зіставленні всього набору ознак даного хворого із відповідними наборами ознак, характерними для різних захворювань. При цьому може бути використана геометрична інтерпретація, коли одна множина ознак зіставляється з іншою в n-вимірному просторі.

Метод фазового інтервалу

Полягає в пошуку мінімальних відстаней між точками на площині, що відносяться до досліджуваного об'єкта (симптому), і точками, що відносяться до різних множин, що представляють різні класи захворювань.

Крім перелічених методів, користуються також методами інформаційної оцінки, методами математичної логіки, логіко-дискретними методами і ін.

Розглянемо теоретичні положення експертних систем медичної діагностики (системи підтримки рішень) на прикладі онкології.

Вивчення тесту для раку з прихованим перебігом (РПП)

Система підтримки рішень (СПР) при аналізі даних використовує наступні показники:

- чутливість: якщо ми маємо 100 пацієнтів з РПП, то 95 з них мають "х" в крові;
- специфічність: якщо ми маємо 100 пацієнтів без РПП, то 95 з них не мають "х" в крові;
- розповсюдженість: з кожних 1000 випадково вибраних індивідумів 5 матимуть РПП.

При такому способі вивченні проблеми дані зручно зберігати у табличному вигляді:

	“x” наявні	**“x” відсутні**	
РПП наявний	475	25	500
РПП відсутній	4975	94525	99500
			100000

Тут числами вказано кількість відповідних індивідуумів.

Стандартна термінологія

Розглянуті вище числа отримали назви відповідно до таблиці:

	тест позитивний	**тест негативний**	
захворювання наявне	вірні позитиви (TP)	невірні негативи (FN)	TP + FN
захворювання відсутнє	невірні позитиви (FP)	вірні негативи (TN)	FP + TN
	TP + FP	FN + TN	повна популяція

Наведемо наступні формальні означення.

Означення

$$\text{Чутливість} = \frac{TP}{TP + FN} = P(T^+ \mid D^+).$$

$$\text{Специфічність} = \frac{TN}{FP + TN} = P(T^- \mid D^-).$$

$$PV^+ = \textit{передбачуване значення} = \frac{TP}{TP + FP} = P(D^+ \mid T^+).$$

Формула для позитивного передбачуваного значення

$$PV^{+} = \frac{(Чутливість)(Розповсюдженість)}{(Чутливість)(Розповсюдженість)+(1-Специфічність)(1-Розповсюдженість)}$$

Виявлення раку простати за допомогою визначення в сироватці простатичної кислої фосфатази радіоімунним методом

Повідомлення про даний метод з'явилося наприкінці 1970-х років [1, 2]. Він дозволив на основі аналізу крові уникати запізнень в діагностиці раку простати. Проілюструємо побудову СПР на основі даного методу.

Чутливість

	Кількість пацієнтів	**Кількість позитивних тестів**	**Чутливість**
Пацієнти з раком простати	113	79	70%
Стадія 1	24	8	33%
Стадія 2	33	26	79%
Стадія 3	31	22	71%
Стадія 4	25	23	92%

Специфічність

	Кількість пацієнтів	**Кількість позитивних тестів**	**Специфічність**
Пацієнти без раку простати	217	13	94 %
Normal controls	50	0	
BPH	36	2	
Після повної простатектомії	28	1	
Інші види раку	83	9	
Misc. GI diorders	20	1	

Використання в якості просіюючого тесту

Якщо даний метод використаємо без ректального дослідження, то маємо наступні значення:

- чутливість = 70 %;
- специфічність = 94 %;
- розповсюдженість = 30/100000;
- PV+ = 0.41 % (тобто, 1 серед 244 суб'єктів).

Якщо метод використаємо з ректальним дослідженням, то маємо:

- чутливість = 33 %;
- специфічність = 94 %;
- розповсюдженість = 33/100000;
- PV+ = 0.19 % (тобто, 1 серед 526 суб'єктів).

Коли тест корисний для просіювання?

Припустимо, що пацієнт має nodule при ректальному дослідженні. Встановлені наступні значення:

- чутливість = 79 %;
- специфічність = 94 %;
- розповсюдженість = 50 %.

Тоді можна обчислити, що:

PV+ = 93 % (ймовірність раку, якщо тест на кислу фосфатазу є позитивним);

PV- = 82 % (ймовірність, що немає раку, якщо тест на кислу фосфатазу є негативним).

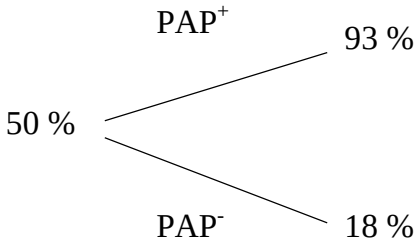


Рис. 1. Передбачувані значення в пацієнтів з nodule. Тут PAP+ — фосфатазу; PAP- — у випадку негативного тесту.

Комбінування тестів для просіювання

Якщо вже виконана біопсія простати, то її можна розглядати як інший тест. В такому випадку для біопсії маємо наступні значення:

- чутливість = 100 %;
- специфічність залежить від досвіду хірурга, що робить процедуру;
- розповсюдженість становить 50 %, якщо тест на кислу фосфатазу не вдалося здійснити, але 93 %, якщо тест на кислу фосфатазу є позитивним і 18 %, якщо тест на кислу фосфатазу є негативним.

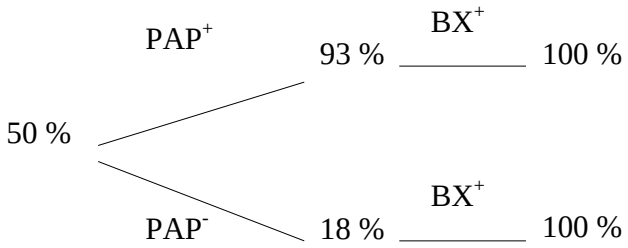


Рис. 2. Послідовне тестування.

Ймовірність раку після негативної біопсії

Чутливість біопсії можна представити наступною таблицею

	Чутливість біопсії		
	50 %	70 %	90 %
Тест на кислу фосфатазу позитивний (93 % перед біопсією)	87 %	80 %	56 %
Тест на кислу фосфатазу негативний (18 % перед біопсією)	10 %	6 %	2 %

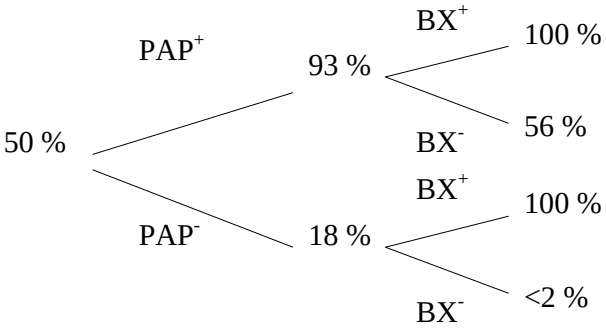


Рис. 3. Послідовне тестування.

Теорема Байєса

Теорема Байєса є одним із фундаментальних фактів теорії ймовірності. Її твердження стосовно нашої проблематики можна представити такими формулами:

$$PV^{+} = \frac{(Чутливість)(Розповсюдженість)}{(Чутливість)(Розповсюдженість)+(1-Специфічність)(1-Розповсюдженість)}$$

або

$$P(D^{+} \mid T^{+}) = \frac{P(T^{+} \mid D^{+})P(D^{+})}{P(T^{+} \mid D^{+})P(D^{+})+(1-P(T^{-} \mid D^{-})(1-P(D^{+}))}.$$

Типові припущення при використанні теореми Байєса

Повнота. Наприклад, всі чоловіки або мають, або не мають рак простати; немає інших можливостей.

Одночасна виключність. Наприклад, якщо чоловік має рак простати, він не може одночасно не мати раку простати.

Умовна незалежність. Наприклад, кисла фосфатаза та результат біопсії є умовно незалежними тестами. Тоді як ректальне дослідження та кисла фосфатаза можуть і не бути умовно незалежними.

ЕКСПЕРТНА СИСТЕМА МЕДИКО-ДІАГНОСТИЧНОГО ПРИЗНАЧЕННЯ НА ОСНОВІ БАЙЄСІВСЬКОЇ СИСТЕМИ ЛОГІЧНОГО ВИВОДУ

Основні положення байєсівської системи логічного виводу

Запропонована в подальшому експертна система грунтується на першому алгоритмі, що одержав в застосуванні до експертних систем назву байєсівська система логічного виводу. Ось його суть.

1. Програма визначає кількість доступних захворювань та симптомів.
2. Програма ініціалізує апріорні імовірності P(H). Вона також виробляє деякі значення масиву правил RULEVALUE. Для кожного питання обчислюється RULEVALUE(I) = RULEVALUE(I) + ABS(P(H:E) — P(H:неE)), що відповідає значенню можливих змін ймовірностей усіх хвороб, до яких вони відносяться. Це робиться для того, щоб визначити, які питання (симптоми) є найважливішими, і з'ясувати, про що запитувати в першу чергу.
3. Програма знаходить найважливіше запитання і задає його. Існує ряд варіантів, що робити з відповіддю: ви можете просто сказати: "Так" або "Ні". Можете спробувати сказати: "Не знаю", — змін при цьому не відбудеться. Набагато складніше використати шкалу від -5 до +5, щоб виразити ступінь впевненості у відповіді.
4. Апріорні ймовірності заміняються новими значеннями при одержанні нових підтверджуючих доказів.
5. Підраховуються нові значення правил. Визначаються також мінімальне і максимальне значення для кожного захворювання, що грунтуються на існуючих на даний час апріорних імовірностях і припущеннях, що докази, що залишилися, будуть на користь гіпотези або суперечити їй. Важливо з'ясувати: чи варто дану гіпотезу продовжувати розглядати чи ні ? Гіпотези, які не мають сенсу, просто відкидаються. Ті ж з них, чиї мінімальні значення вищі певного рівня, можуть вважатися можливими наслідками. Після цього програма повертається до третього кроку і продовжує свою роботу.

Компонент TExpertDialog

Даний компонент дозволяє набути навичок у конструюванні експертних систем медико-діагностичного призначення. Експертні системи такого роду грунтуються на Байєсівській системі логічного виводу. Крім того, використовується методика "гнучких" відповідей. Тобто, відповідаючи на запитання: "Чи була останнім часом простуда чи подібне інфекційне захворювання ?" вибираємо значення із імовірнісного інтервалу [0,1]. Результати діалогу з експертом зберігається в автоматично створеному HTML-документі, тобто, можуть бути опубліковані на WWW.

База даних для збереження даних про захворювання та їх симптоми складається з трьох таблиць, як показано на рисунку 1.

Декларація компонента має вигляд:

```
TQuestionForm = class(TForm)
{declaration is ommited here}
end;

TExpertDialog = class(TSpeedButton)
private
```

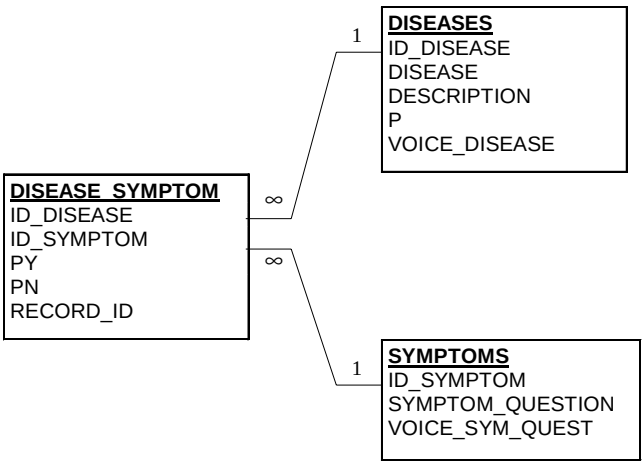


Рис. 1. Структурна таблиця, що використовуються у базі знань для компонента TexpertDialog.

```
        { Private declarations }
        FDataLinkGT: TDataLink;
        FDataLinkHT: TDataLink;
        FDataLinkET: TDataLink;
        FGeneralTable: TDataSource;
        FHypothesesTable: TDataSource;
        FEvidencesTable: TDataSource;
        function GetGeneralTable: TDataSource;
        function GetHypothesesTable: TDataSource;
        function GetEvidencesTable: TDataSource;
        procedure SetGeneralTable(Value: TDataSource);
        procedure SetHypothesesTable(Value: TDataSource);
        procedure SetEvidencesTable(Value: TDataSource);
        protected
        { Protected declarations }
        property DataLinkGT: TDataLink read FDataLinkGT;
        property DataLinkHT: TDataLink read FDataLinkHT;
        property DataLinkET: TDataLink read FDataLinkET;
        public
        { Public declarations }
        constructor Create(AOwner: TComponent); override;
        destructor Destroy; override;
        procedure Click; override;
        published
        { Published declarations }
        property GeneralTable: TDataSource read GetGeneralTable
write SetGeneralTable;
       property HypothesesTable: TDataSource read GetHypotheses Table
write SetHypothesesTable;
       property EvidencesTable: TDataSource read GetEvidences Table
write SetEvidencesTable;
      end;
```

Тут TQuestionForm — це компонент модальної форми для діалогу з експертною системою; найважливішими властивостями компонента TExpertDialog є GeneralTable (таблиця для збереження правил для заключень експертної системи),

HypothesesTable (таблиця для збереження ймовірних діагнозів), EvidencesTable (таблиця для збереження даних про симптоми).

Найпростіший додаток — експертна система з використанням TExpertDialog може бути розроблена в результаті наступних дій.

Необхідне програмне забезпечення

1. Інструментальна система Delphi із встановленим компонентом TExpertDialog.
2. Демонстраційна база даних медико-діагностичного призначення, встановлена в системі під псевдонімом MKBPardx.

Необхідні дії

1. Створіть каталог для збереження проекту.
2. Запустіть Delphi і створіть проект, вибравши з головного меню File — New Application
3. Виберіть на сторінці Data Access Палітри Компонент компонент Table і розмістіть три його примірники у вікні форми. За припущенням вони отримають імена Table1, Table2, Table3.
4. Виберіть на сторінці Data Access Палітри Компонент компонент DataSource і розмістіть три його примірники у вікні форми. За припущенням вони отримають імена DataSource1, DataSource2, DataSource3.
5. Виберіть на сторінці MedInfTraining Палітри Компонент компонент ExpertDialog і розмістіть його у вікні форми.
6. Встановіть властивості у такі значення:

для компонента Table1	
DatabaseName	MKBPardx
TableName	diseases
Active	True
для компонента	Table2
DatabaseName	MKBPardx
TableName	symptoms
Active	True
для компонента	Table3
DatabaseName	MKBPardx
TableName	disease_symptom

Active	True
для компонента	DataSource1
DataSet	Table1
для компонента	DataSource2
DataSet	Table2
для компонента	DataSource3
DataSet	Table3
для компонента	ExpertDialog1
HypothesesTable	DataSet1
EvidencesTable	DataSet2
GeneralTable	DataSet3

7. Збережіть проект у каталозі, створеному на кроці 1.
8. Відкомпілюйте і запустіть проект. Клацаючи кнопку ExpertDialog, ви розпочнете діалог з експертною системою.

На малюнках 2, 3, 4 можна побачити додаток, побудований за допомогою TExpertDialog. Зауважте, що він містить можливість голосового опитування. Для цього запитання у звуковій формі зберігаються у базі даних.

Рис. 2. Головне вікно однієї із розроблених експертних систем.

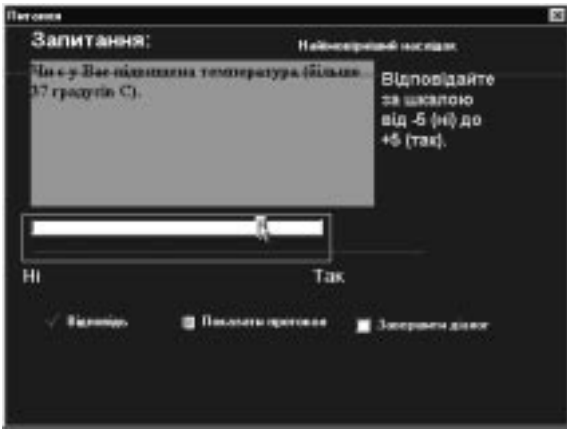


Рис. 3. Вікно запитання розробленої експертної системи.

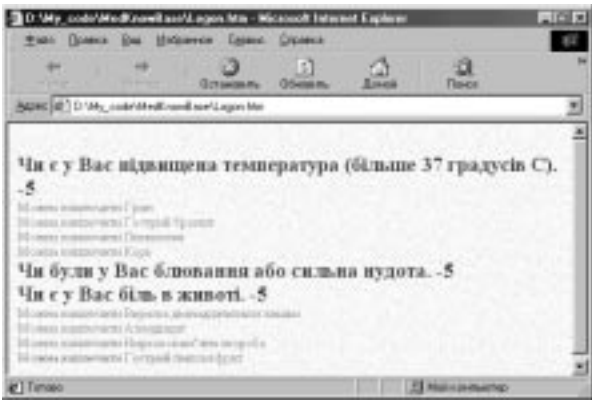


Рис. 4. Протокол зберігає діалог з експертною системою. Червоний колір позначає запитання, синій — відповіді, зелений — поради експерта.

Порядок роботи з програмою

Розпочати діалог Початок діалогу

Для того, щоб розпочати діалог з експертною системою, досить натиснути "швидку" клавішу "Розпочати діалог".

Як будується відповідь ?

Відповідь будується у вікні "Питання". Відповідь на запитання є цілочисельним значенням на проміжку від -5 (Ні) до +5 (Так). Проміжні значення виражають ступінь впевненості у відповіді. Скажімо, вибір в якості відповіді значення "+2" означає приблизно таке: будучи впевненим майже наполовину (а точніше з імовірністю 0.4), я можу сказати "Так".

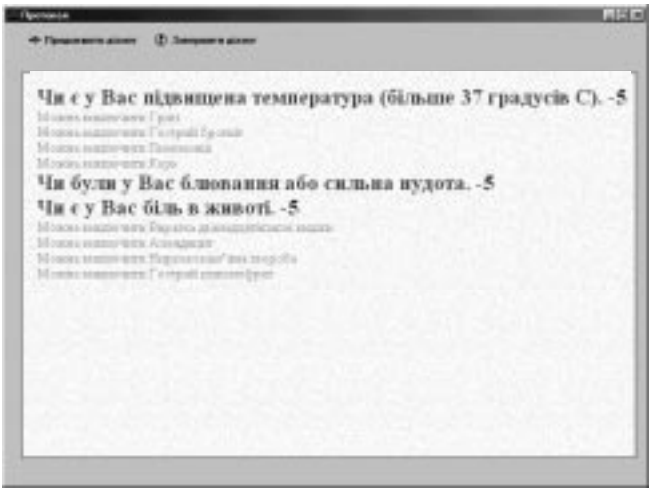
Як вказати відповідь у вікні "Питання"?

1. Необхідно за допомогою елемента керування TrackBar встановити чисельне значення Вашої відповіді.
2. Прийнятні два шляхи. Якщо Ви бажаєте безпосередньо перейти до наступного питання, то натисніть "швидку" кнопку **Відповідь**. Якщо ж Ви бажаєте ввести відповідь і потім

переглянути протокол усього попереднього діалогу, клацніть Показати протокол.

Що таке протокол?

Протокол ведеться з метою майбутнього аналізу процесу логічного виводу експертної системи. Крім того, сюди записується інформація про гіпотези, що можна виключити із розгляду.



Як повернутися із перегляду протоколу у режим ведення діалогу?

Необхідно натиснути "швидку" кнопку Продовжити діалог.

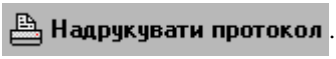
Як завершити діалог?

Якщо активним є вікно "Питання", то встановіть прапорець Завершити діалог і натисніть "швидку" кнопку Відповідь.

Якщо ж Ви у вікні перегляду протоколу, то натисніть 

.

Як після завершення діалогу переглянути його протокол ?

Необхідно в головному вікні програми натиснути "швидку" кнопку 

.

Експертна система

Експертна система — це складна програма, що оперує знаннями у вузькій області, що погано формалізується, з метою одержання задовільного або ефективного розв'язку.

Відповідь

Відповідь на запитання є цілочисельним значенням на проміжку від -5 (Ні) до +5 (Так). Проміжні значення виражають ступінь впевненості у відповіді.

Запитання та вправи

1. Використовуючи компонент TExpertDialog, розробіть експертну систему діагностики вузького класу захворювань, наприклад: інфекційні захворювання, онкологічні захворювання, захворювання черевної порожнини і ін. (Для цього слід заново сформувати базу знань під псевдонімом MKBPardx)
2. Самонавчання є однією з визначальних рис професійних експертних систем. Яким чином можна реалізувати можливість самонавчання для експертних систем на основі компонента TExpertDialog.

СИСТЕМИ ПІДТРИМКИ РІШЕНЬ В МЕДИЦИНІ

Процеси набуття знань, а згодом їх опанування, є досить складними (та дорогими). Дану проблематику можна знайти і в медицині. Так, задачі діагностики та задачі лікування передбачають прийняття рішення високопрофесійними фахівцями-лікарями. При цьому використовується велика множина різноманітних показників (аналізів). Якому з них віддати більшу перевагу — відповідь на таке запитання часто приймається завдяки власній інтуїції. Виявляється, подібні задачі успішно моделюються за допомогою програмного забезпечення, про яке піде мова далі.

Вимоги до медичних систем підтримки рішень

Медичні системи підтримки рішень (СПР):

- **повинні** бути в змозі пояснити свої діагностичні і лікувальні рішення лікарям-споживачам;
- **повинні** відображати (демонструвати) розуміння своїх власних медичних знань;
- **повинні** відображати загальний зміст (сенс), отже, системи підтримки рішень повинні представляти *знання*.

Способи представлення знань

Знання можуть представлятися за допомогою:

- правил: цей спосіб використовується у традиційних експертних системах;
- логіки предикатів першого порядку: це так званий формальний, математичний підхід;
- фреймів: тут використовується об'єктно-орієнтований підхід;
- використання апарату теорії ймовірностей.

Інформація про набуті знання зберігається в СПР у вигляді бази знань.

Типи баз знань (БЗ), що використовуються в медичних СПР

БЗ типу Internist-1

Орієнтована на використання спеціального алгоритму Internist-1. Містить об'ємну базу даних про захворювання, симптоми та зв'язки між ними. Для захворювання зберігаються частотні характеристики.

БЗ типу довідника

Зберігає інформацію у вигляді блок-схеми дій. Включає точки розгалуження, критерії прийнятності, описи дій.

БЗ клінічних експериментів

Включає атрибути, специфічні для предметної області, для збереження інформації про клінічні експерименти:

- інформація про токсичність ліків;
- деталізований критерій прийнятності;
- інформація про рецепт і дозування ліків;
- вимоги до звіту.

Застосування медичних баз знань

Як можуть використовуватися медичні бази знань?

1. БЗ типу Internist-1 використовуються для діагностики.
2. БЗ типу довідників та клінічних експериментів використовуються для:
   - підтримки рішень;
   - визначення прийнятності.

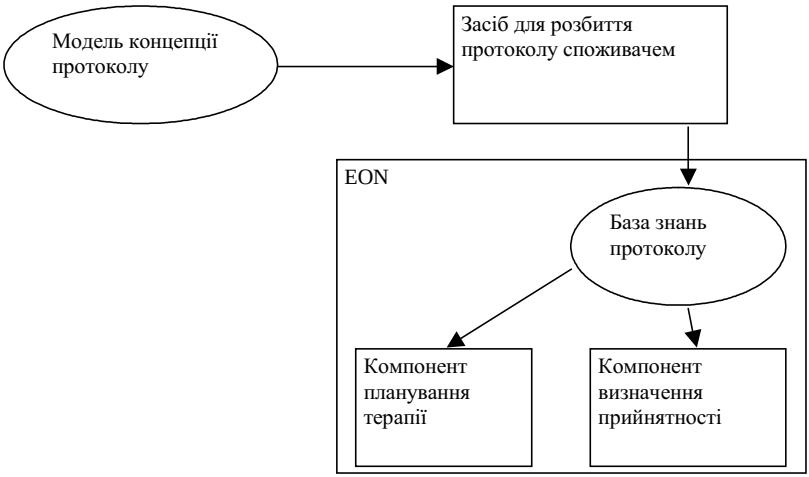


Рис. 1. Модель застосування БЗ типу довідника.

Проблеми використання СПР з БЗ типу довідника

Підтримка рішень і лікування на основі довідника вимагає системи електронних медичних записів. Це вимагає відповідності між елементами із запису пацієнта та відповідними полями бази знань довідника.

Еволюція медичних СПР

Mycin. Це була експертна система, що включала механізм виводу та база правил. Розв'язувала проблему діагностики деяких інфекційних захворювань.

Oncocin. Працювала в предметній області дослідження раку. Розв'язувала задачу планування терапії.

Opal. Виступала як засіб набуття знань для Oncocin. При цьому задача підтримки рішень все ще складно реалізовувалася.

Оболонки експертних систем

З технічної точки зору можна розглядати наступну формулу:

Експертна система = механізм виводу + правила.

При цьому правила є:
- — специфічними для предметної області;
- — керованими "вручну".

Механізм виводу є:
- — незалежним від предметної області;
- — надається розробником (можливо комерційний продукт)

Рис. 2. Набір правил (бази знань).

Рис. 3. Структурна схема експертної системи.

Проблеми розв'язування задач за допомогою баз знань

На сьогодні бази знань є все ще дорогими, щоб набувати, і складними, щоб підтримувати (зберігати). Для зменшення таких затрат вибирають такі шляхи:

- використовують специфічні для проблемних областей засоби набуття знань, що роблять конструювання бази знань простішим;
- з'єднують бази знань;
- повторно використовують методи розв'язання проблем (МРП).

Експертні системи (ЕС) 2-го покоління

Їх метою є повторне використати обох компонент (бази знань та методу розв'язання проблеми). При цьому база знань використовується багатьма методами розв'язання проблем. Метод розв'язання проблеми використовується багатьма базами знань.

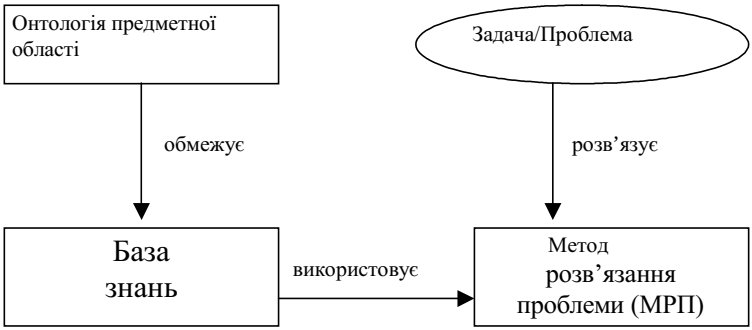


Рис. 4. ЕС 2-го покоління.

Поняття онтології

Існують різні трактування поняття онтології. Так, онтологія — це:

— галузь метафізики, що пов'язана з природою буття;
— опис концептуалізації. Під концептуалізацією мають на увазі словник термінів поєднаних предметних областей;
— онтологічна теорія містить формули, що вважаються завжди правильними незалежно від конкретних обставин;
— онтологія інформує інших про загальні істини і припущення через посередництво бази знань.

ЕС 2-го покоління на основі БЗ типу довідників

Роботу такої ЕС спрощено можна уявити так. На вході ЕС отримує модель (онтологію) деякої проблемної області у вигляді довідника спеціального формату (про це піде мова далі). На ви-

ході ми отримуємо засіб для набуття знань для проблемної області (наприклад, засіб для медичних практичних керівництв).

Модель довідника формату GLIF

Формат GLIF (Guideline Interchange Format) широко використовується у БЗ, основаних на фреймах. Його структурна схема наведена на рисунку.

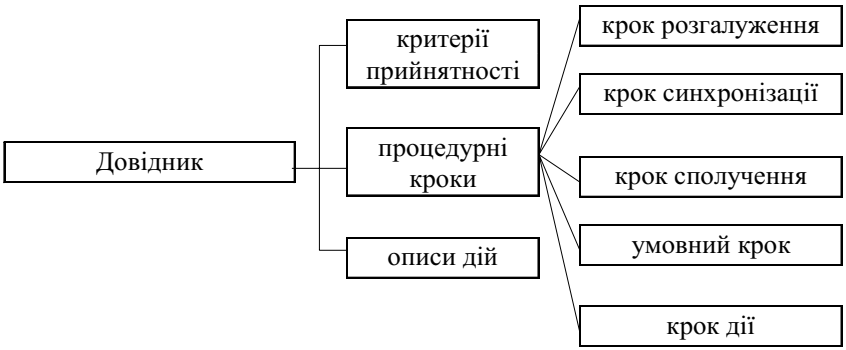


Рис. 5. Модель довідника.

Процедурні кроки

Довідник складається з пов'язаних кроків:

- кроки дії відповідають клінічним втручанням та діям із збору даних;
- умовні кроки визначають розгалуження if-then-else;
- кроки розгалуження та синхронізації моделюють неоднозначність.

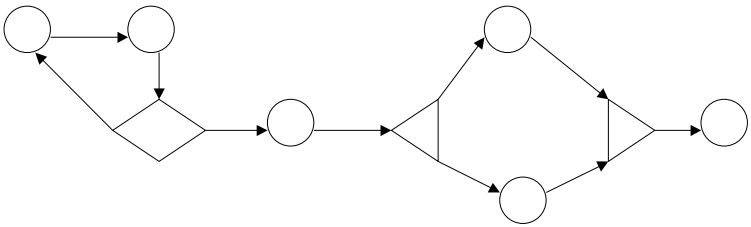
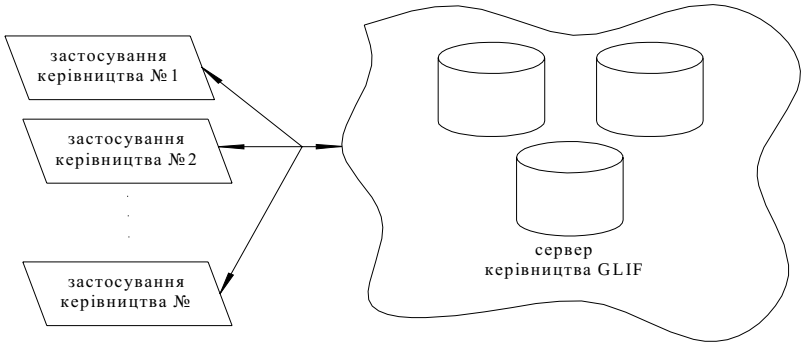


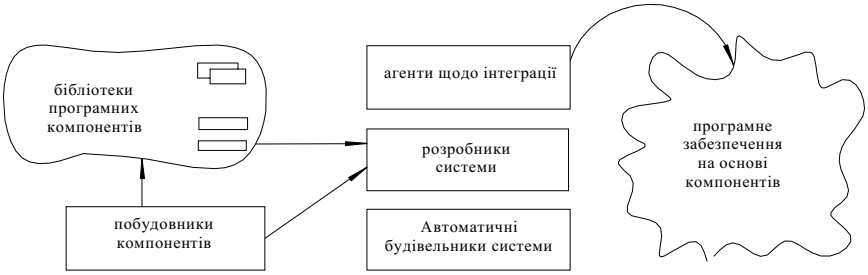
Рис. 6. Процедурні кроки довідників використовують спеціальні позначення.

Повторне використання ЕС

Як вже зазначалося, істотною перевагою ЕС 2-го покоління є повторне використання баз знань та механізмів виводу при розв'язуванні нових завдань. Так, повторне використання довідників може бути налаштоване за допомогою сервера довідників:



Повторне використання програмного забезпечення ЕС може бути організоване таким чином:



Підсумки

Отже, бази знань є важливими в тих медичних застосуваннях:

- яким необхідно маніпулювати та отримувати знання;
- яким потрібно використовувати та отримувати інформацію;
- яким потрібна гнучка, адаптивна база знань.

Все ще відкритими в сучасних ЕС залишаються проблеми:

- найвдалішого способу представлення знань;
- об'єднання та повторного використання знань;
- зв'язків між БЗ та системою запису пацієнтів;
- розробки розповсюджуваних та співпрацюючих БЗ.

ОГЛЯД ВІЗУАЛЬНИХ КОМПОНЕНТ DELPHI

Основні поняття

Компонент — це клас, що походить безпосередньо чи не безпосередньо з TComponent із спеціальними шляхами доступу до його даних, надрукованими властивостями та іншими рисами, що задовольняють вимоги Delphi до візуальних чи не візуальних компонент.

Візуальний компонент — це об'єкт з візуальним представленням під час виконання, такий як кнопка або текстовий коментар, що ви вставляєте у вікно форми. Візуальні компоненти можуть чи ні з'являтися на Палітрі Компонент.

Форма — це також компонент, що може містити об'єкти інших компонент.

Об'єкт компонента — це інстанція компонента, вставлена у форму.

Вступ

Візуальний компонент — це один з об'єктів, таких як кнопка або текстовий коментар, що ви вставляєте у вікно форми. Насправді, форма також є компонентом — таким, що може містити інші об'єкти компонентів. При розробці додатків в Delphi ви витрачаєте більшість свого часу, вставляючи та модифікуючи об'єкти візуальних компонент, отже, необхідно зрозуміти їх як слід. Візуальні компоненти — це свого роду будівельні блоки, на яких примощується фундамент вашого коду.

Бібліотека візуальних компонент

З бібліотекою візуальних компонент Delphi (VCL) ви зможете створювати корисні додатки лише з домішками програмування. Краще за будь-які слова це підтвердить розробка додатка-прикладу на цій лекції.

Категорії візуальних компонент

Для того, щоб вибрати категорію компонент, натисніть закладку сторінки, наприклад, Dialogs, внизу Палітри Компонент Delphi (рис.1). Таблиця 1 перераховує та описує категорії усіх візуальних компонент Delphi. Порядок та кількість категорій на вашому екрані може відрізнятися залежно від опцій інсталяції, або ж встановлених додаткових компонент, або ж ви використали Options|Environment, щоб впорядкувати значки на палітрі.

Рис. 1. Палітра візуальних компонент Delphi містить закладки сторінок, що організують компоненти у категорії.

Стандартні компоненти

Із стандартними компонентами ви можете створювати об'єкти-елементи керування Windows та елементи спільного інтерфейсу, такі як кнопки, прапорці і меню. Оскільки стандартні компоненти є відносно легкими до використання, вони роблять добрий вступ до програмування візуальних компонент.

Розробка додатків з об'□ктів компонент

Якщо Delphi зараз не працює, то запустимо її. Якщо вже завантажено якийсь додаток, використаємо File|New Project, щоб приготувати вікно нової форми. Натиснемо закладку сторінки Standard під Палітрою Компонент, щоб відобразити значки компонент цієї категорії.

Виберемо значок компоненти. Наприклад, натиснемо кнопку, помічену великою літерою A, що представляє компонента Label (мітка). Перемістимо курсор у вікно форми і натиснемо ліву кнопку мишки, щоб вставити об'єкт компонента Label у форму. Точне розміщення не має ваги. Не тягніть компонента у форму; за-

Таблиця 1. Категорії візуальних компонент Delphi.

Категорія	Опис
Standard	Стандартні елементи керування Windows, такі як кнопки, мітки, поля вводу, списки, прапорці та смуги прокручування
Additional	Елементи керування споживача, такі як біт-кнопки з графічними зображеннями (гліфами), швидкі кнопки панелі інструментів, сітки електронних таблиць, деревоподібні списки, зображення бітмапів, графічні поверхні
Data Access	Компоненти для доступу до баз даних через таблиці та SQL-запити, для створення звітів з використанням додатка Borland's ReportSmith, що включено у Delphi
Data Controls	Компоненти для роботи з інформацією баз даних, використовуючи елементи керування, такі як навігатори, поля редагування, коментарі, малюнки, прапорці і списки
Dialogs	Спільні діалогові вікна для таких задач, як вибір файлів і каталогів, вибір шрифтів і кольорів, налаштування друкування, пошук і заміна даних у документах
System	Додаткові компоненти для створення таймерів, областей малювання, інструментів каталогів диска, мультимедіа-додатків, для спільного використання даних через OLE і DDE
VBX	Компоненти, що служать інтерфейсом до елементів керування Visual BASIC Extended (VBX). Довільний елемент керування VBX— не лише ті, поставляються з Delphi — може бути візуальним компонентом.
Samples	Зразкові компоненти, вихідний програмний код яких знаходиться у каталозі Delphi Source\Samples. Ці компоненти у більшості є керівництвами для створення ваших власних компонент, крім того, вони можуть виявитися корисними у додатках

мість цього, натисніть значок компоненти, перемістіть вказівник мишки у форму і натисніть знову, щоб вставити об'єкт.

Примітка. Для того, щоб уникнути двозначності, означимося, що термін компонент має на увазі значок у Палітрі VCL; об'єкт компоненту — це інстанція компонента, вставлена у форму. Якщо від вас вимагається натиснути об'єкт або об'єкт компонента, виберіть його у вікні форми.

Додамо кілька інших об'єктів компонент у форму, вибираючи їх з палітри Standard і натискаючи мишкою у вікні форми. Після того, як ми вставили кілька об'єктів (точне число і типи не мають значення), спробуємо провести наступні експерименти, щоб познайомитися з командами організації компонентів.

Для того, щоб перемістити та змінити розміри компонента, спершу виберемо його мишкою, (delphi автоматично вибирає щойно створені компоненти). Це оточує компонент маленькими прямокутниками, що називаються кермами (хендлами). Для того, щоб змінити вибраний розмір компонента, натисніть і перетягніть кермо. Для того, щоб перемістити об'єкт у формі, натисніть всередині компонента і перетягніть мишку.

Натисніть і перетягніть мишку в задньому плані форми, щоб оточити кілька об'єктів компонента рамкою "з каучукової стрічки". Відпустіть мишку, щоб вибрати об'єкти у рамці. Після цього ви можете натискати і перетягувати всередині вибрані об'єкти, щоб перемістити їх разом. Delphi відображає бліді керма довкола усіх вибраних об'єктів компонента. Ви не зможете використати ці керма для зміни розмірів кількох об'єктів — ви можете лише змінювати розміри індивідуальних об'єктів компонента — одного одночасно.

Далі наведемо інший спосіб для вибору кількох об'єктів. Натисніть перший об'єкт, щоб його вибрати, і тримайте натисненою кнопку Shift, натискаючи мишкою інші об'єкти.

Виберіть Edit|Select All, якщо ви бажаєте вибрати всі об'єкти у формі.

Коли вибрано кілька об'єктів, натисніть будь-який об'єкт одночасно з кнопкою Shift, щоб зняти з нього вибір. У багатьох випадках для того, щоб вибрати велике число об'єктів, легше вибрати їх усіх, а потім зняти вибір з деяких.

Для того, щоб зняти вибір з об'єктів, натисніть мишкою задній план форми. Або, якщо вікно форми активне, натисніть Esc (це добрий спосіб для використання, коли один або більше об'єктів компонента повністю заповнюють форму, що робить складним або неможливим натиснути на задній план вікна). Ви знаєте, що всі об'єкти компонент не знаходяться під вибором, коли не видно жодного керма. Це важливо мати на увазі, тому що ви можете змінювати властивості та події форми лише коли не вибрано жодного об'єкта компонента.

Для того, щоб вирівняти кілька об'єктів компонент, виберіть їх і виберіть команду Edit|Align, яка виводить діалоговий прямокутник. Використовуйте кнопки у цьому діалозі для вирів-

нювання ребер, рівномірного розміщення елементів керування, виконання інших впорядкувань. Дещо повправляйтеся у опціях діалогу, щоб познайомитися з командами впорядкування.

Коли один об'єкт компонента накриває інший, використовуйте Edit|Bring to Front та Edit|Send to Back для зміни порядку відносного відображення об'єктів. Для того, щоб відпрацювати ці команди, вставте кілька об'єктів-прапорців (CheckBox) у форму, і тоді вставте об'єкт GroupBox (приклад компонента-контейнера). Якщо ви переміщаєте GroupBox над кнопками, він покриває їх. Для того, щоб кнопки знову з'явилися, виберіть GroupBox і виберіть Edit|Send to Back, щоб сховати його за кнопками (якщо, здається, що нічого не відбулося, причина може полягати у тому, що ви вибрали кнопки і GroupBox. Щоб вибрати лише GroupBox, спочатку зніміть вибір з усіх об'єктів, і тоді натисніть GroupBox.)

Під час виконання споживачі можуть натискати кнопку Tab, щоб перемістити фокус вводу з одного елементу керування до іншого (фокус вводу посилається на поточний елемент керування, що отримує будь-яку діяльність з клавіатури.) Для встановлення порядку для клавіші tab, виберіть деякі з усіх об'єктів компонент і виберіть команду Edit|Tab Order. Отримаємо діалоговий прямокутник, який можна використати для опису порядку, в якому кожен елемент керування стає активним у відповідь на натиснення клавіші Tab. Команди діалогу є зрозумілими — просто натискайте і перетягуйте імена об'єктів компонент, щоб їх впорядкувати або виберіть ім'я і натисніть кнопки "вверх" або "вниз", що розміщені в діалозі.

Коли ви вибираєте два або більше об'єктів компонент, вікно Інспектора Об'єктів показує їх спільні властивості та події. Довільні зміни цих значень впливають на всі вибрані об'єкти. Наприклад, щоб змінити стиль тексту для набору прапорців, спершу виберіть їх і тоді введіть ваші налаштування у спільну властивість Font.

Порада. Майте на увазі, що ви переглядаєте спільні властивості і події, коли список Інспектора Об'єктів, що розгортається, є порожнім.

Додаток-приклад: MemoPad

Рис. 2. MemoPad демонструє компоненти Label і Memo.

Додаток-приклад MemoPad показує, як використовувати компоненти Label та Memo. Програма також демонструє створення простого рядка команд меню. Малюнок 2 показує дисплей програми MemoPad.

Для негайного запуску додатка MemoPad виберемо File|Open Project, і відкриємо Memopad.Dpr. Натисніть F9, щоб відкомпілювати, лінкувати і запустити програму. Наберіть кілька поміток, тоді вийдіть з програми MemoPad і поверніться у Delphi, де ви тепер можете керувати формою і компонентами додатка.

Вибирайте команди MemoPad File|Save для того, щоб зберегти ваші коментарі у текстовому файлі Memos.Txt в поточному каталозі. MemoPad завжди використовує це ім'я файла. Виберіть File|Exit для завершення MemoPad, що також автоматично зберігає ваші коментарі (ви можете використовувати довільний інший спосіб для виходу з програми.) Вибір Help|About відображає діалоговий прямокутник-повідомлення, що описує додаток.

Дві з команд меню програми-прикладу мають відповідні клавіші акселератори, які споживачі можуть натиснути для вибору команд меню без використання мишки. Натисніть F2 для збереження ваших коментарів. Натисніть F1 для відображення діалогового прямокутника-повідомлення. До того ж, команди меню мають підкреслені "гарячі" клавіші. Наприклад, натисніть Alt+F X для вибору File|Exit.

Програма MemoPad також використовує компонента PopupMenu для того, щоб надати інший альтернативний спосіб для виконання команд. Перемістіть вказівник мишки будь-де всередині вікна MemoPad — але не над рядком меню — і натисніть праву кнопку мишки. Це відкриє мале pop-up меню, що працює

подібно до головного меню, але з'являється біля позиції мишки. Для закриття вікна цього меню натисніть правою кнопкою мишки поза pop-up меню або виберіть команду (або натисніть Esc).

Завжди надавайте споживачу різні методи для виконання загальних команд. При цьому початківці зможуть вибирати команди з меню додатка (що відносно просто вивчити), а експерти зможуть використовувати гарячі клавіші та pop-up меню для швидкого виконання команд.

Розробка додатка-прикладу MemoPad

Після того, як ви познайомилися з додатком MemoPad, за допомогою Delphi дослідимо форму програми та компоненти. Покинемо зараз MemoPad та повернемося у Delphi. Вибираємо кожен з об'єктів компонент програми у вікні форми через одне натиснення мишкою і переглядаємо їх властивості та події у вікні Інспектора Об'єктів.

Перетворимо у звичку одинарне натиснення об'єкта компоненти у вікні форми. Для більшості компонент при подвійному натисненні їх об'єктів Delphi створює процедуру у вхідному коді програми для події за припущенням, переважно OnClick. Якщо це трапляється випадково, виберіть Edit|Undo для відміни (або натисніть Ctrl+Z або Alt+Backspace). Для відміни змін з часу останнього збереження ще раз відкрийте проект і дайте No, якщо Delphi порадить зберегти ваші зміни. Однак, якщо ви випадково створили обробник події, що вам не потрібен, не турбуйтеся — до тих пір, поки ви не додасте жодних тверджень чи коментарів у процедуру, Delphi знищить його автоматично під час наступної компіляції проекту.

Також поглянемо у модуль Main (вікно із написом MAIN.PAS). Якщо ви не можете знайти це вікно, перенесіть його на верх завдяки командам View|Units або натиснувши швидку клавішу Toggle Form/Unit. Main.Pas містить вхідний код Object Pascal, який Delphi синхронізує із формою і її візуальними об'єктами компонент. Під час розробки додатка ви часто будете переключатися між формою та вікном пов'язаного з нею модуля.

Доцільно розглядати форму та відповідаючий їй модуль як різні представлення тих же об'єктів. Форма показує візуальне

представлення вікна та його елементів керування. Модуль показує команди Object Pascal, що виконують дії компонент, такі як відповіді на натиснення кнопки чи збереження даних у файлі.

Не намагайтеся зрозуміти всього в програмуванні файла Main.Pas, лістинг якого наведено нижче. Файли такого типу наводяться лише для посилання. Насправді ж Delphi створює більшість програмного тексту автоматично, вам не доведеться набирати лістинг.

Зараз спробуємо створити наново додаток MemoPad. При виникненні ускладнень ви можете звернутися до лістингу. Отже, виконуємо одна за одною наступні вказівки.

1. Використаємо програму типу Explorer для створення каталогу типу C:\Projects\Memopad для збереження файлів вашого додатка. Якщо ви не бажаєте зберігати додаток, ви можете зберігати файли у каталозі Temp і знищити їх потім.
2. Розпочнемо новий проект. Натисніть всередині форми, щоб її вибрати, і змініть властивість Caption у вікні Інспектора Об'єктів на Memo Pad. З цього місця і надалі, коли я радитиму змінити властивість, спершу виберіть компоненту на формі, і якщо потрібно, натисніть закладку Properties у вікні Інспектора Об'єктів. Тоді виберіть властивість і змініть її значення, що показане праворуч властивості.
3. Змінимо властивість форми Name на MainForm. Це одне слово без пропусків. Якщо би ви набрали недопустимі символи у властивість Name, Delphi виведе повідомлення про помилку. Імена повинні починатися з літери або знаку підкреслення, і вони повинні включати лише букви, цифри і знаки підкреслення.
4. Виберіть File|Save Project, або ж натисніть "швидку" кнопку Save project для того, щоб створити файли проекту. Ви можете зачекати із збереженням проекту, але якщо ви зробите це зараз, то тим самим створите імена файлів, які Delphi використовує для створення різних елементів у тексті програми. Я суворо наполягаю на негайному збереженні проекту, наскільки це можливо після найменування форми головного вікна.
5. Delphi пропонує вам два файлові діалоги, один за одним. У першому діалозі, під назвою Save Unit1 As, змініть каталог,

на той, що створений на кроці 1, і наберіть Main у полі для вводу File Name. Натисніть Enter або натисніть мишкою OK. У наступному другому діалозі, поміченому як Save Project1 As, наберіть Memopad для імені файла. Натисніть Enter або OK. Ви щойно створили два файли: Main.Pas і Memopad.Dpr. Delphi дописує розширення .Pas і .Dpr до імен ваших файлів.

6. Вставте компонент MainMenu у форму. А також вставте компонент PopupMenu. У цьому прикладі ви використовуватимете для ідентифікації об'єктів імена за припущенням: MainMenu1 і PopupMenu1. Оскільки вони є відносно складними компонентами, Delphi виводить їх у вікні форми як значки, що представляють їх остаточну появу. Значки є невидимими під час виконання. Перемістіть обидва значки у лівий верхній кут форми. Точне розташування не має значення.
7. Для того, щоб створити команди меню, двічі натисніть об'єкт компонента MainMenu1, що відкриє підпрограму Delphi Menu Designer. Для того, щоб створити пункт меню File, наберіть &File, і натисніть Enter для зміни властивості Caption для цього об'єкта меню. Набирайте перед літерою амперсанд (&), щоб позначити для команди "швидку" клавішу з Alt.
8. Тепер Menu Designer висвітлює наступне місце для вставки команди, в даному випадку, нижче меню File. Введіть &Save, щоб створити команду Save. Не натискайте Enter (якщо ви це зробили, використайте клавішу "стрілка вверх", щоб повторно висвітлити команду.) Виберіть властивість ShortCut, натисніть її стрілку вниз, і виберіть "швидку" клавішу для цієї команди F2.
9. Натисніть порожній простір нижче команди Save в Menu Designer, і виберіть властивість Caption. Введіть E&xit для створення клавіші виходу і позначте літеру x як "швидку" клавішу з Alt для команди (примітка: коли ви вводите команди нового меню, завжди вибирайте властивість Caption перед набором тексту команди.)
10. Використовуючи подібну методику, введіть меню Help і його команду About. Позначте F1 як "швидку" клавішу для команди таким же шляхом, як ви позначили F2 для Save.
11. Закрийте вікно Menu Designer, щоб побачити ваше меню у вікні форми.

12. Двічі натисніть компонент PopupMenu, щоб повторно відкрити Menu Designer, який цього разу виводить простіший стиль для меню pop-up. Введіть команди About, Save і Exit у це меню. Перед літерами, що повинні бути підкресленими, введіть амперсанди. Використайте властивість ShortCut, щоб позначити "швидкі" клавіші, такі як F1 і F2.
13. Недостатньо просто створити об'єкт PopupMenu object — ви повинні також наказати формі, щоб використати його. Закрийте вікно Menu Designer, і тоді натисніть всередині заднього фону форми, щоб зняти вибір з будь-яких компонентів. Ви повинні побачити у списку, що розгортається, Інспектора Об'єктів: MainForm: TmainForm. Змініть властивість форми PopupMenu на PopupMenu1. Ви можете набрати це ім'я або ж вибрати його із списку, що розгортається, що є у властивості. Це запрограмує форму для виводу об'єкта PopupMenu1, коли споживач натискає праву кнопку мишки.
14. Натисніть F9, щоб відкомпілювати і запустити додаток, який на цій стадії ще незавершений. Я переважно роблю це негайно після створення команд програми і таким чином я перевіряю їх появу. Перепробуйте команди меню програми (жодна з яких поки що не працює), і натисніть праву клавішу мишки для виводу меню pop-up. Для повернення у Delphi або натисніть Alt+F4, або двічі натисніть кнопку системного меню, або натисніть кнопку Windows 95 "закрити".

Щойно ви завершили перший етап програмування MemoPad. Виконайте наступні кроки для завершення розробки додатка.

1. Вставте компонент Label у вікно форми і змініть його властивість Name на MemoLabel. Натисніть Enter і зауважте, що Delphi встановлює властивість Caption в той же текст.
2. Часто буває не бажано показувати внутрішнє ім'я об'єкта компонента. Для зміни виведеного тексту мітки виберіть MemoLabel і змініть його властивість Caption на Memos.
3. Вставте об'єкт компонента Memo у форму. Цього разу ви можете залишити властивість об'єкта Name, встановлену у значення за припущенням, тобто Memo1. Однак знищіть всі символи з властивості Lines, щоб область виводу стала поро-

жньою. Щоб зробити це, виберіть у Memo1 властивість Lines і натисніть кнопку редагування праворуч. Це відкриє редактор списків Delphi, який ви можете використати, щоб знищити рядок Memo1 (натисніть backspace кілька разів або помітьте текст і натисніть Del). Натисніть Enter або виберіть кнопку редактора OK, щоб виконати ці зміни.

4. Змініть розміри Memo1 і налаштуйте всі компоненти, щоб отримати вигляд, подібний до того, що на малюнку. На цьому місці ви можете захотіти зберегти проект і скомпілювати та запустити його, щоб поглянути на остаточний вигляд додатка. Порада: не чекайте моєї поради, щоб випробувати ваші додатки. Ви можете натискати F9, щоб скомпілювати і запустити програму майже на будь-якому етапі його розробки.
5. Для того, щоб запрограмувати команди MemoPad, у вікні форми виберіть команду програми File|Save (не вибирайте команду в додатку, що виконується, вибирайте її саме у вікні форми Delphi.) Вибір цієї команди меню вставляє процедуру, що називається обробником події (хендлом події), у текст програми і розміщує курсор у місце, де ви можете набирати твердження, щоб виконати дії команди. Введіть наступне твердження між begin і end. Текст у дужках повинен бути взятий в одинарні лапки. Не використовуйте подвійні лапки. Зверніться до Memopad\Main.pas, якщо ви помилилися.

   Memo1.Lines.SaveToFile('memos.txt');
6. Зробимо деякі пояснення. Твердження, що ви щойно ввели, зберігає рядки Memo1 у файлі з іменем Memos.Txt. Lines є об'єктом, яким володіє Memo1. SaveToFile — це метод, який може виконувати Lines. Крапки вказують на те, що Memo1, Lines і SaveToFile є пов'язаними. Рядок 'memos.txt' представляє ім'я файла. Твердження передає рядок в дужках методу SaveToFile. Твердження закінчується крапкою з комою. На технічній мові кажучи, в мові Pascal крапка з комою відділяє одне твердження від іншого. В даному випадку, оскільки немає наступних тверджень, ви можете опустити крапку з комою; однак вона не завдасть шкоди, якщо її все ж вставити.
7. Доробіть інші команди MemoPad. Виберіть Select File|Exit у формі. Введіть Close між ключовими словами begin та end

(що стосуються процедури TMainForm.Exit1Click в Main.Pas). Як ви вже знаєте, процедура Close одна із сотні, що ви можете викликати, закриває вікно. Якщо, як в даному випадку, це головне вікно програми, Close також завершує програму.

8. Повертаємося до вікна форми і вибираємо Help|About. Введіть наступне твердження між begin та end. Точно наберіть кожен символ, як це показано.

   MessageDlg('Memo Pad'#13#10'©'#13#10'Version 1.00', mtInformation, [mbOk], 0);

9. Твердження створює діалог повідомлення, що виводить повідомлення про MemoPad copyright і номер версії. У процедуру MessageDlg програма передає чотири аргументи у дужках, відділені комами. Перший аргумент — рядок, що взятий в одинарні лапки. У цьому рядку позначення #13#10 вставляє ASCII-коди керування "повернення каретки" і line feed, щоб розпочати подальший текст на нових рядках. Для того, щоб ввести символ copyright, використовуємо утиліту Windows Character Map, або вирізаємо і вставляємо символ з будь-якого іншого файла. Аргумент mtInformation вибирає стиль діалогу. Аргумент [mbOk] є множиною, яка в цьому прикладі має лише один член mbOk, що описує стиль діалогового прямокутника з кнопкою OK. Аргумент 0 є місцем для допоміжного контексту діалогу, який MemoPad не використовує. Знову ж крапка з комою завершує твердження (для більшої інформації про ці аргументи дивіться MessageDlg у довіднику Delphi.)

10. Для того, щоб запрограмувати команди меню pop-up, двічі натисніть об'єкт PopupMenu1, що ви вставили у форму. Це знову ж відкриє вікно Menu Designer. Натисніть закладку сторінки Events у вікні Інспектора Об'єктів. Виберіть команду About в Menu Designer і встановіть подію OnClick в About1Click. Ви можете ввести ім'я цього обробника події або його із списку, що розгортається OnClick. Встановіть для команди Save обробник події OnClick в Save1Click. Встановіть для команди Exit обробник події OnClick в Exit1Click. Це ті ж обробники подій, що ви програмували для головного меню — підтвердження того, як дві або більше події можуть поділяти той же код.

11. Закрийте вікно Menu Designer. Натисніть всередині заднього фону вікна форми і, якщо потрібно, виберіть закладку сторінки Events у вікні Інспектора Об'єктів. Двічі натисніть порожній простір праворуч події OnActivate, і введіть наступне твердження if-then-else між begin та end. Твердження читає текст вказаного файла в об'єкт Lines компонента:

```
if FileExists('memos.txt')
 then Memo1.Lines.LoadFromFile('memos.txt')

 else Memo1.Lines.SaveToFile('memos.txt');
```

12. Попереднє твердження викликає функцію FileExists, яка повертає True, якщо файл є на диску, або False, якщо ні. Якщо файл існує, твердження читає файл у рядки Memo1. Якщо ж ні, то твердження зберігає рядки в новому файлі. Оскільки властивість об'єкта Lines порожня на цьому етапі, то виклик методу SaveToFile створює порожній новий файл. Хоча може виявитися, що багато операцій мають місце, повна конструкція if-then-else насправді є єдиним твердженням, і, отже, вимагається лише одна крапка з комою для відділення цього твердження від інших. Як і в попередньому випадку, оскільки немає наступних тверджень, крапка з комою суворо не вимагається.
13. Виберіть форму (імовірно, вона вже вибрана), і двічі натисніть простір праворуч події OnClose. Delphi вставляє процедуру для цієї події, яка трапляється, коли ви закриваєте вікно програми. Введіть наступне твердження між begin і end для збереження рядків компонента у вказаному файлі:

```
Memo1.Lines.SaveToFile('memos.txt');
```

14. Натисніть F9, щоб відкомпілювати, зв'язати і запустити завершену програму. Випробуйте команди і поверніться у Delphi. Порівняйте ваш програмний файл Main.Pas з наведеним у додатку лекції. Деякі процедури можуть мати відмінний порядок, але твердження та загальне розміщення повинне бути те ж.

```
{Листинг Memopad\Main.Pas}
unit Main;

interface
```

```
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
Controls,
  Forms, Dialogs, Menus, StdCtrls;

type
  TMainForm = class(TForm)
  MainMenu1: TMainMenu;
  PopupMenu1: TPopupMenu;
  File1: TMenuItem;
  Save1: TMenuItem;
  Exit1: TMenuItem;
  Help1: TMenuItem;
  About1: TMenuItem;
  About2: TMenuItem;
  Save2: TMenuItem;
  Exit2: TMenuItem;
  MemoLabel: TLabel;
  Memo1: TMemo;
  procedure Save1Click(Sender: TObject);
  procedure Exit1Click(Sender: TObject);
  procedure About1Click(Sender: TObject);
  procedure FormActivate(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action:
TCloseAction);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}
```

```
procedure TMainForm.Save1Click(Sender: TObject);
begin
  Memo1.Lines.SaveToFile('memos.txt');
end;

procedure TMainForm.Exit1Click(Sender: TObject);

begin
  Close;
end;

procedure TMainForm.About1Click(Sender: TObject);
begin
  MessageDlg('Memo Pad'#13#10'© 1995 by Tom
Swan'#13#10'Version 1.00',
  mtInformation, [mbOk], 0);
end;

procedure TMainForm.FormActivate(Sender: TObject);
begin
  if FileExists('memos.txt')
  then Memo1.Lines.LoadFromFile('memos.txt')
  else Memo1.Lines.SaveToFile('memos.txt');
end;

procedure TMainForm.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  Memo1.Lines.SaveToFile('memos.txt');

end;


end.
```

Про клавіатуру

Додатки Delphi можуть використовувати два основних методи отримання вводу від клавіатури. Перший і найлегший — це використати об'єкт компонента, такий як Edit, що автоматично відповідає на натиснення клавіш. Для більш загальної обробки клавіатури ви можете створити процедури у формі, що обробляє довільну комбінацію із трьох подій.

OnKeyDown — викликається, коли ви натискаєте довільну клавішу, включаючи функціональні чи спеціальні клавіші, такі як Shift, Alt і Ctrl.

OnKeyPress — викликається, коли ви натискаєте клавішу, що генерує символ ASCII, включаючи клавіші керування.

OnKeyUp — викликається, коли ви звільняєте довільну клавішу.

Кожна з цих подій отримує, принаймні, один параметр під назвою Key, що представляє натиснену клавішу. У подіях OnKeyDown і OnKeyUp, Key — це беззнакове (тобто лише додатнє) значення-слово, що є кодом віртуальної клавіші Windows. У події OnKeyPress Key — це символьне значення, що є символом ASCII. Усі символи ASCII мають відповідні коди віртуальних клавіш але, навпаки, — неправильно. Існує багато віртуальних клавіш, що не мають відповідників ASCII.

Windows представляє коди віртуальних клавіш рядками символів, що починаються з vk_. Наприклад, vk_alt — це код віртуальної клавіші для клавіші Alt. Список кодів віртуальних клавіш можна знайти в онлайновому довіднику Delphi.

Про порядок обробки натиснення клавіші

Багато компонент відповідають на дії, пов'язані з клавіатурою, і часто необхідно вирішити питання про те, який з компонентів форми першим прийняв певну подію клавіатури.

Windows використовує фокус введення для визначення місця, куди були передані події клавіатури. Наприклад, поточно вибраний компонент у формі має фокус введення і, таким чи-

ном, саме він отримує події від клавіатури. Якщо компонент не обробляє події — наприклад, не запрограмовано використовувати певні клавіші, такі як Esc або Ctrl — ця подія передається до власника компоненту, який переважно є формою.

Щоб послати події клавіатури до форми, перед тим, як вибраний компонент отримає ці події, встановіть властивість форми KeyPreview у значення True. Наприклад, ви можете робити це, щоб модифікувати певні клавіші або, щоб не дозволити споживачу вводити деякі символи, такі як цифри або знаки пунктуації.

Простенький приклад покаже значимість властивості KeyPreview, яку ви можете використати, щоб накласти обмеження на введення у компоненти редагування тексту. Отже, розпочнемо новий проект і виконаємо такі дії.

1. Вставте компонент Edit у форму. Вам не слід змінювати жодних його властивостей.
2. Змініть властивість KeyPreview з False у True, двічі клацнувши її значення.
3. Перейдіть на сторінку Events у вікні Object Inspector і двічі клацніть подію форми OnKeypress, щоб створити обробник. Переконайтеся, що ви це робите для форми, а не об'єкта Edit1. Введіть наступне речення між begin та end, щоб перетворити символи малих літер у великі:

   Key := Upcase(Key);
4. Натисніть F9, щоб скомпілювати та запустити програму. Якщо Delphi виведе діалог типу "file-save", збережіть модуль та проект під іменами за припущенням у тимчасовому каталозі.
5. Введіть деякий текст в Edit1. Програма автоматично перетворить символи малих літер у великі, незалежно від того, чи ви натиснули Caps Lock.
6. Вийдіть з програми, встановіть KeyPreview у False, і тоді натисніть F9, щоб скомпілювати та запустити. Програма більше не перетворюватиме символи у великі літери. Це сталося тому, бо елемент керування, що має фокус введення (відмінний від форми), отримує події клавіатури першим.

Цей метод також корисний для обмеження видів символів, які може вводити споживач. Наприклад, щоб виключити можливість введення споживачем цифр у компонент Edit, додайте на-

ступне твердження до обробника події форми OnKeypress і встановіть властивість KeyPreview в True:

```
if Key in ['0' .. '9'] then

  Key := Chr(0);
```

Це речення типу розгалуження перевіряє, чи параметр Key, переданий в обробник події, знаходиться в множині символів від '0' до '9'. Якщо так, то другий рядок речення присвоює символьне значення нуля змінній Key. Оскільки нуль, як значення символу ASCII не має жодного відповідника, це унеможливлює введення у компонент Edit.

Попереднє речення розгалуження використовує деякі, можливо, нові для вас конструкції мови програмування Pascal. Вираз '0' .. '9' називається діапазоном. Він складається із двох значень, відділених двома крапками. Pascal інтерпретує цей вираз, як еквівалентний ряду символів:

'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'

Властивості форми

Багато властивостей форми мають очевидне призначення і ми не будемо їх пояснювати тут (наприклад, властивості форми Height та Width). Далі перерахуємо та пояснимо призначення решти властивостей.

ActiveControl — встановлюється у значення властивості Name того об'єкта, для якого ми бажаємо надавати фокус введення (робити активним) відразу ж після появи вікна форми. Фокус введення не є постійним і споживач може натиснути клавішу Tab, щоб передати фокус іншим компонентам.

AutoScroll — встановлюється у True, якщо ви бажаєте, щоб смуги прокручування автоматично з'являлися у вікні при появі інформації, що виходить поза віконні рамки. Також впливає на масштабування форми. Якщо властивість встановлена у False, то і клієнтська частина вікна і все вікно будуть масштабуватися, якщо ж True, то масштабується лише клієнтська частина. Переважно у масштабованих формах вона встановлюється у False.

Cursor — описує вигляд курсора, який він має при переміщеннях вказівника мишки у клієнтській частині вікна. Вибирається одне із значень із списку, що розгортається для цієї властивості.

Enabled — переважно встановлюється у True, щоб надати можливість відповідати на події, пов'язані з мишкою, таймером та клавіатурою. Встановлення у False виключає обробку подій. Ніколи не робіть цього для форм головних вікон. Встановлюйте значення цієї властивості False лише для форм, якими керує програма (наприклад, для вікна типу splash (спалах), що з'являється при запуску додатка).

HorzScrollBar — встановлює значення для горизонтальної смуги прокручування. Смуга прокручування з'явиться тоді, лише коли "підзначення" Visible встановлено у True, а Range більше ніж ClientWidth. Двічі клацніть назву властивості (але не її значення), щоб отримати ці "підзначення".

Icon — вибирає іконку для позначення вікна форми. Якщо форма — це головне вікно програми, то властивість Icon описує системну іконку, яку виводить Windows, коли ви мінімізуєте вікно.

KeyPreview — встановлення цієї властивості у True надасть вам можливість отримувати більшість подій клавіатури перед тим, як вони потраплять до елементу керування, що має фокус введення. Якщо ж її значення False, то усі події клавіатури потраплять до вибраного елементу керування. Незважаючи на установки KeyPreview, форма ніколи не отримує подій кнопок навігації, таких як згенерованих натисненням Tab, BackTab та клавішами переміщення курсора.

Menu — описує, який компонент MainMenu використовувати для рядка головного меню форми. Переважно Delphi встановлює цю властивість у перший об'єкт форми типу MainMenu, але ви можете змінити його у інше меню.

ObjectMenuItem — використовується додатками OLE.

PixelsPerInch — визначає, як додаток створюватиме вікно форми, змасштабоване відповідно до числа пікселів на дюйм. Використовуючи цю властивість разом з Scaled, можна створювати форми, що матимуть подібний вигляд при різних розділь-

них здатностях екрану. Якщо Scaled встановлено у False, PixelsPerInch не має жодного впливу. Назва цієї властивості дещо є невлучною, оскільки її значення не дорівнює числу пікселів на дюйм. Насправді ж Delphi визначає це число на основі розміру системного шрифту і тоді використовує це значення, щоб змасштабувати вікно форми.

Position — визначає метод для обчислення розміру форми та її розміщення. Установка за припущенням poDefault виводить вікно у його розмірах, заданих під час проектування, але обчислює його розміри під час виконання (Windows визначає початковий розмір форми часу виконання); poDefaultSizeOnly встановлює розміри вікна, такі ж як під час проектування, але розміщає його під час виконання (Windows визначає, де вікно з'являється); poScreenCenter розміщує вікно у центрі екрану.

Scaled — встановіть цю властивість у True, щоб використати властивість PixelsPerInch і створити форму, що автоматично масштабуються до такого ж розміру незалежно від роздільної здатності монітора. Установка False не масштабує форми.

Tag — не має жодного попереднього призначення. Властивість Tag використовується для зберігання довільних цілочисельних значень — наприклад, номер версії, код, який програма бажає взнати під час виконання.

VertScrollBar — має таке ж призначення, що й властивість HorzScrollBar, але конфігурує "підвластивості" вертикальної смуги прокручування.

Visible — встановлення у True робить компонент видимим; False приховує його до тих пір, поки програма не викличе для об'єкта компонента метод Show. Додаток автоматично робить свої форми видимими у потрібний час. Для форм, що використовуються для підпорядкованих вікон, діалогових вікон, властивість Visible може бути встановлена у False, щоб приховати вікно до потрібного часу.

WindowMenu — у додатках з багатовіконним інтерфейсом (MDI) визначає меню, що виводить назви відкритих вікон. Вставіть цю властивість у значення довільного пункту меню (переважно це Window) у об'єкті форми MainMenu.

Події форми

Список подій форми можна переглянути у вікні Object Inspector. Деякі з подій для форми є очевидними (наприклад, OnClick і OnDblClick). Про інші події зараз піде мова.

OnActivate — викликається, коли програма активізує форму — тобто, коли вона вперше отримує фокус введення, а також, коли ви, наприклад, повертаєтеся назад у програму з іншого додатка.

OnClose — викликається, коли форма закривається. Дивіться також на подію OnDestroy.

OnCloseQuery — викликається якраз перед закриттям форми, переважно коли додаток викликає процедуру Close. Ви можете використати цю подію, щоб уникнути втрати даних, повідомивши споживача про необхідність збереження змінених файлів перед завершенням додатка, або щоб зберегти форму від закриття.

OnCreate — викликається відразу ж, коли програма створює об'єкт форми у пам'яті. Використовуйте цю подію, щоб виконати одноразову ініціалізацію.

OnDeactivate — викликається, коли споживач перемикається з додатка у інший. Дивіться також про подію OnActivate.

OnDestroy — викликається саме перед руйнуванням об'єкта форми. Використовуйте цю подію, щоб звільнити системні ресурси, або щоб виконати дії при аварійному завершенні. Якщо форма є головним вікном програми, OnDestroy — це ваша остання можливість виконати якісь дії перед завершенням програми.

OnDragDrop, OnDragOver — це частини сервісу Delphi drag-and-drop.

OnHide — викликається, щоб виконати якісь дії, коли форма є прихованою. Наприклад, обробник події OnHide може звільнити пам'ять та інші ресурси, коли форма є невидимою.

OnKeyDown — викликається, коли споживач натискає довільну клавішу, включаючи функціональні та спеціальні. Використовуйте цю подію, щоб інтерпретувати натиснення комбінацій клавіш, наприклад, з Alt, Shift, функціональними клавішами. Можливо, отримувати кілька подій OnKeyDown без отримання

пов'язаних подій OnKeyUp events. Для використання цієї події переважно встановлюють властивість KeyPreview у True.

OnKeyPress — викликається, коли споживач натискає єдину клавішу ASCII або керування, але не функціональну чи іншу спеціалізовану клавішу. Цю подію використовують для аналізу натискань на символьні клавіші. Ця подія настає після події OnKeyDown, але перед подією OnKeyUp. Для використання цієї події переважно встановлюють властивість KeyPreview у True.

OnKeyUp — викликається, коли споживач звільняє довільну клавішу, включаючи функціональну чи спеціальну. Завжди існує одна подія OnKeyUp — відповідник події OnKeyDown. event. Для використання цієї події переважно встановлюють властивість KeyPreview у True.

OnMouseDown — викликається, коли споживач натискає довільну клавішу мишки і вказівник мишки розміщується над клієнтською частиною форми.

OnMouseMove — викликається, коли споживач переміщає вказівник мишки через клієнтську частину форми. Ви також можете визначити, чи споживач натискав клавіші Shift, Alt або Ctrl переміщуючи мишку.

OnMouseUp — викликається, коли споживач звільняє кнопку мишки, що попередньо було згенеровано подією OnMouseDown event, незважаючи на розміщення вказівника мишки.

OnPaint — викликається, коли вміст форми збирається змінитися — наприклад, після того, як споживач перекривав форму іншим вікном.

OnResize — викликається після того, як вікно змінюється у розмірах. Це може настати у відповідь на переміщення кнопки розміру вікна та мінімізуючи чи максимізуючи вікно.

OnShow — викликається саме перед тим, як вікно стає видимим. Ця подія використовується для ініціалізацій перед появою вікна.

Література

1. Guide to Medical Informatics, the Internet and Telemedicine, by Enrico Coiera. August 1997, Chapman & Hall; ISBN: 0412757109.
2. Handbook of Medical Informatics, by J. Van Bemmel, Mark A. Musen (Editors). October 1997, Springer Verlag; ISBN: 3540633510.
3. Introduction to Clinical Informatics (Computers in Health Care) by Patrice Degoulet, Benjamin Phister (Translator), Marius Fieschi (Contributor). October 1996, Springer Verlag; ISBN: 0387946411.
4. Medical Informatics : Computer Applications in Health Care, by Edward H. Shortliffe, Leslie E. Perreault, editors, Gio Wiederhold, Lawrence M. Fagan, associate editors. 1999.
5. Michel A. Audette, Frank P. Ferrie, Terry M. Peters, An algorithmic overview of surface registration techniques for medical imaging, Medical Image Analysis, 4 (2000) 201-217.
6. Tom Swan, Introducing Delphi Programming.
7. Гудима А.А., Марценюк В.П. Організація вивчення інформатики у вищому медичному навчальному закладі // Медична освіта — 1999. — №1, — С. 72-75.
8. Дэвис Стефен Р. Программирование на Microsoft Visual Java++/ Пер. с англ. — М.: Издательский отдел "Русская редакция" ТОО "Channel Trading Ltd.", 1997. — 376с.: ил.
9. Керниган Б., Ритчи Д., Фьюэр А. Язык программирования Си. Задачи по языку Си: Пер.с англ — М.: Финансы и статистика, 1985.
10. Красовский Н.Н. Некоторые задачи теории устойчивости движения. — М.: Физматгиз, 1959. — 212с.
11. Марценюк В.П. Про розробку мультимедійних підручників з інформатики для вищих медичних закладів. — Сучасні проблеми підготовки фахівців у вищих медичних та фармацевтичних закладах освіти I-IV рівнів акредитації МОЗ України // Матеріали доповідей науково-методичної конференції. — Київ-Тернопіль, 29 вересня — 1 жовтня 1999 року. — С. 120-121.
12. Марчук Г.И. Математические модели в иммунологии. — М.: Наука, 1980. — 264с.
13. Минцер О.А., Молотков В.Н., Угаров Б.Н. и др. Биологическая и медицинская кибернетика. Справочник. — К.: Наукова думка, 1989. — 375 с.

14. Сердюк А.М., Торбін В.Ф., Поліщук В.М., Сайкевич А.І., Щербатий А.А. Комп'ютерні технології в медицині та Державний реєстр. — Рівне: Вертекв, 1997. — 200 с.
15. Хайрер Д., Нерсетт В., Ваннер Т. Решение обыкновенных дифференциальных уравнений. Нежесткие задачи. — М.: Мир, 1990. — 412 с.
16. Хейл Дж. Теория функционально-дифференциальных уравнений. — М.: Мир, 1984. — 421 с.
17. Чалий О.В., Дяків В.А., Хаїмзон І.І. Основи інформатики: Навчальний посібник. — К., 1993. — 143 с.

```
unit DialogButton;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Buttons, comctrls, stdctrls, dbtables, MMSystem, db, FMXUtils;

const upperbound = 600;
      Logon = 'Logon.htm';

type

  TQuestionForm = class(TForm)
    TrackBar1: TTrackBar;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    QuestionLabel: TLabel;
    CheckBox1: TCheckBox;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    procedure Button1Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TDialogButton = class(TSpeedButton)
  private
    { Private declarations }
    FDataLinkGT: TDataLink;
    FDataLinkHT: TDataLink;
    FDataLinkET: TDataLink;
    FGeneralTable: TDataSource;
    FHypothesesTable: TDataSource;
    FEvidencesTable: TDataSource;
    function GetGeneralTable: TDataSource;
    function GetHypothesesTable: TDataSource;
    function GetEvidencesTable: TDataSource;
    procedure SetGeneralTable(Value: TDataSource);
    procedure SetHypothesesTable(Value: TDataSource);
    procedure SetEvidencesTable(Value: TDataSource);
  protected
    { Protected declarations }
    property DataLinkGT: TDataLink read FDataLinkGT;
    property DataLinkHT: TDataLink read FDataLinkHT;
    property DataLinkET: TDataLink read FDataLinkET;
  public
    { Public declarations }
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
```

```
procedure Click; override;
  published
    { Published declarations }
    property GeneralTable: TDataSource read GetGeneralTable write SetGeneralTable;
    property HypothesesTable: TDataSource read GetHypothesesTable write SetHypothesesTable;
    property EvidencesTable: TDataSource read GetEvidencesTable write SetEvidencesTable;
  end;


var    response: integer;

procedure Register;


implementation

{$R  *.DFM}

procedure TQuestionForm.Button1Click(Sender: TObject);
begin
     response := TrackBar1.Position;
     Close;
end;

procedure TQuestionForm.SpeedButton1Click(Sender: TObject);
begin
     response := TrackBar1.Position;
     ModalResult := 10;
end;

procedure TQuestionForm.SpeedButton3Click(Sender: TObject);
begin
     response := TrackBar1.Position;
     ModalResult := 20;
end;

procedure TQuestionForm.FormCreate(Sender: TObject);
begin
     Width := Round(Screen.Width /1.5);
     Height := Round(Screen.Height / 1.5);
     Top := Round(Screen.Height / 7);
     Left := (Screen.Width - Width) div 2;
end;

procedure TQuestionForm.FormShow(Sender: TObject);
 var   ms : TMemoryStream;
begin
   {begin question - voice}
      {try
        ms := TMemoryStream.Create;
        (EvidencesTable.DataSet.FieldByName('Voice_sym_quest') as TBlobField).SaveToStream(ms);
        sndPlaySound(ms.Memory, SND_ASYNC or SND_MEMORY);
      except
        ShowMessage('Playing error');
      end;
      ms.Free;}
     {end question - voice}
end;
```

```
constructor TDialogButton.Create(AOwner: TComponent);
begin
    inherited Create(AOwner);

    FDataLinkGT := TDataLink.Create;
    FDataLinkHT := TDataLink.Create;
    FDataLinkET := TDataLink.Create;
end;

destructor TDialogButton.Destroy;
begin
    FDataLinkGT.Free;
    FDataLinkHT.Free;
    FDataLinkET.Free;
    inherited Destroy;

end;

function TDialogButton.GetGeneralTable: TDataSource;
begin
  Result := FDataLinkGT.DataSource;
end;

function TDialogButton.GetHypothesesTable: TDataSource;
begin
  Result := FDataLinkHT.DataSource;
end;

function TDialogButton.GetEvidencesTable: TDataSource;
begin
  Result := FDataLinkET.DataSource;
end;

procedure TDialogButton.SetGeneralTable(Value: TDataSource);
begin
 if Value = FDatalinkGT.Datasource then Exit;

  FDataLinkGT.DataSource := Value;
  if Value <> nil then Value.FreeNotification(Self);

end;

procedure TDialogButton.SetHypothesesTable(Value: TDataSource);
begin
 if Value = FDatalinkHT.Datasource then Exit;

  FDataLinkHT.DataSource := Value;
  if Value <> nil then Value.FreeNotification(Self);

end;

procedure TDialogButton.SetEvidencesTable(Value: TDataSource);
begin
 if Value = FDatalinkET.Datasource then Exit;

  FDataLinkET.DataSource := Value;
  if Value <> nil then Value.FreeNotification(Self);

end;
```

```
procedure TDialogButton.Click;
   var i, j, k,  hypotheses, evidence, bestvar, best, resp: integer;
   rulevalue, varflag: array [1..70] of real;
   p_array, mini, maxi: array [1..100] of real;
   questions, varq: array [1..upperbound] of integer;
   hypotheses_mentioned: array [1..100] of integer;
   rv, a1, a2, a3, a4, prior, p, py, pn, pe, maxofmin: real;
   add_strings: TStrings;
   ms: TMemoryStream;
   QuestionForm: TQuestionForm;

label 340;
begin
   inherited Click;
   QuestionForm := TQuestionForm.Create(Self);
   best := 1;
   add_strings := TStringList.Create;
   add_strings.SaveToFile(Logon);
   add_strings.Append('<HTML>');
   add_strings.Append('<HEAD>');
   add_strings.Append('</HEAD>');
   add_strings.Append('<BODY  background="bkgnd.jpg" TEXT="#000000"  LINK="#0000EE"
VLINK="#551A8B" ALINK="#FF0000">');
   add_strings.Append('</BODY>');
   add_strings.Append('</HTML>');
   hypotheses := HypothesesTable.DataSet.RecordCount;
   evidence :=  EvidencesTable.DataSet.RecordCount;
   for j := 1 to evidence do varflag[j] := 1;
   for j := 1 to hypotheses do  hypotheses_mentioned[j] := 1;

   {definition of appriory probabilities and rule values}

   for i:= 1 to  hypotheses
      do
        begin
            p_array[i] := HypothesesTable.DataSet.FieldValues['p'];
            HypothesesTable.DataSet.Next;
        end;
   for i:= 1 to  GeneralTable.DataSet.RecordCount
      do
        begin
            questions[round(GeneralTable.DataSet.FieldValues['id_decease'])] :=
questions[round(GeneralTable.DataSet.FieldValues['id_decease'])] + 1;
            rulevalue[round( GeneralTable.DataSet.FieldValues['id_symtom'])] := rulevalue[round(
GeneralTable.DataSet.FieldValues['id_symtom'])] + abs(GeneralTable.DataSet.FieldValues['py'] -
GeneralTable.DataSet.FieldValues['pn']);
            GeneralTable.DataSet.Next;
        end;
   for i:= 1 to  GeneralTable.DataSet.RecordCount do varq[i] := questions[i];

{determining the most appropriate sympthom and question}
 340:
   rv := 0;
   bestvar := 0;
   for j := 1 to evidence
```

```
do
        begin
            if rulevalue[j]>rv
              then
                 begin
                     bestvar := j;
                     rv := rulevalue[j];
                 end;
            rulevalue[j] := 0;
        end;
    if (bestvar = 0)
      then
        begin
            add_strings.Strings[add_strings.Count - 2] := '<BR><B><FONT SIZE=14><FONT
COLOR="#DD0000"> Симптомів більше немає </FONT></FONT></B>';
            add_strings.Strings[add_strings.Count - 1] := '</BODY>';
            add_strings.Append('</HTML>');
            PlaySound('nosymptoms.wav', 0, SND_FILENAME);
            ShowMessage('Симптомів більше немає');
            add_strings.SaveToFile(Logon);
            {DisplayForm.Display.LoadStrings(add_strings);}
            Enabled := false;
            Hint := 'Для початку нового діалога перезапустіть Експертну систему';
            Exit;
            {Application.Terminate;}
        end;
    EvidencesTable.DataSet.First;
    EvidencesTable.DataSet.Filter := 'id_symtom =' + IntToStr(bestvar);
    EvidencesTable.DataSet.FindNext;
    varflag[bestvar] := 0;
    QuestionForm.QuestionLabel.Caption := EvidencesTable.DataSet.FieldValues['symptom_question'];

    resp := QuestionForm.ShowModal;

    if  resp = 10
      then
        begin
            add_strings.SaveToFile(Logon);
            ExecuteFile(Logon,'',ExtractFilePath(Application.ExeName),1);
        end
      else if resp = 20 then
         begin
             HypothesesTable.DataSet.First;
             HypothesesTable.DataSet.Filter := 'id_decease =' + IntToStr(best);
             HypothesesTable.DataSet.FindNext;
              PlaySound('mostprobable.wav', 0, SND_FILENAME);
               {begin question - voice}
               try
                  ms := TMemoryStream.Create;
                  (HypothesesTable.DataSet.FieldByName('Voice_decese') as
TBlobField).SaveToStream(ms);
                  sndPlaySound(ms.Memory, SND_SYNC or SND_MEMORY);
               except
                  ShowMessage('Playing error');
               end;
               ms.Free;
               {end question - voice}
             ShowMessage('Найімовірніший наслідок - ' +
HypothesesTable.DataSet.FieldValues['decease'] + ' з імовірністю ' + FloatToStr(p_array[best]));
```

```
end;
    if QuestionForm.CheckBox1.Checked
      then
        begin
          add_strings.SaveToFile(Logon);
          Enabled := false;
          Hint := 'Для початку нового діалога перезапустіть Експертну систему';
          Exit;
        end;
    add_strings.Strings[add_strings.Count - 2] := '<BR><B><FONT SIZE=+2><FONT
COLOR="#DD0000">' + EvidencesTable.DataSet.FieldValues['symptom_question'];
    add_strings.Strings[add_strings.Count - 1] := ' <FONT COLOR="#0000FF">' + IntToStr(response)
+ '</FONT></FONT></FONT></B>';
    add_strings.Append('</BODY>');
    add_strings.Append('</HTML>');

    {analysis of response of customer and correction of probabilities}

    {correction of probabilities based on response}

    for i := 1 to hypotheses do
      begin
        GeneralTable.DataSet.First;
        GeneralTable.DataSet.Filter := 'id_decease =' + IntToStr(i);
        for k := 1 to varq[i] do
          begin
            GeneralTable.DataSet.FindNext;
            if (GeneralTable.DataSet.FieldValues['id_symtom'] <> bestvar) or (questions[i] = 0)
then Continue;
            questions[i] := questions[i] - 1;
            p:=p_array[i];
            py:= GeneralTable.DataSet.FieldValues['py'];
            pn:= GeneralTable.DataSet.FieldValues['pn'];
            pe:=p*py + (1-p)*pn;
            if response > 0
              then p_array[i]:=p*(1+(py/pe-1)*response/5)
              else p_array[i]:=p*(1+(py-(1-py)*pe/(1-pe))*response/5);
            if p_array[i]=round(p_array[i]) then questions[i]:=0;
          end;
      end;

      {determining of new values of rules and minimal and maximal values of the probabilities}

      maxofmin:=0;
      best:=0;
      for i:=1 to hypotheses
        do
          begin
            GeneralTable.DataSet.First;
            GeneralTable.DataSet.Filter := 'id_decease =' + IntToStr(i);
            p:=p_array[i];
            a1:=1;
            a2:=1;
            a3:=1;
            a4:=1;
            HypothesesTable.DataSet.First;
            HypothesesTable.DataSet.Filter := 'id_decease =' + IntToStr(i);
            HypothesesTable.DataSet.FindNext;
            prior := HypothesesTable.DataSet.FieldValues['p'];
```

```
            for k:=1 to varq[i] do
              begin
                GeneralTable.DataSet.FindNext;
                {**********}
                j:=GeneralTable.DataSet.FieldValues['id_symtom'];
                {**********}
                py:= GeneralTable.DataSet.FieldValues['py'];
                pn:= GeneralTable.DataSet.FieldValues['pn'];
                if varflag[j]*questions[i]=0 then Continue;
                if (pn > py)
                  then
                    begin
                      py:=1-py;
                      pn:=1-pn;
                    end;
                rulevalue[j]:=rulevalue[j]+abs(py-pn);
                a1:=a1*py;
                a2:=a2*pn;
                a3:=a3*(1-py);
                a4:=a4*(1-pn);
              end;
            maxi[i]:=p*a1/(p*a1+(1-p)*a2);
            mini[i]:=p*a3/(p*a3+(1-p)*a4);
            if maxi[i]< prior
              then
                begin
                  questions[i]:=0;
                  if hypotheses_mentioned[i] = 1
                    then
                      begin
                        add_strings.Strings[add_strings.Count - 2] := '<BR> <FONT
COLOR="#00FF00"> Можна виключити  ' + HypothesesTable.DataSet.FieldValues['decease'] +
'</FONT>';
                        add_strings.Strings[add_strings.Count - 1] := '</BODY>';
                        add_strings.Append('</HTML>');
                        PlaySound('canexclude.wav', 0, SND_FILENAME);
                        {begin question - voice}
                        try
                          ms := TMemoryStream.Create;
                          (HypothesesTable.DataSet.FieldByName('Voice_decese') as
TBlobField).SaveToStream(ms);
                          sndPlaySound(ms.Memory, SND_SYNC or SND_MEMORY);
                        except
                          ShowMessage('Playing error');
                        end;
                        ms.Free;
                        {end question - voice}
                      end;
                  hypotheses_mentioned[i] := 0;
                end;
              if mini[i]>maxofmin
                then
                  begin
                    best := i;
                    maxofmin := mini[i];
                  end;
          end;

      { search of the most probabale hypothes }
```

```
for i:=1 to hypotheses
        do if (mini[best] <= maxi[i]) and (i <> best) then maxofmin := 0;
      if maxofmin = 0 then goto 340;
      HypothesesTable.DataSet.First;
      HypothesesTable.DataSet.Filter := 'id_decease =' + IntToStr(best);
      HypothesesTable.DataSet.FindNext;
      PlaySound('mostprobable.wav',0, SND_FILENAME);
      {begin question - voice}
      try
        ms := TMemoryStream.Create;
        (HypothesesTable.DataSet.FieldByName('Voice_decese') as TBlobField).SaveToStream(ms);
        sndPlaySound(ms.Memory, SND_SYNC or SND_MEMORY);
      except
        ShowMessage('Playing error');
      end;
      ms.Free;
      {end question - voice}
      ShowMessage('Найімовірніший наслідок - ' + HypothesesTable.DataSet.FieldValues['decease']
+ ' з імовірністю ' + FloatToStr(p_array[best]));
      add_strings.Strings[add_strings.Count - 2] := '<BR><B><FONT SIZE=14><FONT
COLOR="#DD0000"> Найімовірніший наслідок - ' +
HypothesesTable.DataSet.FieldValues['decease'] + ' з імовірністю ' + FloatToStr(p_array[best]) +
'</FONT></FONT></B>';
      add_strings.Strings[add_strings.Count - 1] := '</BODY>';
      add_strings.Append('</HTML>');
      add_strings.SaveToFile(Logon);
      Enabled := false;
      Hint := 'Для початку нового діалога перезапустіть Експертну систему';
end;

procedure Register;
begin
  RegisterComponents('MedInfTraining', [TDialogButton]);
end;

end.
```

```
unit SurgeryOperationsView;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs,
  ComCtrls, inifiles, FMXUtils;

type
  TSurgeryOperationsView = class(TTreeView)
  private
    { Private declarations }
    ini: TIniFile;
    strs, first_substrs, second_substrs, third_substrs: TStrings;
  protected
    { Protected declarations }
  public
    { Public declarations }
    procedure DblClick; override;
    procedure ShowItems;
  published
    { Published declarations }
  end;

procedure Register;

implementation

procedure CutStr (s: string; var first_substr: string; var
second_substr: string; var third_substr: string);
         var add_str: string;
begin
     {cutting string}
     first_substr:=s;
     Delete(first_substr, 1, Pos('=', first_substr));
     add_str:=first_substr;
     Delete(first_substr, Pos('|', first_substr),
length(first_substr));
     second_substr:=add_str;
     Delete(second_substr, 1, Pos('|', second_substr));
     add_str:= second_substr;
     Delete(second_substr, Pos('|', second_substr),
length(second_substr));
     third_substr:=add_str;
     Delete(third_substr, 1, Pos('|', third_substr));
     {add_str:= third_substr;
     Delete(third_substr, Pos('|', third_substr),
length(third_substr)); and so on}
end;

procedure CutTStrings(strs: TStrings; var first_substrs: TStrings;
var second_substrs: TStrings; var third_substrs: TStrings);
var i: integer;
    first_substr, second_substr, third_substr: string;
begin
     {cutting TStrings}
      first_substrs.Assign(strs);
      second_substrs.Assign(strs);
      third_substrs.Assign(strs);
      {*****************}
      for i:=0 to strs.Count -1
          do
            begin
                 first_substr:=first_substrs.Strings[i];
                 second_substr:=second_substrs.Strings[i];
```

```
                third_substr:=third_substrs.Strings[i];
                CutStr(strs.Strings[i], first_substr, second_substr,
third_substr);
                first_substrs.Strings[i]:= first_substr;
                second_substrs.Strings[i]:= second_substr;
                third_substrs.Strings[i]:= third_substr;
           end;
end;


procedure Register;
begin
  RegisterComponents('MedInfTraining', [TSurgeryOperationsView]);
end;

procedure TSurgeryOperationsView.DblClick;
begin
inherited DblClick;
if  Selected.ImageIndex = 2 then
     begin
          ini.ReadSectionValues(Selected.Parent.Text, strs);
          CutTStrings(strs, first_substrs, second_substrs,
third_substrs);
          ExecuteFile(third_substrs.Strings[Selected.AbsoluteIndex -
Selected.Parent.AbsoluteIndex - 1],'',
ExtractFilePath(Application.ExeName)+'Video' , SW_SHOW);
     end;
end;

procedure TSurgeryOperationsView.ShowItems;
var
    CurNode, SubNode: TTreeNode;
    numsec, counter_items: integer;
    sections: TStrings;
begin
   ShowHint := false;
   Parent.Width := Screen.Width;
   Parent.Height := Screen.Height;
   Width := (Parent.Width * 9) div 10;
   Height := (Parent.Height * 9) div 10;
   Left := (Parent.Width - Width) div 2;
   Top := (Parent.Height - Height) div 2;
   sections:= TStringList.Create;
   strs := TStringList.Create;
   first_substrs := TStringList.Create;
   second_substrs := TStringList.Create;
   third_substrs := TStringList.Create;
   ini:=
TIniFile.Create(ExtractFilePath(Application.ExeName)+'\Video\Video.in
i');
   ini.ReadSections(sections);
   Items.AddFirst(nil,'Îïåðàö³¿');
   for numsec:=0 to sections.Count -1  do
       begin
            CurNode := Items.AddChild(Items[0],
sections.Strings[numsec]);
            CurNode.ImageIndex := 1;
            CurNode.SelectedIndex := 1;
            {***************************************}
            first_substrs.Clear;
            second_substrs.Clear;
            third_substrs.Clear;
            strs.Clear;
            {***************************************}
      ini.ReadSectionValues(sections.Strings[numsec], strs );
```

```
CutTStrings(strs, first_substrs, second_substrs, third_substrs);
             for counter_items:=0 to strs.Count - 1 do
                  begin
                       SubNode := Items.AddChild(CurNode,
first_substrs.Strings[counter_items] + second_substrs.Strings[counter_items]);
                       SubNode.ImageIndex := 2;
                       SubNode.SelectedIndex := 3;
                  end;
       end;

end;


end.
```

```
unit MedicalSlidesViewer;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs,
  ComCtrls, jpeg, inifiles, extctrls, stdctrls;

type
  TMedicalSlidesViewer = class(TTreeView)
   private
    { Private declarations }
    ini: TIniFile;
    strs, first_substrs, second_substrs, third_substrs: TStrings;
    FDisplay: TImage;
    FExplanation: TLabel;
    procedure SetDisplay (Value: TImage);
    procedure SetExplanation (Value: TLabel);
   protected
    { Protected declarations }
   public
    { Public declarations }
     procedure DblClick; override;
     procedure ShowItems;
     constructor Create(AOwner: TComponent); override;
   published
    { Published declarations }
    property  Display: TImage read FDisplay write SetDisplay;
    property Explanation: TLabel read FExplanation write
SetExplanation;
  end;

  procedure Register;

implementation



procedure TMedicalSlidesViewer.SetDisplay(Value: TImage);
begin
 if (Value <> nil) then FDisplay := Value;
end;

procedure TMedicalSlidesViewer.SetExplanation(Value: TLabel);
begin
 if (Value <> nil) then FExplanation := Value;
end;

procedure CutStr (s: string; var first_substr: string; var
second_substr: string; var third_substr: string);
         var add_str: string;
begin
     {cutting string}
     first_substr:=s;
     Delete(first_substr, 1, Pos('=', first_substr));
     add_str:=first_substr;
     Delete(first_substr, Pos('|', first_substr),
length(first_substr));
     second_substr:=add_str;
     Delete(second_substr, 1, Pos('|', second_substr));
     add_str:= second_substr;
     Delete(second_substr, Pos('|', second_substr),
length(second_substr));
     third_substr:=add_str;
     Delete(third_substr, 1, Pos('|', third_substr));
     {add_str:= third_substr;
```

```
Delete(third_substr, Pos('|', third_substr), length(third_substr));
and so on}
end;

procedure CutTStrings(strs: TStrings; var first_substrs: TStrings;
var second_substrs: TStrings; var third_substrs: TStrings);
var i: integer;
    first_substr, second_substr, third_substr: string;
begin
     {cutting TStrings}
      first_substrs.Assign(strs);
      second_substrs.Assign(strs);
      third_substrs.Assign(strs);
      {*****************}
      for i:=0 to strs.Count -1
          do
            begin
                 first_substr:=first_substrs.Strings[i];
                 second_substr:=second_substrs.Strings[i];
                 third_substr:=third_substrs.Strings[i];
                 CutStr(strs.Strings[i], first_substr, second_substr,
third_substr);
                 first_substrs.Strings[i]:= first_substr;
                 second_substrs.Strings[i]:= second_substr;
                 third_substrs.Strings[i]:= third_substr;
            end;
end;


procedure Register;
begin
  RegisterComponents('MedInfTraining', [TMedicalSlidesViewer]);
end;

procedure TMedicalSlidesViewer.DblClick;
var peg: TJPEGImage;
begin
inherited DblClick;
if  Selected.ImageIndex = 2 then
     begin
          ini.ReadSectionValues(Selected.Parent.Text, strs);
          CutTStrings(strs, first_substrs, second_substrs,
third_substrs);

          {ExecuteFile(third_substrs.Strings[Selected.AbsoluteIndex -
Selected.Parent.AbsoluteIndex - 1],'',
ExtractFilePath(Application.ExeName)+'Video' , SW_SHOW);}


{*********************************************************************
***********************}
       peg := TJPEGImage.Create;

peg.LoadFromFile(ExtractFilePath(Application.ExeName)+'\Slides\'+thir
d_substrs.Strings[Selected.AbsoluteIndex -
Selected.Parent.AbsoluteIndex - 1]);
       Display.Picture.Bitmap.Assign(peg);
       {if (Display.Picture.Width >= Parent.Width - 40) or
(Display.Picture.Height >= Parent.Height - 40)
          then
              if Display.Picture.Width >=  Display.Picture.Height
                 then
                     begin
                                    Display.Height :=
Round(Display.Picture.Bitmap.Height * ((Parent.Width -
40)/Display.Picture.Bitmap.Width));
```

```
Display.Width := Parent.Width -40;
                    end
                else
                    begin
                        Display.Width :=
Round(Display.Picture.Bitmap.Width * ((Parent.Height -
40)/Display.Picture.Bitmap.Height));
                        Display.Height := Parent.Height -40;
                    end;
       Display.Left := Round((Parent.Width - Display.Width)/2);
       Display.Top := Round((Parent.Height - Display.Height)/2);}

       Explanation.Caption :=
second_substrs.Strings[Selected.AbsoluteIndex -
Selected.Parent.AbsoluteIndex - 1];
     end;
end;

procedure TMedicalSlidesViewer.ShowItems;
var
    CurNode, SubNode: TTreeNode;
    numsec, counter_items: integer;
    sections: TStrings;
begin
   ShowHint := false;
   Parent.Width := Screen.Width;
   Parent.Height := Screen.Height;
   {Width := (Parent.Width * 9) div 10;
   Height := (Parent.Height * 9) div 10;
   Left := (Parent.Width - Width) div 2;
   Top := (Parent.Height - Height) div 2;}
   sections:= TStringList.Create;
   strs := TStringList.Create;
   first_substrs := TStringList.Create;
   second_substrs := TStringList.Create;
   third_substrs := TStringList.Create;
   ini:=
TIniFile.Create(ExtractFilePath(Application.ExeName)+'\Slides\Slides.
ini');
   ini.ReadSections(sections);
   Items.AddFirst(nil,'Слайди');
   for numsec:=0 to sections.Count -1  do
       begin
            CurNode := Items.AddChild(Items[0],
sections.Strings[numsec]);
            CurNode.ImageIndex := 1;
            CurNode.SelectedIndex := 1;
            {***************************************}
            first_substrs.Clear;
            second_substrs.Clear;
            third_substrs.Clear;
            strs.Clear;
            {***************************************}
            ini.ReadSectionValues(sections.Strings[numsec], strs );
             CutTStrings(strs, first_substrs, second_substrs,
third_substrs);
             for counter_items:=0 to strs.Count - 1 do
                 begin
                     SubNode := Items.AddChild(CurNode,
first_substrs.Strings[counter_items] +
second_substrs.Strings[counter_items]);
                     SubNode.ImageIndex := 2;
                     SubNode.SelectedIndex := 3;
                 end;
       end;

end;

constructor TMedicalSlidesViewer.Create(AOwner: TComponent);
```

```
begin
     inherited Create(AOwner);

     FDisplay := TImage.Create(AOwner);
     FExplanation := TLabel.Create(AOwner);
end;

end.
```

```
//*****************************************************************
***********
// DelaySystemSolution.java:  Applet
//
//*****************************************************************
***********
import java.applet.*;
import java.awt.*;

//=================================================================
==========
// Main Class for applet DelaySystemSolution
//
//=================================================================
==========
public class DelaySystemSolution
{
      // Parameters of the system
    double m_dBeta;
      double m_dGamma;
      double m_dAlpha;
      double m_dMu_c;
      double m_dRo;
      double m_dMu_f;
      double m_dEta;
      double m_dSigma;
      double m_dMu_m;
      double m_dTau;


      // SOLUTION (initial value and solution at xend) ///////////
      double[] y;
      ////////////////////////////////////////////////////////////



      //  POSITS ////////////////////////////////////////////////

      // Meaning of these variables:
      // ifirst   lowest step number still in memory coef;
      // last           address of last data written by store on
common block coef
      //                Must be set to 0 in the calling program
      //                before the first call.
      // x0       initial point, must be set  in the calling program
      //                before the first call.
      // xlast    = x + h of last written step;
      // ipos           position of last successful search in
function ylag;
      // disc           logical variable, necessary for the
distinction
      //                of k7 and k1 of the following step in the
case
      //                when y(x0) is different from phi(x0).

      int ifirst;
      int last;
      double x0;
      double xlast;
      int ipos;
      boolean disc;

      ////////////////////////////////////////////////
      // STAT contains statistical information:
      ////////////////////////////////////////////////
      //    nfcn  number of function evaluations
      // nstep    number of computed steps
      // naccpt   number of accepted steps
      // nrejct   number of rejected steps

      int nfcn;
```

```
int nstep;
      int naccpt;
      int nrejct;

      ////////////////////////////////////////////////
      // UROUND   smallest number satisfying 1. + uround > 1.
      //               (to be adapted by user)
      ////////////////////////////////////////////////


      ////////////////////////////////////////////////
      // COEF contains coefficients for global solution
      ////////////////////////////////////////////////


      final int nn = 4;
      final int mxst = 800;
      private double xstor[];
      private double ystor[][];
      private double c1[][];
      private double c2[][];
      private double c3[][];
      private double c4[][];

      final double uround = 5.e-8;

      ////////////////////////////////////////////////
      // SIZE OF SYSTEM  ////////////////////////////
      ////////////////////////////////////////////////
      int n;
      ////////////////////////////////////////////////



      ////////////////////////////////////////////////////////////////
////
      // This method return value of system right-side f(x,y(x),y(x-
tau))
      // for given (x,y)
      ////////////////////////////////////////////////
      ////////////////////////////////////////////////////////////////
////
      synchronized public double[] fcn(double x, double y[])
      {
            double[] dRight_side = new double[Math.max(n, nn) + 1];
            // ìîäåëèðîâàíèå âîñïàëèòåëüíîãî ïðîöåññà èíôåêöèîííîé
ïðèðîäû
            dRight_side[1] = (m_dBeta - m_dGamma * y[3])*y[1];
            dRight_side[2] = xi(y[4])     * m_dAlpha * ylag(1, x -
m_dTau) * ylag(3, x - m_dTau) - m_dMu_c * (y[2] - 1.);
            dRight_side[3] = m_dRo * y[2] - (m_dMu_f + m_dEta *
m_dGamma * y[1]) * y[3];
            dRight_side[4] = m_dSigma * y[1] - m_dMu_m * y[4];
            return dRight_side;
      }

      public double phi(int i, double x)
      {
            switch (i)
            {
                  case 1:
                        return Math.max(0, x + 1.e-6);
                  case 2:
                        return 1.;
                  case 3:
                        return 1.;
                  default:
                        return 0.;
```

```
}
      }

      double xi(double m)
      {
            if(m <= .1)
            {
                  return 1.;
            }
            if((m >= .1) & (m <= 1))
            {
                  return (1 - m) / (10./9.);
            }
            else
            {
                  return 0.;
            }
      }


      synchronized public void retard(int n, double x, double y[],
double xend, double eps, double hmax, double h)
      {


            // Numerical solution of a system of first order
            // retarded differential equations y'=f(x,y(x),y(x-
tau),...).
            // This is based on an embedded Runge-Kutta method of
order (4)5
            // due to Dormand & Prince (with stepsize control).
            // C. F. sections II.5 and II.15
            //
            //
            //
            // INPUT PARAMETERS
            ////////////////////////////////////////
            // n              dimension of the system (n <= 51)
            // fcn                  name of void function computing
the
            //                      first derivative f(x,y)
            // x              initial x-value
            // xend                 final x-value  (xend > x)
            // double[] y     initial values for y
            // eps                  local tolerance
            // hmax                 maximal step size
            // h              initial step size guess
            //
            //
            // OUTPUT PARAMETERS
            ////////////////////////////////////////////
            // double y             solution at xend
            //
            ////////////////////////////////////////////

            double[] k1 = new double[Math.max(n, nn) + 1];
            double[] k2 = new double[Math.max(n, nn) + 1];
            double[] k3 = new double[Math.max(n, nn) + 1];
            double[] k4 = new double[Math.max(n, nn) + 1];
            double[] k5 = new double[Math.max(n, nn) + 1];
            double[] k6 = new double[Math.max(n, nn) + 1];
            double[] k7 = new double[Math.max(n, nn) + 1];
            double[] y1 = new double[Math.max(n, nn) + 1];
            boolean reject;
            double xph;
            double err;
```

```
double demon;
            double fac;
            double hnew;
            int iadr;

            int nmax = 3000; // maximal number of steps

            // initial preparations
            ////////////////////////////////////////

            hmax = Math.abs(hmax);
            h = Math.min(Math.max(1.e-4, Math.abs(h)), hmax);
            h = h / Math.abs(h); //signum of h
            eps = Math.max(eps, 7. * uround);
            reject = false;
            naccpt = 0;
            nrejct = 0;
            nfcn = 1;
            nstep = 0;
            disc = true;
            k1 = fcn(x, y);

            if (! disc)
            {
                  k1 = fcn(x, y);
            }

            // basic integration step
            ////////////////////////////////////////////
            // NewStepLabel:
            disc = true;
            while (!((nstep > nmax) | (x + .1 * h == x)))
            {
                  if ((x - xend) + uround > 0.)
                  {
                        return;
                  }
                  if ((x + h - xend) > 0.)
                  {
                        h = xend - x;
                  }
                  nstep = nstep + 1;

                  // the 7 Runge - Kutta stages
                  for (int i = 1; i <= n; i++)
                  {
                        y1[i] = y[i] + h * .2 * k1[i];
                  }
                  k2 = fcn(x + .2 * h, y1);
                  for (int i = 1; i <= n; i++)
                  {
                        y1[i] = y[i] + h * ((3./40.) * k1[i] +
(9./40.) * k2[i]);
                  }
                  k3 = fcn(x + .3 * h, y1);
                  for (int i = 1; i <= n; i++)
                  {
                        y1[i] = y[i] + h * ((44./45.) * k1[i] -
(56./15.) * k2[i] + (32./9.) * k3[i]);
                  }
                  k4 = fcn(x + .8 * h, y1);
                  for (int i = 1; i <= n; i++)
                  {
```

```
y1[i] = y[i] + h * ((19372./6561.) * k1[i] - (25360./2187.) * k2[i] +
(64448./6561.) * k3[i] - (212./729.) * k4[i]);
                  }
                  k5 = fcn(x + (8./9.) * h, y1);
                  for (int i = 1; i <= n; i++)
                  {
                        y1[i] = y[i] + h * ((9017./3168.) * k1[i] -
(355./33.) * k2[i] + (46732./5247.)* k3[i] + (49./176.) * k4[i] -
(5103./18656.) * k5[i]);
                  }
                  xph = x + h;
                  k6 = fcn(xph, y1);
                  for (int i = 1; i <= n; i++)
                  {
                        y1[i] = y[i] + h * ((35./384.) * k1[i] +
(500./1113.) * k3[i] + (125./192.) * k4[i] - (2187./6784.) * k5[i] +
(11./84.) * k6[i]);
                  }
                  disc = true;
                  k7 = fcn(xph, y1);
                  for (int i = 1; i <= n; i++)
                  {
                        k2[i] = ((71./57600.) * k1[i] - (71./16695.)
* k3[i] + (71./1920.) * k4[i] - (17253./339200.) * k5[i] + (22./525.)
* k6[i] - (1./40.) * k7[i]) * h;
                  }
                  nfcn = nfcn + 6;

                  // error estimation

                  err = 0.;
                  for (int i = 1; i <= n; i++)
                  {
                        demon = Math.max(Math.max(1.e-5,
Math.abs(y1[i])), Math.max(Math.abs(y[i]), 2. * uround / eps));
                        err = err + Math.pow(k2[i] / demon, 2);
                  }
                  err = Math.sqrt(err / (double)n);

                  // computation of hnew
                  // We require  .2 <= hnew / h <= 10.

                  fac = Math.max(.1, Math.min(5., Math.pow(err / eps,
1./5.) / .9));
                  hnew = h / fac;
                  if (err <= eps)
                  {
                        // step is accepted
                        naccpt = naccpt + 1;

      //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                        store(x, xph, k1, k3, k4, k5,
k6);////////////////////

      //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                        //g.setColor(Color.white);
                        //g.fillRect(90, 20, 30, 20);
                        //g.setColor(Color.black);
                        //g.drawString(Double.toString(fac), 90, 40);
                        for (int i = 1; i <= n; i++)
                        {
                              k1[i] = k7[i];
                              y[i] = y1[i];
                        }
                        // recompute k1 in the case of discontinuous
initial phase
```

```
if (! disc)
                        {
                              k1 = fcn(xph, y);
                        }
                        x = xph;
                        if (Math.abs(hnew) > hmax)
                        {
                              hnew = nmax;
                        }
                        if (reject)
                        {
                              hnew = Math.min(Math.abs(hnew),
Math.abs(h));
                        }
                        reject = false;
                  }
                  else
                  {
                        // step is rejected
                        reject = true;
                        if (naccpt >= 1)
                        {
                              nrejct = nrejct + 1;
                        }
                  } // end if

                  h = hnew;


            //    break NewStepLabel;
            disc = true;
            }
            // fail exit
            //g.drawString("exit of retard at x = " +
Double.toString(x), 10, 20);
            // return;
      }


      synchronized private void store(double x, double xph, double
fg1[], double fg3[], double fg4[], double fg5[], double fg6[])
      {
            int iadr;


            last = last + 1;
            ifirst = Math.max(1, last - mxst + 1);
            iadr = ((last - 1) % mxst) + 1;
            xlast = xph;

            xstor[iadr] = x;

            for (int i = 1; i <= nn; i++)
            {
                  ystor[i][iadr] = y[i];
                  c1[i][iadr] = fg1[i];
                  c2[i][iadr] = -(1337./480.) * fg1[i] +
(105400./27825.) * fg3[i] - (135./80.) * fg4[i] - (54675./212000.) *
fg5[i] + (66./70.) * fg6[i];
                  c3[i][iadr] = (1039./360.) * fg1[i] -
(468200./83475.) * fg3[i] + (9./2.) * fg4[i] + (400950./318000.) *
fg5[i] - (638./210.) * fg6[i];
                  c4[i][iadr] = -(1163./1152.) * fg1[i] +
(37900./16695.) * fg3[i] - (415./192.) * fg4[i] - (674325./508800.) *
fg5[i] + (374./168.) * fg6[i];
```

```
}

            return;
      }

      synchronized double ylag(int i, double x)
      {
            int iadr;
            double h;
            double s;

            // initial phase
            if (disc)
            {
                  if (Math.abs(x - x0) <= (3. * uround *
Math.abs(x)))
                  {
                        disc = false;
                  }
                  if (x <= x0)
                  {

                        return phi(i, x);
                  }
            }




            // compute the position of x
            if (x < xstor[ifirst])
            {
                  //g.drawString("Memory full, mxst = " +
Integer.parseInt(mxst), 10, 30);
                  // stop
            }

            ipos = Math.max(ifirst, Math.min(last, ipos));
            iadr = ((ipos - 1) % mxst) + 1;
            while ((x < xstor[iadr]) & (ipos > ifirst))
            {
                  ipos = ipos - 1;
                  iadr = ((ipos - 1) % mxst) + 1;
            }
            iadr = (ipos % mxst) + 1;
Cycle:
            while(ipos < last)
            {
                  if (x > xstor[iadr])
                  {
                        ipos = ipos + 1;
                        iadr = (ipos % mxst) + 1;
                        continue;
                  }
                  else
                  {
                        break Cycle;
                  }
            }

            // Compute the desired approximation

            iadr = ((ipos - 1) % mxst) + 1;
            if (ipos == last)
```

```
{
                h = xlast - xstor[iadr];
            }
            else
            {
                h = xstor[(ipos % mxst) + 1] - xstor[iadr];
            }

            s = (x - xstor[iadr]) / h;


            return ystor[i][ipos] + h * s * (c1[i][ipos] + s *
(c2[i][ipos] + s * (c3[i][ipos] + s * c4[i][ipos])));

      }



      // DelaySystemSolution Class Constructor
      //-------------------------------------------------------------
-------------
      public DelaySystemSolution(int arg_n, double arg_x0, double
arg_x1, double dDelay, double hmax, double dBeta, double dGamma,
double dAlpha,    double dMu_c, double dRo, double dMu_f, double
dEta, double dSigma, double dMu_m, double dTau)
      {
            // GENERAL INITALIZATION ////////////////////


            xstor = new double[mxst + 1];
            ystor = new double[nn + 1][mxst + 1];
            c1 = new double[nn + 1][mxst + 1];
            c2 = new double[nn + 1][mxst + 1];
            c3 = new double[nn + 1][mxst + 1];
            c4 = new double[nn + 1][mxst + 1];

            n = arg_n;
            x0 = arg_x0;
            double x1 = arg_x1;
            ////////////////////////////////
            m_dBeta = dBeta;
            m_dGamma = dGamma;
            m_dAlpha = dAlpha;
            m_dMu_c = dMu_c;
            m_dRo = dRo;
            m_dMu_f = dMu_f;
            m_dEta = dEta;
            m_dSigma = dSigma;
            m_dMu_m = dMu_m;
            m_dTau = dTau;
            ////////////////////////////////
            last = 0;
            double x;
            double  h = 0.5;
            y = new double[Math.max(n, nn) + 1];
            for(int i = 1; i <= Math.max(n, nn); i++)
            {
                  if(i <= n)
                  {
                        y[i] = phi(i, x0);
                        //y[i] = .1;
                  }
                  else
                  {
```

```
y[i] = 0;
                  }
            }
            for (double i = (int)x0; i <= (int)x1; i += dDelay)
            {
                  x = i - dDelay;
                  double xend = (double)(i);
                  double eps = 1.e-6;
                  retard(n, x, y, xend, eps, hmax, h);

            }
      }


      public DelaySystemSolution(int arg_n, double dDelay, double
hmax)
      {
            // GENERAL INITALIZATION ////////////////////

            xstor = new double[mxst + 1];
            ystor = new double[nn + 1][mxst + 1];
            c1 = new double[nn + 1][mxst + 1];
            c2 = new double[nn + 1][mxst + 1];
            c3 = new double[nn + 1][mxst + 1];
            c4 = new double[nn + 1][mxst + 1];

            n = arg_n;
            x0 = 0;
            double x1 = 1;
            last = 0;
            double x;
            double  h = 0.5;
            y = new double[Math.max(n, nn) + 1];
            for(int i = 1; i <= Math.max(n, nn); i++)
            {
                  if(i <= n)
                  {
                        y[i] = phi(i, x0);
                        //y[i] = .1;
                  }
                  else
                  {
                        y[i] = 0;
                  }
            }
            for (double i = (int)x0; i <= (int)x1; i += dDelay)
            {
                  x = i - dDelay;
                  double xend = (double)(i);
                  double eps = 1.e-6;
                  retard(n, x, y, xend, eps, hmax, h);

            }
      }

}
```

```
/*
 *
 * GraphConstruction
 *
 */

import java.applet.*;
import java.awt.*;

//////////////////////////////////////////////////////
// клас-аплет - побудова графіків математичних ///////////
// функцій. Працює дуже просто. Вимагається лише //////////
// означити спосіб обчислення функції у  /////////////////
// function f(x) та вказати ліву та праву межі ////////////
// області визначення функції у файлі .html в ////////////
// якості параметрів x0 та x1; параметр Scale - ///////////
// щільність шкали числових позначок на осях. ////////////
// За допомогою перетягування мишкою можна вказати   ///////
// нові межі області визначення функції.  /////////////////
// Натискуючи кнопку Back можна повернутися    ///////////////
// у попередні межі області визначення. ///////////////////
//////////////////////////////////////////////////////

public class GraphConstruction extends java.applet.Applet implements
Runnable
{
      DelaySystemSolution SampleSystemSolution;
      Thread m_Graphics = null;
      double m_fps = .1;

      private int m_nAreaNumber;
      private double m_dHmax;
      private double m_dDelay;
      private boolean m_bLegend;
      private boolean m_bPunctureLine;
      private double m_x0;    // ліва межа області визначення
      private double m_x1;    // права межа області визначення
     private double m_dBeta;      ////////////////////////////
      private double m_dGamma;
      private double m_dAlpha;     //  ПАРАМЕТРИ
      private double m_dMu_c;
      private double m_dRo;
      private double m_dMu_f;
      private double m_dEta;        //       МОДЕЛІ
      private double m_dSigma;
      private double m_dMu_m;
      private double m_dTau;         ////////////////////////////
      private LinkedList m_llCursorLocations = new LinkedList(); //
зв'язаний список меж областей визначення
      private FontMetrics m_fm;           //метрики шрифту
      private Dimension m_dimCursorLocBegin;  // координати курсора з
початку перетягування
      private Dimension m_dimCursorLocEnd;  // координати курсора
      в кінці перетягування
      private boolean m_bDrag; //   якщо TRUE - то в даний момент -
перетягування
      private boolean m_bBeginDrag; // якщо TRUE - то початок
перетягування
      private int m_nOffset_x;
      private int m_nOffset_y;
      private double m_dStep; // m_dStep - крок сітки  по x
      private int m_nScale; // щільність шкали
      private Button m_button = new Button("Back");
      private Button m_buttonFewGraphs = new Button("Mark few
graphs");
      private Button m_buttonWholeScreen = new Button("Whole
screen");
```

```
private Dimension m_dimDragEvtLoc;
      private int m_nGraphWidth;
      private int m_nGraphHeight;
      private int m_nGraphCount;
      private boolean m_bMarkGraphs;
      private boolean m_bWholeScreenMode;
      private boolean m_bWholeScreenChosen = false;
      private FunctionList m_nFunctionList = null;

      final String PARAM_x0 = "x0";  // параметр
      final String PARAM_x1 = "x1";  // параметр
      final String PARAM_nScale = "Scale";  // параметр
      final String PARAM_bLegend = "Legend"; // параметр
      final String PARAM_nGraphWidth = "GraphWidth"; // параметр
      final String PARAM_nGraphHeight = "GraphHeight"; // параметр
      final String PARAM_nGraphCount = "GraphCount"; // параметр
      final String PARAM_bPunctureLine = "PunctureLine"; // параметр
      final String PARAM_dDelay = "Delay"; // параметр
      final String PARAM_dHmax = "Hmax"; // параметр
    final String PARAM_Beta = "Beta";      ////////////////////////
      final String PARAM_Gamma = "Gamma";
      final String PARAM_Alpha = "Alpha";    //  ПАРАМЕТРИ
      final String PARAM_Mu_c = "Mu_c";
      final String PARAM_Ro = "Ro";
      final String PARAM_Mu_f = "Mu_f";
      final String PARAM_Eta = "Eta";          //       МОДЕЛІ
      final String PARAM_Sigma = "Sigma";
      final String PARAM_Mu_m = "Mu_m";
      final String PARAM_Tau = "Tau";     ////////////////////////



      // другий буфер зображення
      private Image m_image; // позаекранне зображення
      private Graphics m_g; // асоційований з ним графічний об'єкт
      Dimension m_dimImage; // розмір позаекранного зображення


    double f(int i, double x)
      {
            switch(i)
            {
                  case 1:
                        return 1.e4 * SampleSystemSolution.ylag(i,
x);
                  case 2:
                        return .5 * SampleSystemSolution.ylag(i, x);
                  case 3:
                        return SampleSystemSolution.ylag(i, x);
                  case 4:
                        return 10 * SampleSystemSolution.ylag(i, x);
                  default:
                        return 0.;
            }
    }

      synchronized public void init()
      {
            String param;

            param = getParameter(PARAM_x0);
            if (param != null)
                  m_x0 = Double.valueOf(param).doubleValue();
```

```
param = getParameter(PARAM_x1);
            if (param != null)
                  m_x1 = Double.valueOf(param).doubleValue();


            param = getParameter(PARAM_nScale);
            if (param != null)
                  m_nScale = Integer.parseInt(param);

            param = getParameter(PARAM_nGraphWidth);
            if (param != null)
                  m_nGraphWidth = Integer.parseInt(param);

            param = getParameter(PARAM_nGraphHeight);
            if (param != null)
                  m_nGraphHeight = Integer.parseInt(param);

            param = getParameter(PARAM_nGraphCount);
            if (param != null)
                  m_nGraphCount = Integer.parseInt(param);


            param = getParameter(PARAM_bLegend);
            if (param != null)
                  m_bLegend = Boolean.valueOf(param).booleanValue();

            param = getParameter(PARAM_bPunctureLine);
            if (param != null)
                  m_bPunctureLine =
Boolean.valueOf(param).booleanValue();

            param = getParameter(PARAM_dDelay);
            if (param != null)
                  m_dDelay = Double.valueOf(param).doubleValue();

            param = getParameter(PARAM_dHmax);
            if (param != null)
                  m_dHmax = Double.valueOf(param).doubleValue();

            param = getParameter(PARAM_Beta);
            if (param != null)
                  m_dBeta = Double.valueOf(param).doubleValue();

            param = getParameter(PARAM_Gamma);
            if (param != null)
                  m_dGamma = Double.valueOf(param).doubleValue();


            param = getParameter(PARAM_Alpha);
            if (param != null)
                  m_dAlpha = Double.valueOf(param).doubleValue();

            param = getParameter(PARAM_Mu_c);
            if (param != null)
                  m_dMu_c = Double.valueOf(param).doubleValue();

            param = getParameter(PARAM_Ro);
            if (param != null)
                  m_dRo = Double.valueOf(param).doubleValue();

            param = getParameter(PARAM_Mu_f);
            if (param != null)
                  m_dMu_f = Double.valueOf(param).doubleValue();
```

```
param = getParameter(PARAM_Eta);
            if (param != null)
                  m_dEta = Double.valueOf(param).doubleValue();

            param = getParameter(PARAM_Sigma);
            if (param != null)
                  m_dSigma = Double.valueOf(param).doubleValue();

            param = getParameter(PARAM_Mu_m);
            if (param != null)
                  m_dMu_m = Double.valueOf(param).doubleValue();

            param = getParameter(PARAM_Tau);
            if (param != null)
                  m_dTau = Double.valueOf(param).doubleValue();

            // отримання метрик шрифту
            Font f = getFont();
            m_fm = getFontMetrics(f);

            // можна починати режим перетягування
            m_bBeginDrag = true;

            m_bDrag = false;

            m_bWholeScreenMode = false;

            // обчислити відступи від країв для осей x та y
            m_nOffset_x = m_nGraphWidth / 8;
            if (m_bLegend)
            {
                  m_nOffset_y = m_nGraphHeight / 5;
            }
            else
            {
                  m_nOffset_y = m_nGraphHeight / 8;
            }

            // ініціювати зв"язаний список меж визначення функції
            if (m_x0 > m_x1)
            {
                  m_x0 = m_x0 + m_x1;
                  m_x1 = m_x0 - m_x1;
                  m_x0 = m_x0 - m_x1;
            }

            ////////////////////////////////////////////////////

            SampleSystemSolution = new
DelaySystemSolution(m_nGraphCount, m_x0, m_x1, m_dDelay, m_dHmax,
m_dBeta, m_dGamma, m_dAlpha,  m_dMu_c, m_dRo, m_dMu_f, m_dEta,
m_dSigma, m_dMu_m, m_dTau);

            m_x0 = m_x0 - m_dDelay;

            BoundsLocation bl = new BoundsLocation();
            bl.m_dLeftBound = m_x0;
            bl.m_dRightBound = m_x1;
            m_llCursorLocations.Add(bl);

            add(m_button);
            add(m_buttonFewGraphs);
            add(m_buttonWholeScreen);
```

```
ColorList.Add(Color.yellow);
            ColorList.Add(Color.darkGray);
            ColorList.Add(Color.gray);
            ColorList.Add(Color.lightGray);
            ColorList.Add(Color.magenta);
            ColorList.Add(Color.pink);
            ColorList.Add(Color.cyan);
            ColorList.Add(Color.black);
            ColorList.Add(Color.orange);
            ColorList.Add(Color.green);
            ColorList.Add(Color.blue);
            ColorList.Add(Color.red);

      }


    public void paint(Graphics g)
      {

            //    перемалювати існуюче зображення
            if(m_image != null)
            {
                  g.drawImage(m_image, 0, 0, null);
            }
      }


      public void update(Graphics g)
      {
            // переконатися, що зображення має такий самий розмір,
            // що й вікно аплету
            ResizeImage();

            //    очистити позаекранне зображення (а не те, що на
екрані)
            Color colFG = getForeground();
            Color colBG = getBackground();
            m_g.setColor(colBG);
            m_g.fillRect(0, 0, m_dimImage.width, m_dimImage.height);
            m_g.setColor(colFG);


            if (m_bWholeScreenChosen)
            {
                  if (m_nAreaNumber != (m_nGraphCount + 1))
                  {
                        PaintDataFrame(m_g, m_nAreaNumber, 0, 0,
size().width, size().height, m_x0, m_x1, Color.red);
                  }
                  else
                  {
                        PaintFewDataFrame(m_g, 0, 0, size().width,
size().height, m_x0, m_x1);
                  }
            }
            else
            {
                  //  намалювати осі координат і вивести дані
                  PaintFewGraphs(m_g, m_nGraphCount, m_nGraphWidth,
m_nGraphHeight, m_x0, m_x1);
            }


            if (m_bDrag & !(m_bMarkGraphs) & !(m_bWholeScreenMode))
```

```
{
                m_g.setColor(Color.blue);
                //draw Rectangle
                int xPoints[] = {m_dimCursorLocBegin.width,
m_dimDragEvtLoc.width, m_dimDragEvtLoc.width,
m_dimCursorLocBegin.width, m_dimCursorLocBegin.width};
                int yPoints[] = {m_dimCursorLocBegin.height,
m_dimCursorLocBegin.height, m_dimDragEvtLoc.height,
m_dimDragEvtLoc.height, m_dimCursorLocBegin.height};
                m_g.drawPolygon(xPoints, yPoints, 5);
                m_g.drawString(Double.toString(m_x0  +
(m_dimCursorLocBegin.width - (m_dimCursorLocBegin.width /
m_nGraphWidth) * m_nGraphWidth - m_nOffset_x) * m_dStep),
m_dimCursorLocBegin.width + 1, m_dimCursorLocBegin.height + 1);
                m_g.drawString(Double.toString(m_x0  +
(m_dimDragEvtLoc.width - (m_dimCursorLocBegin.width / m_nGraphWidth)
* m_nGraphWidth - m_nOffset_x) * m_dStep), m_dimDragEvtLoc.width + 1,
m_dimDragEvtLoc.height + 1);
           }

           //  тепер перемалювати зображення
           paint(g);

     }


     private void PaintFewGraphs(Graphics g, int n, int nGraphWidth,
int nGraphHeight, double x0, double x1)
     {
           int nCountWidth = size().width / nGraphWidth;
           int nCountHeight = size().height / nGraphHeight;
           int nPossibleToDisplay = nCountWidth * nCountHeight;
           int nFunctionCounter = 0;
           if (n <= nPossibleToDisplay)
           {
UsualGraphs:
                for(int j = 0; j <= nCountHeight - 1; j++)
                {
                     for(int i = 0; i <= nCountWidth - 1; i++)
                     {
                          if (nFunctionCounter++ <= n - 1)
                          {
                               PaintDataFrame(g,
nFunctionCounter, nGraphWidth * i, nGraphHeight * j, nGraphWidth,
nGraphHeight, x0, x1, Color.red);
                          }
                          else
                          {
                               break UsualGraphs;
                          }
                     }
                }
           }
           else
           {
                for(int j = 0; j <= nCountHeight - 1; j++)
                {
                     for(int i = 0; i <= nCountWidth - 1; i++)
                     {
                          PaintDataFrame(g, nFunctionCounter++,
nGraphWidth * i, nGraphHeight * j, nGraphWidth, nGraphHeight, x0, x1,
Color.red);
                     }
                }
```

```
g.drawString("Some graphs couldn't be displayed", nCountWidth *
nGraphWidth, nCountHeight * nGraphHeight);
            }
            // Paint few graphs together
            if (FunctionList.m_head != null)
            {
                  PaintFewDataFrame(g, ((n) % (size().width /
m_nGraphWidth)) * m_nGraphWidth , ((n) / (size().width /
m_nGraphWidth)) * m_nGraphHeight, m_nGraphWidth, m_nGraphHeight, x0,
x1);
            }
      }


      private void PaintFewDataFrame(Graphics g, int nLeftBound, int
nUpperBound, int nWidth, int nHeight, double x0, double x1)
      {
            // m_nFunctionList - numbers of functions f_i(x)
            // nLeftBound - x-coordinate of left-upper corner
            // nUpperBound - y-coordinate of left-upper corner
            // nWidth - width of rectangle for graph and legend
            // nHeight - height of rectangle for graph and legend

            // m_dStep - крок сітки  по x
            m_dStep = (x1 - x0) / (nWidth - m_nOffset_x);

            //задній фон аплету - білий
            g.setColor(Color.green);
            g.drawRect(nLeftBound, nUpperBound, nWidth, nHeight);
            g.setColor(Color.white);
            g.fillRect(nLeftBound + 1, nUpperBound + 1, nWidth - 2,
nHeight - 2);


            //обчислення області значень y
            double ymin = f(FunctionList.m_head.Data(), x0);
            double ymax = f(FunctionList.m_head.Data(), x0);
            double y;
            for (FunctionList fl = FunctionList.m_head; fl != null;
fl = FunctionList.Next(fl))
            {
                  for (double x = x0; x < x1 ; x = x + m_dStep)
                  {
                        y = f(fl.Data(), x);
                        if (y < ymin)
                        {
                              ymin = y;
                        }
                        if (y > ymax)
                        {
                              ymax = y;
                        }
                  }
            }

            // h - крок сітки по y
            double h = (nHeight - m_nOffset_y) / (ymax - ymin);

            ////////////////////////////////////////////////////
            // креслення осей координат і шкал /////////////////
            ////////////////////////////////////////////////////

            // креслення осей
            g.setColor(Color.black);
```

```
g.drawLine(nLeftBound + m_nOffset_x, nUpperBound + nHeight -
m_nOffset_y, nLeftBound + m_nOffset_x, nUpperBound);
            g.drawLine(nLeftBound + m_nOffset_x, nUpperBound +
nHeight - m_nOffset_y, nLeftBound + nWidth, nUpperBound + nHeight -
m_nOffset_y);
            int nSymb_Height = m_fm.getHeight(); // висота символа
            // нанесення шкали      y
            for (int nMark = 0; nMark < nHeight - m_nOffset_y; nMark
+= m_nScale)
            {
                  int nL = 3;
                  if (nMark % (2 * m_nScale) == 0)
                  {
                        nL = 6;
                  }
                  g.drawLine(nLeftBound + m_nOffset_x, nUpperBound +
nHeight - m_nOffset_y - nMark, nLeftBound + m_nOffset_x - nL,
nUpperBound + nHeight - m_nOffset_y - nMark);
                  String sAdd_String = Double.toString(ymin +
(double)(nMark / h));//чисельне значення
                  int nStrWidth =
m_fm.stringWidth(sAdd_String);//ширина чисельної позначки в пікселах
                  while (nStrWidth > m_nOffset_x - nL - 2)
                  {
                        //  підлаштування чисельного значення на осі y
                        sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                        nStrWidth = m_fm.stringWidth(sAdd_String);
                  }
                  g.drawString(sAdd_String, nLeftBound + m_nOffset_x
- nStrWidth - 8, nUpperBound + nHeight - m_nOffset_y - nMark +
nSymb_Height / 2);
            }
            //    нанесення шкали x
            for (int nMark = 0; nMark < nWidth - m_nOffset_x; nMark
+= m_nScale)
            {
                  int nL = 3;
                  if (nMark % (2 * m_nScale) == 0)
                  {
                        nL = 6;
                  }
                  g.drawLine(nLeftBound + m_nOffset_x + nMark,
nUpperBound + nHeight - m_nOffset_y, nLeftBound + m_nOffset_x +
nMark, nUpperBound + nHeight - m_nOffset_y + nL);
                  String sAdd_String = Double.toString(x0 + nMark *
m_dStep);//чисельне значення
                  int nStrWidth = m_fm.stringWidth(sAdd_String) /
2;//ширина чисельної позначки в пікселах / 2
                  while (nStrWidth > (m_nScale / 2) - 1)
                  {
                        //  підлаштування чисельного значення на осі
                        sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                        nStrWidth = m_fm.stringWidth(sAdd_String) /
2;
                  }
                  g.drawString(sAdd_String, nLeftBound + m_nOffset_x
+ nMark - nStrWidth, nUpperBound + nHeight - m_nOffset_y +
nSymb_Height + 8);
            }


            ////////////////////////////////////////////////
            // зображення легенди і ///////////////////////
            ////////////////////////////////////////////////

            ////////////////////////////////////////////////
```

```
// креслення графіків ////////////////////////////
            ////////////////////////////////////////////////////
            int nCellWidth = 70;
            int nLegendOffset = 5;
            int nGraphWidthForLegend = nWidth - nLegendOffset;
            int nCellHeight = (m_nOffset_y - nSymb_Height - 8) /
((FunctionList.m_nNumberOfFunctions / (nGraphWidthForLegend /
nCellWidth)) + 1);
            ColorList cl = ColorList.m_head;
            int FunctionCounter = 0;
            for (FunctionList fl = FunctionList.m_head; fl != null;
fl = FunctionList.Next(fl))
            {
                  g.setColor(cl.Data());
                  PaintLegendCell(g, nLeftBound + nLegendOffset +
((FunctionCounter) % (nGraphWidthForLegend / nCellWidth)) *
nCellWidth , nUpperBound + nHeight - m_nOffset_y + nSymb_Height + 10
+ ((FunctionCounter) / (nGraphWidthForLegend / nCellWidth)) *
nCellHeight, nCellWidth, nCellHeight, fl.Data(), cl.Data());
                  FunctionCounter++;
                  int xStart, yStart, xEnd, yEnd;
                  double dLengthRemainder = 0;
                  for (int ix = 0 ; ix < nWidth - m_nOffset_x; ix++)
                  {
                        xStart = nLeftBound + m_nOffset_x + ix;
                        yStart = nUpperBound + (int)(nHeight -
m_nOffset_y - (f(fl.Data(), x0 + ix * m_dStep) - ymin) * h);
                        xEnd = nLeftBound + m_nOffset_x + ix + 1;
                        yEnd = nUpperBound + (int)(nHeight -
m_nOffset_y - (f(fl.Data(), x0 + (ix + 1) * m_dStep) - ymin) * h);
                        if(! m_bPunctureLine)
                        {
                              g.drawLine(xStart, yStart, xEnd, yEnd);
                        }
                        else
                        {
                              dLengthRemainder = drawPunctureLine(g,
dLengthRemainder, xStart, yStart, xEnd, yEnd, 5 * (fl.Data() - 1),
cl.Data());
                        }
                  }
                  cl = ColorList.Next(cl);
            }
    }

      private void PaintLegendCell(Graphics g, int nLeftBound, int
nUpperBound, int nWidth, int nHeight, int nFunctionNumber, Color
colLineColor)
      {
            int nSymb_Height = m_fm.getHeight(); // висота символа
            String sCaption = "x" +
Integer.toString(nFunctionNumber);
            g.setColor(Color.black);
            g.drawString(sCaption, (int)(nLeftBound + (3./4.) *
nWidth - m_fm.stringWidth(sCaption)), (int)(nUpperBound + (nHeight +
nSymb_Height) / 2));
            if (m_bPunctureLine)
            {
                  drawPunctureLine(g, 0, nLeftBound, nUpperBound +
(nHeight / 2), (int)(nLeftBound + (3./4.) * nWidth -
m_fm.stringWidth(sCaption) - 5),  nUpperBound + (nHeight / 2), 5 *
(nFunctionNumber - 1), colLineColor);
            }
            else
            {
                  g.setColor(colLineColor);
                  g.drawLine(nLeftBound, nUpperBound + (nHeight /
2),(int)(nLeftBound + (3./4.) * nWidth - m_fm.stringWidth(sCaption) -
5),  nUpperBound + (nHeight / 2));
```

```
}

      }


      private void PaintDataFrame(Graphics g, int nFunctionNumber,
int nLeftBound, int nUpperBound, int nWidth, int nHeight, double x0,
double x1, Color GraphColor)
      {
            // nFunctionNumber - number of function f_i(x)
            // nLeftBound - x-coordinate of left-upper corner
            // nUpperBound - y-coordinate of left-upper corner
            // nWidth - width of rectangle for graph and legend
            // nHeight - height of rectangle for graph and legend

            // m_dStep - крок сітки  по x
            m_dStep = (x1 - x0) / (nWidth - m_nOffset_x);

            //задній фон аплету - білий
            g.setColor(Color.yellow);
            g.drawRect(nLeftBound, nUpperBound, nWidth, nHeight);
            g.setColor(Color.white);
            g.fillRect(nLeftBound + 1, nUpperBound + 1, nWidth - 2,
nHeight - 2);


            //обчислення області значень y
            double ymin = f(nFunctionNumber, x0);
            double ymax = f(nFunctionNumber, x0);
            for (double x = x0; x < x1 ; x = x + m_dStep)
            {
                  double y = f(nFunctionNumber, x);
                  if (y < ymin)
                  {
                      ymin = y;
                  }
                  if (y > ymax)
                  {
                        ymax = y;
                  }
            }

            // h - крок сітки по y
            double h = (nHeight - m_nOffset_y) / (ymax - ymin);

            ////////////////////////////////////////////////////
            // креслення осей координат і шкал /////////////////
            ////////////////////////////////////////////////////

            // креслення осей
            g.setColor(Color.black);
            g.drawLine(nLeftBound + m_nOffset_x, nUpperBound +
nHeight - m_nOffset_y, nLeftBound + m_nOffset_x, nUpperBound);
            g.drawLine(nLeftBound + m_nOffset_x, nUpperBound +
nHeight - m_nOffset_y, nLeftBound + nWidth, nUpperBound + nHeight -
m_nOffset_y);
            int nSymb_Height = m_fm.getHeight(); // висота символа
            // нанесення шкали      y
            for (int nMark = 0; nMark < nHeight - m_nOffset_y; nMark
+= m_nScale)
            {
                  int nL = 3;
                  if (nMark % (2 * m_nScale) == 0)
                  {
```

```
nL = 6;
                  }
                  g.drawLine(nLeftBound + m_nOffset_x, nUpperBound +
nHeight - m_nOffset_y - nMark, nLeftBound + m_nOffset_x - nL,
nUpperBound + nHeight - m_nOffset_y - nMark);
                  String sAdd_String = Double.toString(ymin +
(double)(nMark / h));//чисельне значення
                  int nStrWidth =
m_fm.stringWidth(sAdd_String);//ширина чисельної позначки в пікселах
                  while (nStrWidth > m_nOffset_x - nL - 2)
                  {
                        //  налаштування чисельного значення на осі
                        sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                        nStrWidth = m_fm.stringWidth(sAdd_String);
                  }
                  g.drawString(sAdd_String, nLeftBound + m_nOffset_x
- nStrWidth - 8, nUpperBound + nHeight - m_nOffset_y - nMark +
nSymb_Height / 2);
            }
            //    нанесення шкали x
            for (int nMark = 0; nMark < nWidth - m_nOffset_x; nMark
+= m_nScale)
            {
                  int nL = 3;
                  if (nMark % (2 * m_nScale) == 0)
                  {
                        nL = 6;
                  }
                  g.drawLine(nLeftBound + m_nOffset_x + nMark,
nUpperBound + nHeight - m_nOffset_y, nLeftBound + m_nOffset_x +
nMark, nUpperBound + nHeight - m_nOffset_y + nL);
                  String sAdd_String = Double.toString(x0 + nMark *
m_dStep);//чисельне значення
                  int nStrWidth = m_fm.stringWidth(sAdd_String) /
2;//ширина чисельної позначки в пікселах / 2
                  while (nStrWidth > (m_nScale / 2) - 1)
                  {
                        //  налаштування чисельного значення на осі
                        sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                        nStrWidth = m_fm.stringWidth(sAdd_String) /
2;
                  }
                  g.drawString(sAdd_String, nLeftBound + m_nOffset_x
+ nMark - nStrWidth, nUpperBound + nHeight - m_nOffset_y +
nSymb_Height + 8);
            }


            ////////////////////////////////////////////////
            // зображення легенди //////////////////////////
            ////////////////////////////////////////////////
            if (m_bLegend)
            {
                  // minimal  value
                  String sAdd_String = "Minimal value = " +
Double.toString(ymin);//
                  int nStrWidth =
m_fm.stringWidth(sAdd_String);//ширина
                  while (nStrWidth > (nWidth / 2) - 10)
                  {
                        //  налаштування
                        sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                        nStrWidth = m_fm.stringWidth(sAdd_String);
                  }
```

```
g.drawString(sAdd_String, nLeftBound + (int)((nWidth / 4.) -
(nStrWidth / 2.)), nUpperBound + (int)((nHeight / 3.) + (2. *
(nHeight - m_nOffset_y + nSymb_Height + 8.) / 3.) + (nSymb_Height /
2.)));


                // left bound
                sAdd_String = "Left bound = " +
Double.toString(x0);
                nStrWidth = m_fm.stringWidth(sAdd_String);//ширина
                while (nStrWidth > (nWidth / 2) - 10)
                {
                      //  налаштування
                      sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                      nStrWidth = m_fm.stringWidth(sAdd_String);
                }

                g.drawString(sAdd_String, nLeftBound + (int)(((3. *
nWidth) / 4.) - (nStrWidth / 2.)),nUpperBound + (int)((nHeight / 3.)
+ (2. * (nHeight - m_nOffset_y + nSymb_Height + 8.) / 3.) +
(nSymb_Height / 2.)));

                // maximal value
                sAdd_String = "Maximal value = " +
Double.toString(ymax);
                nStrWidth = m_fm.stringWidth(sAdd_String);//ширина
                while (nStrWidth > (nWidth / 2) - 10)
                {
                      //  налаштування
                      sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                      nStrWidth = m_fm.stringWidth(sAdd_String);
                }

                g.drawString(sAdd_String, nLeftBound +
(int)((nWidth / 4.) - (nStrWidth / 2.)),nUpperBound + (int)(((2. *
nHeight) / 3.) + (nHeight - m_nOffset_y + nSymb_Height + 8.) / 3. +
(nSymb_Height / 2.)));

                // right bound
                sAdd_String = "Right bound = " +
Double.toString(x1);
                nStrWidth = m_fm.stringWidth(sAdd_String);//ширина
                while (nStrWidth > (nWidth / 2) - 10)
                {
                      //  налаштування
                      sAdd_String = sAdd_String.substring(0,
sAdd_String.length() - 1);
                      nStrWidth = m_fm.stringWidth(sAdd_String);
                }

                g.drawString(sAdd_String, nLeftBound + (int)((3. *
nWidth / 4.) - (nStrWidth / 2.)),nUpperBound + (int)(((2. * nHeight)
/ 3.) + (nHeight - m_nOffset_y + nSymb_Height + 8.) / 3. +
(nSymb_Height / 2.)));

           }

           ////////////////////////////////////////////////
           // креслення графіка ///////////////////////////
           ////////////////////////////////////////////////
           g.setColor(GraphColor);
           int xStart, yStart, xEnd, yEnd;
       for (int ix = 0 ; ix < nWidth - m_nOffset_x; ix++)
           {
                xStart = nLeftBound + m_nOffset_x + ix;
                yStart = nUpperBound + (int)(nHeight - m_nOffset_y
- (f(nFunctionNumber, x0 + ix * m_dStep) - ymin) * h);
                xEnd = nLeftBound + m_nOffset_x + ix + 1;
```

```
yEnd = nUpperBound + (int)(nHeight - m_nOffset_y -
(f(nFunctionNumber, x0 + (ix + 1) * m_dStep) - ymin) * h);

                g.drawLine(xStart, yStart, xEnd, yEnd);

          }
    }



     ////////////////////////////////////////////////////////
     // креслення пунктирної лінії //////////////////////////
     ////////////////////////////////////////////////////////
     private double drawPunctureLine(Graphics g, double
dLengthRemainder, int x0, int y0, int x1, int y1, double
dPieceLength, Color colLineColor)
     {
          dPieceLength = Math.abs(dPieceLength);
          g.setColor(colLineColor);
          if (dPieceLength == 0)
          {
               g.drawLine(x0, y0, x1, y1);
               return 0;
          }
          else
          {
               int nDirection_x = x1 - x0;
               int nDirection_y = y1 - y0;
               double dLineLength = lineLength(x0, y0, x1, y1);
               if (Math.abs(dLengthRemainder) > dLineLength)
               {
                    if(dLengthRemainder > 0)
                    {
                         g.drawLine(x0, y0, x1, y1);
                         return dLengthRemainder - dLineLength;
                    }
                    else
                    {
                         return -(Math.abs(dLengthRemainder) -
dLineLength);
                    }
               }
               else
               {
                    boolean bDraw = true;
                    int x1_new = x0 +
(int)((Math.abs(dLengthRemainder) / dLineLength) * nDirection_x);
                    int y1_new = y0 +
(int)((Math.abs(dLengthRemainder) / dLineLength) * nDirection_y);
                    if(dLengthRemainder > 0)
                    {
                         g.drawLine(x0, y0, x1_new, y1_new);
                         bDraw = false;
                    }
                    int x0_new = x1_new;
                    int y0_new = y1_new;
                    double dCurrentLengthRemainder = dLineLength
- Math.abs(dLengthRemainder);
                    while(dCurrentLengthRemainder >=
dPieceLength)
                    {
                         x1_new = x0_new + (int)((dPieceLength /
dLineLength) * nDirection_x);
```

```
y1_new = y0_new + (int)(((dPieceLength / dLineLength) * nDirection_y);
                              if(bDraw)
                              {
                                    g.drawLine(x0_new, y0_new,
x1_new, y1_new);
                              }
                              x0_new = x1_new;
                              y0_new = y1_new;
                              bDraw = ! bDraw;
                              dCurrentLengthRemainder =
dCurrentLengthRemainder - dPieceLength;
                        }
                        // Finally, for dCurrentLengthRemainder >= 0
                        if(bDraw)
                        {
                              g.drawLine(x0_new, y0_new, x1, y1);
                              return (dPieceLength -
dCurrentLengthRemainder);
                        }
                        else
                        {
                              return (dCurrentLengthRemainder -
dPieceLength);
                        }
                  }
            }
      }



      private double lineLength(int x0, int y0, int x1, int y1)
      {
            return Math.sqrt(Math.pow(x1 - x0, 2) + Math.pow(y1 - y0,
2));
      }

      public void start()
      {
            if (m_Graphics == null)
            {
                  m_Graphics = new Thread(this);
                  m_Graphics.start();
            }
      }

      public void stop()
      {
            if (m_Graphics != null)
            {
                  m_Graphics.stop();
                  m_Graphics = null;
            }
      }

      public void run()
      {
            int   nSleepTime = (int)(1000 / m_fps);

            while (true)
            {
                  try
                  {
                        repaint();

                        // перемальовувати із швидкістю fps
```

```
Thread.sleep(nSleepTime);
                  }
                  catch (InterruptedException e)
                  {
                        stop();
                  }
            }
      }

      public boolean mouseDrag(Event evt, int x, int y)
      {
            m_bDrag = true;
            if ((m_bBeginDrag == true) & !m_bMarkGraphs &
!m_bWholeScreenMode)
            {
                  m_dimCursorLocBegin = new Dimension(x, y);
                  if(x - (x / m_nGraphWidth) * m_nGraphWidth <
m_nOffset_x)
                  {
                        m_dimCursorLocBegin.width = (x /
m_nGraphWidth) * m_nGraphWidth + m_nOffset_x;                    ;
                  }
                  if(y - (y / m_nGraphHeight) * m_nGraphHeight >
m_nGraphHeight - m_nOffset_y)
                  {
                        m_dimCursorLocBegin.height = (y /
m_nGraphHeight) * m_nGraphHeight + m_nGraphHeight - m_nOffset_y;
                  ;
                  }
                  m_bBeginDrag = false;
            }

      ////////////////////////////////////////////////////////////
////////////
            int k_width = m_dimCursorLocBegin.width / m_nGraphWidth;
            int k_height = m_dimCursorLocBegin.height /
m_nGraphHeight;
            m_dimDragEvtLoc = new Dimension(x, y);
            if(x / m_nGraphWidth > k_width)
            {
                  m_dimDragEvtLoc.width = m_nGraphWidth * (k_width +
1);
            }
            if(x - k_width * m_nGraphWidth < m_nOffset_x)
            {
                  m_dimDragEvtLoc.width = k_width * m_nGraphWidth +
m_nOffset_x;
            }
            if(y / m_nGraphHeight < k_height)
            {
                  m_dimDragEvtLoc.height = m_nGraphHeight * k_height;

            }
            if(y - k_height * m_nGraphHeight > m_nGraphHeight -
m_nOffset_y)
            {
                  m_dimDragEvtLoc.height = k_height * m_nGraphHeight
+ m_nGraphHeight - m_nOffset_y;
            }

      ////////////////////////////////////////////////////////////
////////////
            repaint();
            return true;  // кінець обробки події
      }

      public boolean mouseUp(Event evt, int x, int y)
      {
```

```
m_nAreaNumber = ((x / m_nGraphWidth) + 1) + (y /
m_nGraphHeight)*(size().width / m_nGraphWidth);
            if (m_bWholeScreenChosen)
            {
                  m_bWholeScreenMode = false;
                  m_bWholeScreenChosen = false;
                  repaint();
                  m_button.show();
                  m_buttonFewGraphs.show();
                  m_buttonWholeScreen.setLabel("Whole screen");
                  m_buttonWholeScreen.show();
            }

            if (m_bWholeScreenMode & (m_nAreaNumber >= 1) &
(m_nAreaNumber <= m_nGraphCount + 1))
            {
                  m_bWholeScreenChosen = true;
                  m_button.hide();
                  m_buttonFewGraphs.hide();
                  m_buttonWholeScreen.hide();
                  repaint();
            }

            if (m_bMarkGraphs & (m_nAreaNumber >= 1) & (m_nAreaNumber
<= m_nGraphCount))
            {
                  m_nFunctionList.Add(((x / m_nGraphWidth) + 1) + (y
/ m_nGraphHeight)*(size().width / m_nGraphWidth));
                  repaint();
            }

            m_bDrag = false;
            if ((m_bBeginDrag == false) & !m_bMarkGraphs &
!m_bWholeScreenMode)
            {
                  int k_width = m_dimCursorLocBegin.width /
m_nGraphWidth;
                  int k_height = m_dimCursorLocBegin.height /
m_nGraphHeight;

                  m_dimCursorLocEnd = new Dimension(x, y);
                  if(x / m_nGraphWidth > k_width)
                  {
                        m_dimDragEvtLoc.width = m_nGraphWidth *
(k_width + 1);
                  }
                  if(x - k_width * m_nGraphWidth < m_nOffset_x)
                  {
                        m_dimDragEvtLoc.width = k_width *
m_nGraphWidth + m_nOffset_x;
                  }
                  if(y / m_nGraphHeight > k_height)
                  {
                        m_dimDragEvtLoc.height = m_nGraphHeight *
(k_height + 1);
                  }
                  if(y - k_height * m_nGraphHeight > m_nGraphHeight -
m_nOffset_y)
                  {
                        m_dimDragEvtLoc.height = k_height *
m_nGraphHeight + m_nGraphHeight - m_nOffset_y;
                  }

                  m_bBeginDrag = true;

                  // нові межі визначення функції
```

```
m_x1 = m_x0 + (m_dimCursorLocEnd.width - (m_dimCursorLocEnd.width /
m_nGraphWidth) * m_nGraphWidth - m_nOffset_x) * m_dStep;
                  m_x0 = m_x0 + (m_dimCursorLocBegin.width -
(m_dimCursorLocBegin.width / m_nGraphWidth) * m_nGraphWidth -
m_nOffset_x) * m_dStep;
                  if (m_x0 != m_x1)
                  {
                        if (m_x0 > m_x1)
                        {
                              m_x0 = m_x0 + m_x1;
                              m_x1 = m_x0 - m_x1;
                              m_x0 = m_x0 - m_x1;
                        }

                        // добавити новий елемент у голову зв"язаного
списку
                        BoundsLocation bl = new BoundsLocation();
                        bl.m_dLeftBound = m_x0;
                        bl.m_dRightBound = m_x1;
                        m_llCursorLocations.Add(bl);

      ////////////////////////////////////////////////

                        repaint();   // перемалювання графіка в
новому інтервалі
                  }
            }
            return true; // кінець обробки події
      }



      public boolean action(Event event, Object obj)
      {
            Object oTarget = event.target;
            if (oTarget instanceof Button)
            {
                  Button buttonTarget = (Button)oTarget;
                  String sButtonString = buttonTarget.getLabel();
                  if (sButtonString.compareTo("Back") == 0)
                  {
                        // нові межі визначення функції
                        m_llCursorLocations =
LinkedList.Next(m_llCursorLocations);
                        m_x1 =
m_llCursorLocations.Data().m_dRightBound;
                        m_x0 =
m_llCursorLocations.Data().m_dLeftBound;

                        repaint();   // перемалювання графіка в
новому інтервалі
                        return true;
                  }
                  if (sButtonString.compareTo("Mark few graphs") ==
0)
                  {
                        m_buttonFewGraphs.setLabel("Marking...");
                        FunctionList.m_head = null;
                        FunctionList.m_nNumberOfFunctions = 0;
                        m_bMarkGraphs = true;

                        repaint();
                  }
                  if (sButtonString.compareTo("Marking...") == 0)
                  {
                        m_buttonFewGraphs.setLabel("Mark few
graphs");
                        m_bMarkGraphs = false;
                  }
                  if (sButtonString.compareTo("Whole screen") == 0)
```

```
{
                        m_bWholeScreenMode = true;
                        m_buttonWholeScreen.setLabel("Point graph");
                  }


            }
            return false;
      }


      private void ResizeImage()
      {
            // отримати розмір вікна аплету
            Dimension dim = size();
            int nWidth = dim.width;
            int nHeight = dim.height;


            // порівняємо його з розміром нашого зображення;
            // якщо він не був зміненим ...
            if(m_dimImage != null &&
                  m_dimImage.width == nWidth &&
                  m_dimImage.height == nHeight)
            {
                  //    ... не робити нічого
                  return;
            }
            // створити графічне зображення для виводу
            m_dimImage = new Dimension(nWidth, nHeight);
            m_image = createImage(nWidth, nHeight);
            m_g = m_image.getGraphics();

      }

}

class FunctionList
{
      public static FunctionList m_head = null;
      public static FunctionList m_tail = null;
      public static int m_nNumberOfFunctions = 0;

      private FunctionList m_next;
      private int m_nIndex;

      public static void Add(int n)
      {
            FunctionList fl = new FunctionList();

            m_nNumberOfFunctions++;
            fl.m_nIndex = n;
            fl.m_next = m_head;

            if (m_head == null)
            {
                  m_head = fl;
                  m_tail = fl;
            }
            else
            {
                  m_head = fl;
            }
```

```
}

      public static FunctionList Next(FunctionList fl)
      {
            return fl.m_next;
      }

      public int Data()
      {
            return m_nIndex;
      }

}


class ColorList
{
      public static ColorList m_head = null;
      private static ColorList m_tail = null;

      private ColorList m_next;
      private Color m_colObject;

      public static void Add(Color col)
      {
            ColorList cl = new ColorList();

            cl.m_colObject = col;
            cl.m_next = m_head;

            if (m_head == null)
            {
                  m_head = cl;
                  m_tail = cl;
            }
            else
            {
                  m_head = cl;
            }
      }

      public static ColorList Next(ColorList cl)
      {
            if (cl.m_next == null)
            {
                  return m_head;
            }
            return cl.m_next;
      }

      public Color Data()
      {
            return m_colObject;
      }

}
```

Навчальний посібник

Марценюк Василь Петрович
Семенець Андрій Володимирович

Медична інформатика. Інструментальні та експертні системи

Літературний редактор	*Ірина Папуша*
Технічний редактор	*Світлана Демчишин*
Коректор	*Наталя Сороката*
Оформлення обкладинки	*Павло Кушик*
Комп'ютерна верстка	*Ірина Петрикович*

Підписано до друку 09.04.2001. Формат 60x84/16.
Папір офсетний № 1. Гарнітура Antiqua. Друк офсетний.
Ум. др. арк. 10,35. Обл.-вид. арк. 7,69.
Наклад 2000. Зам. № 67.

Оригінал-макет підготовлено у відділі комп'ютерної верстки
видавництва "Укрмедкнига" Тернопільської державної медичної академії
ім. І.Я. Горбачевського.
Майдан Волі, 1, м. Тернопіль, 46001, Україна.

Надруковано у друкарні видавництва "Укрмедкнига" Тернопільської
державної медичної академії ім. І.Я. Горбачевського.
Майдан Волі, 1, м. Тернопіль, 46001, Україна.

Свідоцтво про внесення до державного
реєстру суб'єктів видавничої справи
ДК № 348 від 02.03.2001 р.