МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

К О Н С П Е К Т Л Е К Ц І Й

по курсу «Інформатика та обчислювальна техніка» для студентів фізико-металургійного факультету

Затверджено на засіданні кафедри «Обчислювальна математика та програмування»
Протокол № 3 від 6.10.2010

Донецьк 2010

УДК 681.3.06 (071)

Конспект лекцій по курсу «Інформатика та обчислювальна техніка» для студентів фізико-металургійного факультету // Укл. Анохина І.Ю., Кононенко І.М./ Донецьк, ДонНТУ – 2010 -176с.

Конспект лекцій містить відомості щодо створення додатків різного характеру та рівня складності, призначений для отримання навиків роботи в середовищі програмування Delphi.

Розраховано на студентів фізико-металургійного факультету та студентів інших спеціальностей, які вивчають в межах курсу «Інформатика та комп'ютерна техніка» розділ «Програмування в середовищі «Delphi».

Рецензент: Копитова О.М.
Відп. за випуск: Анохина І.Ю.

ВСТУП

Початкове призначення комп'ютерних технологій - допомога людині в обробці і створенні інформації, а зовсім не "служіння самим собі", своєму подальшому розвитку. Тому основна функція більшості комп'ютерних програм - отримання потрібного результату, обробка інформації, що вводиться.

Створити ж програму, що дає дійсно необхідний результат, може лише людина, що безпосередньо працює в області, для якої дана програма пишеться, - лікар, інженер, астроном, математик. Тільки фахівець знає всі тонкощі своєї справи і зможе передбачити в програмі всі необхідні умови. Професійний програміст, хоч і має можливість створити програму, яка працюватиме у декілька разів швидше і займатиме на диску значно менше місця, чим написана інженером або лікарем, не уявляє собі тонкощів області людської діяльності, для якої ця програма пишеться, а тому, при всій зовнішній привабливості, така програма працюватиме гірше, ніж перша. І це нормально - у кожного свій талант, свої дарування. Але, завдяки старанням сотень і тисяч програмістів, частіше безвісних, зараз існують середовища програмування, освоїти які без проблем може кожен. І тому практично будь-який фахівець своєї справи може в наші дні використовувати в своїй роботі новітні досягнення інформаційних технологій - створити потрібну йому програму, розробивши її алгоритм на основі свого досвіду.

І тому не варто так вже прагнути обов'язково знати асемблер, вивчити всі функції API і жаліти, що не можеш вводити програми відразу в машинних кодах, - краще уважніше вивчити Delphi для того, щоб уміти використовувати всі його можливості для втілення своїх побажань в життя.

Програма, створена в цьому середовищі, дозволяє не тільки проводити обчислення за формуламих будь-якої складності, але й давати графічну інтерпретацію отриманим даним. За допомогою Delphi створюються програми з анімацією, будуються тривимірні зображення довільних фігур.

Зручний інтерфейс, вбудовані компоненти дозволяють вважати вказане середовище програмування однимї з найбільш доступних для вивчення як професіоналами, так і тим, чиї професійні інтереси далекі від програмування, наприклад, металургам, гірникам, економістам.

Тому цей продукт заслуговує пильного вивчення, чим ми і пропонуємо вам занятся.

Не варто думати, що "програмування мені ніколи не знадобиться" - сучасні комп'ютерні технології розвиваються з величезною швидкістю, і практично будь-яка область людської діяльності компьютеризирована. А щоб отримати в своїй роботі якнайкращий результат, потрібно уміти пристосувати знаряддя праці - комп'ютер - під свої потреби, тобто - програмувати.

ЗМІСТ

Основні поняття

Кожна наука має свої основоположні поняття. У фізиці - це маса тіла і швидкість, щільність і температура. У математиці - це цифри і числа, математичні операції. Одне з основоположних понять інформатики – це ***алгоритм.***

Алгоритм — це порядок дій, що дозволяють перейти від заданого початкового положення в намічений кінцевий стан.

Слово алгоритм відбулося від латинського написання слова Аль–Хорезмі (algorithm), під яким в середньовічній Європі знали найбільшого математика з Хорезма Мухамеда бен Мусу, що жив в 783—850 рр. Ви знайомі з одним з його творінь – діями над числами «стовпчиком».

З поняттям алгоритму ми стикаємося щодня. Відповідаючи на питання, як пройти абикуди, ви формулюєте алгоритм дій: спочатку направо, два повороти наліво і три сходинки вгору. Рецепти приготування блюд – це кулінарні алгоритми: все порізати, змішати і варити впродовж 30 хвилин. Складаючи вранці плани на день, ви і тут не обходитеся без алгоритму: спочатку поїхати в інститут; якщо вийде, втекти з другої пари, якщо ні, то - з третьої; заїхати до друга; повернутися додому і зробити креслення до зустрічі з друзями на дискотеці.

До обчислювальних алгоритмів, що однозначно визначають процес отримання результату з початкових даних, пред'являється ряд вимог .

Алгоритм, що створюється для подальшого перетворення на комп'ютерну програму, повинен бути представлений у вигляді послідовного виконання кроків. Це властивість алгоритму називається дискретністю. Під *кроком* розуміється кожна дія алгоритму. Допустимо, потрібно знайти суму натуральних чисел від 1 до 1000. Якщо просто написати знак суми між цими числами, то поставлене завдання буде виконано невірно. А ось якщо розписати процес отримання суми як виконання послідовності операцій складання, то такий набір елементарних дій приведе до рішення поставленої задачі.

Алгоритм виконується покроково. Кожен з кроків повинен бути однозначно здійснимий. Цю властивість називають визначеністю. Наприклад, в одному з кулінарних рецептів сказано: злегка розмішати, підігріти і влити трохи молока. Чи зрозумілий

такий опис? Що означає «злегка», до якої температури слід підігріти і яку кількість молока потрібно влити?

У алгоритмі не повинні бути присутніми невизначені слова: небагато, трошки, злегка і так далі

Результативність — алгоритм повинен приводити до рішення задачі за певну кількість кроків. Ця властивість має на увазі, що кожен крок (і алгоритм в цілому) після свого завершення дає середовище, в якому всі наявні об'єкти однозначно визначені. Якщо це за будь-якими причинами неможливо, то повинно бути передбачене повідомлення, що рішення задачі не існує.

Прикладом може служити алгоритм вирішення квадратного рівняння. У тому випадку, коли дискримінант приймає від'ємне значення, важливо отримати повідомлення про відсутність дійсного кореня, що і буде результатом роботи алгоритму.

Масовість — алгоритм складається в загальному виді, тобто він повинен бути застосовний до ряду завдань, що розрізняються початковими даними.

При розробці алгоритму його прагнуть описати так, щоб можна було запускати програму на розрахунок при різних початкових даних. Наприклад, розраховуючи траєкторію руху снаряда як початкові дані задають його масу, початкову швидкість, швидкість вітру. Ці величини не вводяться в програму як константи, а задаються у вигляді змінних. В цьому випадку при розрахунку траєкторії польоту снаряда з іншою масою досить ввести нові початкові дані, а не змінювати текст програми.

Залежно від поставленого завдання і послідовності виконуваних кроків розрізняють наступні види алгоритмів.

Лінійний — кроки алгоритму слідують один за одним не повторюючись, дії відбуваються тільки в одній заздалегідь означеній послідовності. Це аналогічно почерговому обчисленню формул. Наприклад, необхідно обчислити

$z=3x^2+y$

$y=x+x^2+x^3$

$x=a+b$, де a, b – початкові дані.

Якби проводили розрахунки уручну, то спочатку визначили б значення x, потім - y і тільки потім - z. Математичні дії велися б без перевірок умов, послідовно, один розрахунок за іншим. Це і є лінійний алгоритм.

Алгоритм з розгалуженою структурою — залежно від виконання або невиконання якої-небудь умови проводяться різні послідовності дій. Кожна така послідовність дій називається *гілкою* алгоритму, виконується на вибір або одна, або інша його гілка.

Якщо дискримінант квадратного рівняння більше нуля, то слід визначити за формулою два корені; якщо він рівний нулю, то визначається один корінь і, якщо менше нуля, то видається повідомлення про відсутність дійсного коріння. Залежно від коефіцієнтів квадратного рівняння буде реалізована тільки одна гілка алгоритму і ніколи все три разом. Дискримінант не може бути одночасно і додатнім, і від'ємним і нульовим.

Якщо Ви дивитеся на світлофор, то залежно від його кольору виконується одна з трьох дій: червоний – стоїте, бо переходити вулицю не можна, жовтий – чекаєте зміни кольору і на зелений – переходите. Колір не може бути одночасно і жовтим, і червоним, і зеленим.

Алгоритм циклічної структури. Залежно від виконання або невиконання деякої умови повторюється певна послідовність дій, звана *тілом циклу*. Процес такого типу називається циклічним. Кожного ранку прокидаючись ми чистимо зуби, снідаємо... Ці дії повторюються день за днем, доки ми живі. Такий циклічний процес називається циклом *з невідомим числом повторень*. Кожен понеділок ми йдемо на роботу, і так п'ять днів на тиждень. Цей циклічний процес називається циклом *з відомим числом повторень*.

При створенні програми виконується декілька обов'язкових етапів.

Постановка завдання — складання точного і зрозумілого словесного опису того, як повинна працювати майбутня програма і що повинен робити користувач в процесі її роботи.

Розробка інтерфейсу (інтерфейс — спосіб спілкування) — створення екранної форми (вікна програми).

Складання алгоритму. Визначення, яким чином повинний бути отриманий результат, вибір математичних, статистичних і інших формул, створення моделі.

Програмування — розробка програмного коду на мові програмування.

Відладка програми – процес пошуку і виправлення помилок, виявлених під час виконання програми.

Тестування програми — перевірка правильності її роботи на якнайможна більшій кількості вхідних наборів даних, у тому числі і на свідомо невірних.

Створення документації, яка описує роботу програми, допомоги. Якщо розробник припускає, що програмою користуватимуться інші, то він обов'язково повинен створити довідкову систему і забезпечити користувачеві зручний доступ до довідкової інформації під час роботи з програмою.

Програма, представлена у вигляді інструкцій мови програмування, називається початковою програмою. Вона складається з інструкцій, зрозумілих людині, але не зрозумілих процесору комп'ютера. Щоб процесор зміг виконати роботу відповідно до інструкцій початкової програми, початкова програма повинна бути перекладена машинною мовою — мовою команд процесора. Завдання перетворення початкової програми в машинний код виконує спеціальна програма — *компілятор*.

Слід зазначити, що генерація виконуваної програми відбувається тільки в тому випадку, якщо в тексті початкової програми немає синтаксичних помилок. Переконатися, що програма працює правильно можна тільки в процесі її тестування — пробних запусках програми і аналізі отриманих результатів.

У середовищі програмування Delphi програма є послідовністю інструкцій, які досить часто називають *операторами*. Одна інструкція від іншої відділяється крапкою з комою.

Програма може оперувати даними різних типів: цілими і дробовими числами, символами, рядками символів, логічними величинами.

1 ОСНОВИ РОБОТИ В СЕРЕДОВИЩІ РОЗРОБКИ DELPHI

Середовище програмування Delphi - це складний механізм, що візуально реалізується декількома одночасно розкритими на екрані вікнами, що несуть в собі деяку функціональність, тобто призначеними для зміни тих або інших функціональних властивостей створюваної вами програми.

Для запуску інтегрованого середовища розробки Delphi відкрийте пункт меню Программи|Borland Delphi|Delphi або знайдіть файл delphi32.exe, який використовується для запуску. Вид вікна показаний на рис. 1.1.

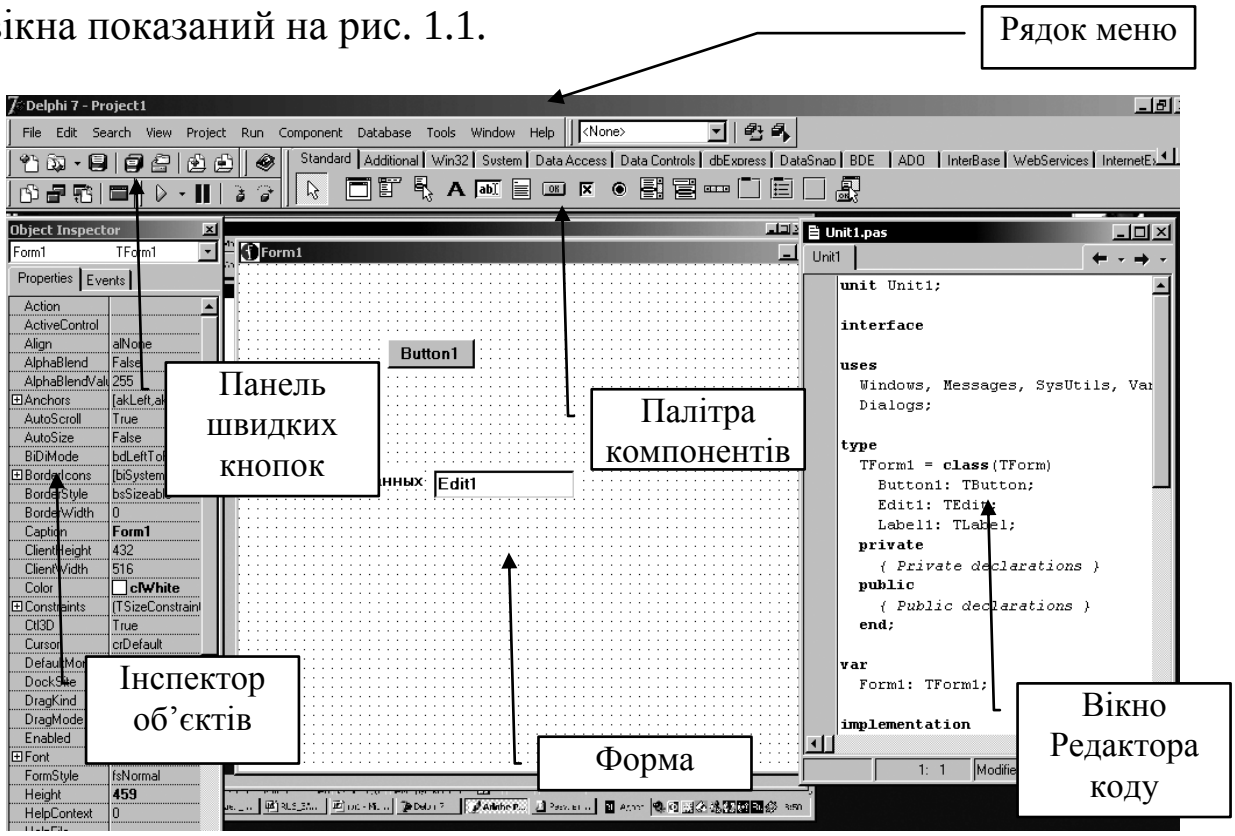


Рисунок 1.1 - Вікно середовища програмування Delphi

У верхній частині екрану розташований рядок головного меню. Нижче за рядок розміщено дві інструментальні панелі: *панель швидких кнопок*, дублюючих найбільш часто використовувані команди меню і *панель палітри компонентів*. Палітра компонентів складається з окремих панелей, в яких згруповані компоненти за певним принципом (**STANDARD** — найбільш часто використовувані компоненти, **SYSTEM** — системні компоненти: таймер, плейєр. і ряд інших панелей). Клацаючи лівою кнопкою миші по закладці панелі, користувач отримує в своє розпорядження компоненти, що знаходяться там.

Основою всіх додатків, що створюються в Windows є *форма*, На якій розміщуються різні елементи управління-об’єкти (кнопки, меню, перемикачі, списки, елементи введення і т. д.).

Щоб розмістити будь-який компонент на формі, слід знайти його на панелях, клацнути по ньому лівою кнопкою миші, включаючи його, як кнопку, а потім помістивши курсор в потрібне місце форми знов клацнути лівою кнопкою миші. Він розміститься там, де було проведено клацання. Після цього можна перетягувати його, змінювати розміри, як це робиться з вікнами в Windows або з намальованими об'єктами в Microsoft Office.

У режимі проектування маніпулювання властивостями здійснюється за допомогою вікна *Інспектора Об'єктів,* сторінка властивостей (Properties) якого показує властивості того об'єкту, який в даний момент виділений вами. Клацнувши на вікні порожньої форми і на сторінці властивостей Інспектора Об'єктів можна побачити властивості форми. Ці властивості можна змінювати. Наприклад, змінивши властивість **Caption** (напис) вашої форми, написав в ньому потрібний заголовок, і ви побачите, що цей напис з'явиться в смузі заголовка вашої форми. Існує декілька типів властивостей (рис.1.2).

Прості властивості – це ті, значення яких є числає або строки. Наприклад, властивості **Left** і **Top** приймають цілі значення, що визначають координати лівого верхнього кута объекту. Властивості **Caption** і **Name** представляють собою строки і визначають заголовок і ім’я объекту.

Наведені властивості - ті, які можуть прийимати значення зі списку. Простіший приклад – кнопка може бути активною або неактивною.

Вкладені властивості - це ті, які підтримують вкладені значення (або об’єкти). **Object Inspector** зображає знак “+” ліворуч від назви цих властивостей.

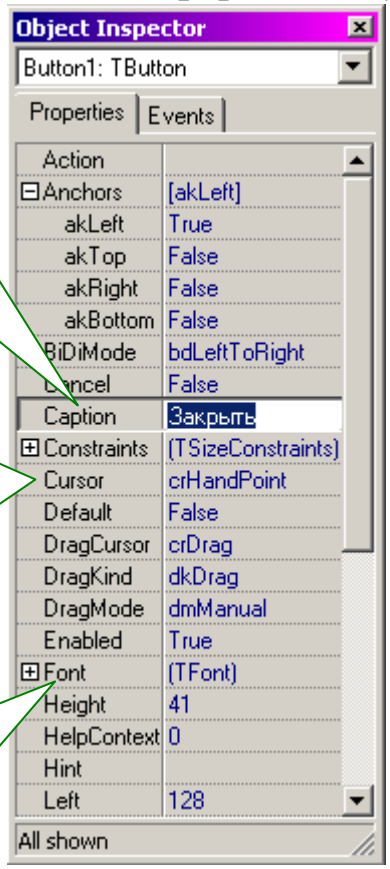


Рисунок 1.2 – Зовнішній вид вікна Інспектора Об'єктів інтегрованого середовища розробки Delphi

Наприклад, для того, щоб змінити властивість **Height** (висоту) і **Width** (ширину) кнопки, досить "зачепити" мишкою будь-який її кут і розсунути до потрібного розміру. Такий же результат може бути досягнутий простій підстановкою нового значення властивостей **Height** і **Width** у вікні Інспектора Об'єктів.

У табл. 1.1 приводяться основні властивості компонентів Delphi.

Таблиця 1.1 - Властивості компонентів Delphi

Властивості	Коментарі
Autosize	Висота елементу автоматично адаптується до розміру символів тексту - ***true***; ***false*** - не адаптується.
Caption	Рядок тексту, напис на кнопці, мітці, заголовок форми.
Color	Колір фону компоненту
Cursor	Визначає вид курсора миші при попаданні на об'єкт.
Enabled	Визначає, чи реагує компонент на події, пов'язані з мишею, клавіатурою, таймером.
Font	Задає властивості шрифту.
Height	Визначає висоту компоненту
Hint	Визначає текст підказки
Left	Координата лівого краю компоненти в пікселях
ShowHint	Визначає, показувати вікно підказки чи ні (***true-показывать*** підказку; ***false*** – не показувати)
Top	Координата верхнього краю компоненти в пікселях
Width	Визначає ширину компоненту
WindowState	Визначає розмір вікна форми

Однією з найбільш важливих частин середовища Delphi є *Редактор коди*. У його вікні створюється і редагується складений по спеціальних правилах текст програми, яка описує алгоритм роботи і виконує задану послідовність дій.

Коли створювана програма відкомпілюється і запущена, форма перетвориться на звичайне вікно Windows і почне виконувати ті дії, які для неї визначені. Таких вікон в програмі може бути скільки завгодно.

Ознайомимося з принципами роботи Delphi на прикладі простого застосування.

1. Відкрийте сторінку **ADDITIONAL**. Знайдіть компонент **Shape**  і перенесіть його на порожню форму. Для цього клацніть

по ньому лівою кнопкою миші, а потім помістите курсор в потрібному місці форми.

2. Визначите властивості об'єкту, використовуючи Інспектор об'єктів. Властивості багато, задамо тільки найнеобхідніші. Властивість **Shape** задає зовнішній вигляд компоненту, це може бути круг (stCircle), прямокутник (stRectangle), еліпс (stEllipse) або фігури із закругленими кінцями, тоді в назву властивостей додається слово **Round**. Задайте на свій розсуд висоту, ширину і колір об'єкту, їх англійські назви приведені в таблиці 1.1.

3. Відкрийте сторінку **STANDARD** і виберіть кнопку **Button**. Для цього об'єкту задамо властивість **Caption**, що визначає напис, допустимий «Круг», в рядку **Hint** вкажемо підказку «Натисни на кнопку для отримання круга» і задамо ShowHint=true, щоб підказка висвічувалася при попаданні курсора на кнопку.

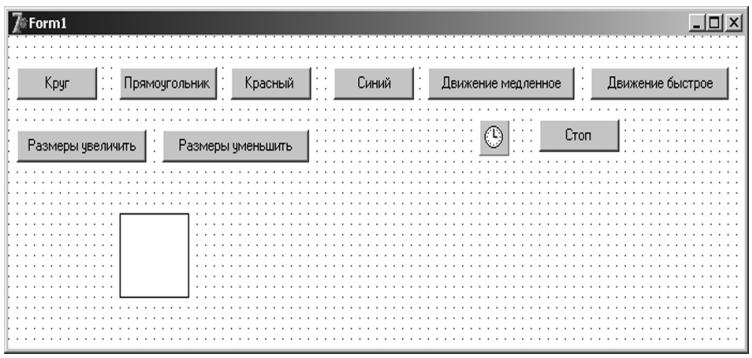


Рисунок 1.2 – Загальний вид створеної форми

4. Задайте аналогічні кнопки, керуючись наведеним на рис.1.2. На ньому розташовані наступнв кнопки: кнопка «Круг», яка дозволяє змінити форму фігури, перетворивши її на коло; кнопка «Прямокутник», що перетворює фігуру в прямокутник; кнопки «Червоний» і «Синій», які змінюють колір; кнопки «Рух у; кнопка «Стоп» зупинює об’єкт і дві кнопки, які задають зміни об’єкта.

5. Оскільки ми припускаємо рух фігури, то на форму поміщається **Timer** (сторінка **SYSTEM**). Цей об'єкт, хоча і виноситься на форму, на екрані невидимий, тому поміщати його можна в будь-якому місці. Властивість **Interval** задає час в мілісекундах. Чим більше інтервал, тим повільніше рухається об'єкт. Властивість **Enabled** задає у разі установки *True* початок відліку часу і, відповідно, початок руху об'єкту, при значенні рівному *False* – зупинку таймера.

Тепер, коли всі об'єкти розміщені на формі, перша дія, яку хотілося б порекомендувати б вам виконати після створення проекту, — зберегти його.

Якщо ви починаєте роботу з того, що зберігаєте проект, і надалі регулярно повторюєте збереження, ви можете не побоюватися будь-яких несподіванок типу збою комп'ютера або Delphi, викликаних технічними причинами або неприпустимими діями вашого власного ще не відладженого застосування при його запуску. Але є і ще аргумент на користь негайного збереження проекту і модулів. У багатовіконних застосуваннях, з якими ви зустрінетеся не раз, це дозволяє відразу задати модулям імена, які використовуватимуться в програмі для взаємних посилань модулів один на одного. Якщо ви не виконали відразу збереження форми, то вимушені спочатку посилатися на її ім'я за умовчанням, а надалі змінювати ці посилання.

Після того, як ви створили додаток з порожньою формою, відразу збережіть його в потрібній теці. І протягом роботи над проектом почастіше виконуйте збереження.

Зберегти проект можна командою File | Save All. Зручно також використовувати відповідну швидку кнопку. При першому збереженні Delphi запитає у вас ім'я файлу модуля, що зберігається, а потім - ім'я файлу проекту.

Будь-який проект має, принаймні, шість файлів, пов'язаних з ним.

❖ Головний файл проекту, спочатку називається **PROJECT1.DPR**. Введіть свою назву, при цьому дозволяється використовувати прописні і рядкові латинські букви і цифри, при цьому на першому місці в імені файлу повинна стояти буква.

❖ Модуль програми |Unit|, зберігається як **UNIT1.PAS**, але його можна назвати будь-яким іншим ім'ям, наприклад, якщо файл з розширенням DPR ми назвали gr.dpr, то другий файл назвемо gr1.pas, тоді при сортуванні на ім'я всіх файлів, що відносяться до одного проекту будуть розташовані поряд, що полегшить їх пошук, наприклад, для копіювання на дискету.

❖ Файл головної форми з розширенням ***.DFM**, використовується для збереження інформації про зовнішній вигляд головної форми.

❖ Файл з розширенням ***.RES** містить ікону для проекту і створюється автоматично.

❖ Файл з розширенням ***.OPT** є текстовим файлом для збереження установок, пов'язаних з даним проектом.

❖ Файл з розширенням ***.DSK** містить інформацію про стан робочого простору.

Після компіляції програми з'являється файл з розширенням ***.EXE** - виконуваний файл.

Заведіть собі за правило відводити для кожного нового проекту нову теку Windows.

Зручна структура тек істотно полегшує роботу над проектами.

Зв'язана ця рада з тим, що якщо поміщати декілька проектів в один каталог, то і ви, і, можливо, Delphi скоро заплутаєтеся в тому, які файли до якого проекту відносяться. Розміщення проектів в різних теках позбавить вас надалі від багатьох неприємностей.

Якщо у вас виникає декілька варіантів виконання проекту, а зазвичай так і буває, то бажано усередині теки проекту створювати підкаталоги для кожного варіанту. Зручна структура каталогів істотно полегшує роботу над серйозними проектами.

Працюючи над програмою, програміст, особливо початківець, повинен добре уявляти, що програма, яку він розробляє, призначена, з одного боку, для користувача, з іншої — для самого програміста. Тому для того, щоб робота була ефективною, програма повинна бути легко читаною, її структура повинна відповідати структурі і алгоритму вирішуваного завдання. Як цього добитися? Треба слідувати правилам хорошого стилю програмування - набору правил, яким слідує програміст (усвідомлено або тому, що "так роблять інші") в процесі своєї роботи. Хороший стиль програмування припускає:

- використання коментарів;
- використання такихімен змінних, процедур і функцій, що несуть смислове навантаження ;
- використання відступів;
- використання порожніх рядків.

Слідування правилам хорошого стилю програмування значно зменшує вірогідність появи помилок на етапі набору тексту, робить програму легко читаною, що, у свою чергу, полегшує процеси відладки і внесення змін.

7. Тепер після завдання зовнішнього виду форми, слід написати текст програми, щоб визначити виконувані при натискуванні на кожну кнопку дії. Натиснемо двічі по кнопці «Коло», автоматично з'явиться скомпільований текст процедури.

```
unit f12; ---------- модуль починається з ключового слова Unit, після
                     якого слідує ім'я файлу.
        interface
        uses  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,  Dialogs, ExtCtrls, StdCtrls;  ------- список підключених
модулів
        type  TForm1 = class(TForm)
        Button1: TButton;
        Shape1: TShape;
        Button2: TButton;
        Button3: TButton;
        Button4: TButton;
        Timer1: TTimer;
        Button5: TButton;
        Button6: TButton;
        Button7: TButton;
        Button8: TButton;
        Button9: TButton;
procedure   Button1Click(Sender:TObject);   ---  список реалізованих
                                                 процедур
private { Private declarations } --опис змінних, функцій і про-цедур,
                                   доступних тільки в цьому модулі
public { Public declarations } ---опис змінних, функцій і про-цедур,
                                   доступних в інших модулях
        end;
        var  Form1: TForm1;
        implementation
        {$R *.dfm}
        procedure TForm1.Button1Click(Sender: Tobject);
        begin
shape1. shape:=stCircle;  --{цей оператор задає зміну зовнішнього виду
                              компоненту Shape}
        end;
```

Список компонент, які розміщені на формі

У фігурних дужках тут і далі приводитимуться коментарі для вас. Оператор завдання виду фігури складається з її назви (shape1) і властивості фігури, яка змінюватиметься (shape). Знак **:=** є ознакою оператора привласнення. Таким чином, можна сказати, що властивості **Shape** об'єкту **Shape1** привласнюється найменування stCircle. В кінці кожного оператора ставлять крапку з комою. Звернете увагу, що при створенні програми, окрім оператора **Shape1.shape:=stCircle;** не вводиться нічого, все компілюється середовищем Delphi автоматично. Для зміни виду фігури необхідно вказати ім'я об'єкту, який видозмінюється, в нашому випадку це **Shape1**. Зовнішній вигляд визначає властивість **Shape**. Відкривши її, можна побачити набір можливих модифікацій фігури. Вкажіть потрібну .

8. При натискуванні по кнопці «Прямокутник» фігура повинна в нього перетворитися. Двічі клацніть по кнопці з аналогічною назвою, з'явиться скомпільований текст програми і курсор миготітиме між словами ***begin*** і ***end***, куди і слід ввести оператора **shape1. shape:=stRectangle;**

```
procedure TForm1.Button2Click(Sender: TObject);
begin
Тут миготить курсор і сюди вводиться текст програми!
end;
```

Для задання зміни кольору розглянемо технологію їх створення. Комп'ютерні монітори відображають кольорові гамми за технологією RGB (**R**ed – красный; **G**reen—зелений; **B**lue—синій). Щоб подивитись, яким чином діє технологія RGB, відкрийте властивість **Color** форми. Натисніть двічі по кольору, що з'явився, відкриється діалогове вікно з набором кольорових прямокутників. Натисніть по кнопці «Визначити колір». Виберіть будь-який колір і подивіться на набір кольорів «Червоний», «Зелений» і «Синій» в правому нижньому кутку екрану (рис.1.4). Щоб отримати будь-які кольори, що відображаються на комп'ютері, ці кольори змішують. У вашому розпорядженні від 0 до 255 значень, вважайте літрів фарби. Наприклад, в рядку «червоний» вкажемо максимально можливу кількість —255, у останніх — по нулю. Колір зразка (правий нижній кут) зміниться на «справжній» червоний. Якщо вказувати максимальну інтенсивність для зеленого або синього, а решту кольорів не використовувати, ви отримаєте яскраво синій чи яскраво зелений. Це логічно, якщо взяти олівець або фломастер синього кольору і почати ним малювати, то це буде тільки синій колір.

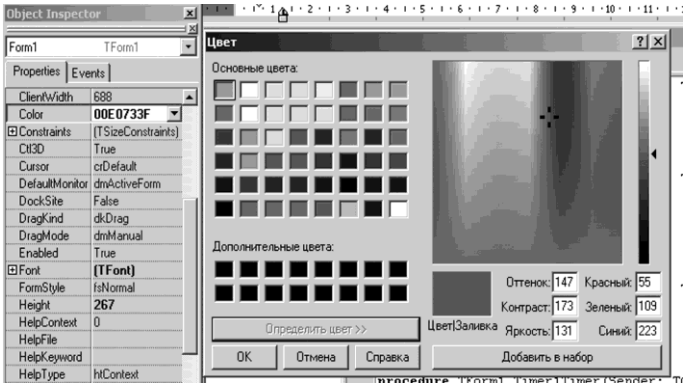


Рисунок 1.4 – Палітра кольорів

Взявши червоного і зеленого по 255, а синього взагалі (нуль), ви отримаєте чистий жовтий колір. Все по 128 дасть сірий, все по нулю — чорний, все по 255 дає білий колір.

Тепер подивимося, як задати зміну кольору в програмі. У об'єкта Shape є властивість **Brush**, яка у свою чергу розпадається на дві підвластивості: ***Color*** і ***Style***. Перша з них дозволяє задавати

колір, а друга – тип заливки, тобто штрихування. Оператор зміни кольору виглядає таким чином: **shape1.brush.Color:=rgb(255,0,0);** Це завдання червоного кольору об'єкту. Введіть такий самий оператор, але з іншим набором складових RGB при натискуванні по кнопці «Синій».

Задамо рух. На екран був поміщений компонент **Timer1**. Встановите спочатку властивість **Enabled** рівним ***false***, це означає, що таймер знаходиться в неактивному, неробочому стані. Тоді при натискуванні по кнопці «Рух повільний» включимо таймер і задамо велику тривалість інтервалу:

timer1.enabled:=true; timer1. interval:=500;.

Якщо об'єкт повинен рухатися зліва направо, це означає, що значення координати **Left** збільшуватиметься, оскільки збільшується відстань від лівого краю форми. Це задається оператором:

shape1.Left:= shape1.Left +5;

Цифра п'ять вибрана довільно. Вона встановлює, на скільки пікселів повинен зміщуватися об'єкт кожні 500 мілісекунд.

Швидкий рух об'єкту задається у виді :

timer1.enabled:=true; timer1. interval:=10;

тобто знову включається таймер, але інтервал значно менший, означає ті ж п'ять пікселів фігура повинна пройти не за 500, а за 10 мілісекунд.

Щоб зупинити рух, слід вимкнути таймер

timer1.enabled:=false;

Збільшення в два рази розмірів фігури по висоті і ширині відбувається по командах:

shape1.Width:=shape1.Width*2; shape1.height:= shape1.height*2;

Зменшення розмірів об'єкту задається за допомогою тих же операторів, але слід використовувати знаки «мінус» або «ділення».

4. Щоб додаток краще виглядав, можемо задати колір самої форми (властивість Color), змінити колір і зображення букв на кнопках (властивість Font).

Теперь, коли готова програмна частина додатку, збережемо його ще раз і запустимо. Для запуска використовується пункт меню **Run** (напроти нього вказана швидка клавиша **F9)**. Можно викоритовувати і трикутник, він синього (Delphi3) або зеленого (Delphi7) кольору (рис.1.5).

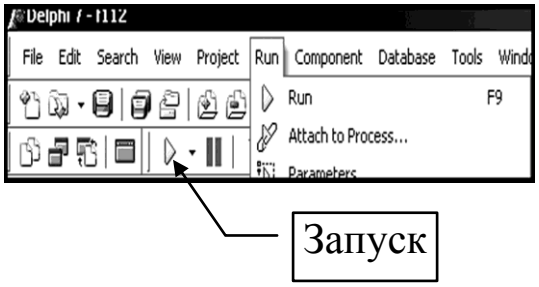


Рисунок 1.5 - Вибір пункту меню Run

Оскільки в програмі не був реалізований вихід, то скористайтеся стандартною кнопкою закриття вікна в Windows, якщо хочете повернутися в режим програмування.

Якщо компонент слід прибрати з екрану (тобто зробити його невидимим), використовують оператора

Shape1.visible:=false;

Якщо він знов повинен з'явитися, його властивість видимості рівна ***true*** і записується таким чином:

Shape1.visible:=true;

Іноді виникає відчуття, що програма «зависла». Це може бути, наприклад, унаслідок неправильного введення початкових даних. Виконаєте команду **Run|Program Reset** (виконання команди можна замінити використанням клавіш **Ctrl+F2**), відбудеться повернення в середу програмування.

Таблиця 1.2 - Основні команди меню, необхідні при роботі:

Найменування команди	Призначення команди
File (Файл).	Дозволяє відкривати, зберігати, закривати і друкувати проекти.
New	Дозволяє створити нове застосування (Application), додати нові складові (Form, Unit.) у вже існуючий проект.
Open	Аналог пункту меню Відкрити в будь-якому з додатків Office, з його допомогою можна відкрити вже розроблені застосування.
Reopen	Відкриває список останніх із збережених застосувань.
Save	Зберігає розроблені застосування.
Save All	Зберегти проект і всі його файли.
Close	Закриття проекту.
Print	Друк активних файлів.
Exit	Вихід з середовища Delphi.
Меню Edit (Правка)	Включає підміню редагування для роботи з текстом і компонентами.
Undelete	Відміняє попередню операцію. Аналогічно команді

	Правка/відмінити в Windows.
Redo	Відмінити попередню команду Undelete.
Cut	Вирізати.
Copy	Копіювати
Paste	Вставити.
Delete	Видалити.
Select All	Виділити всі компоненти активної форми або весь код програми.

Компонент Tform Хоча компонент TForm відсутній в палітрі компонентів, але все таки має властиві для нього властивості, події.

Итак, компонент TForm это окно в оконном приложении. Оно может быть минимум одно, максимальное ограничение на количество не накладывается. Первое созданное окно в приложении автоматически становится главным. При закрытии главного окна приложение завершает свою работу и, занимаемая им память, освобождается.

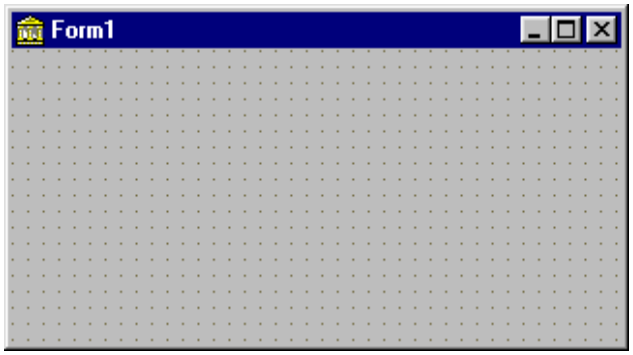


Рисунок 1.6 – Компонент TForm

Якщо необхідно, щоб головне вікно форми автоматично не з'являлося на екрані, то в події OnCreate для цього вікна, або в програмі DPR до команди створення головного вікна необхідно вказати наступний рядок:

```
Application.ShowMainForm:=false;
```

У цій команді указується, що в цьому застосуванні заборонено відображення головної форми (вікна). В цьому випадку програмістові потрібно самостійно викликати команду

```
Form1.Show;
```

для відображення цього вікна. Тут Form1 – головне вікно додатку.

Даний метод може виявитися корисним для виведення запиту пароля на запуск програми. Саме це діалогове вікно потрібно показати користувачеві перед показом головного вікна, але не роблячи його головним вікном, тобто при закритті вікна введення пароля, додаток не завершує свою роботу. І при введенні невірного пароля можна без проблем завершити роботу програми командою

```
Form1.Close;
```

Або дати команду завуршения роботи додатку.

Кожен компонент в програмі, як і сама форма, має унікальне ім'я, вказане у властивості Name. Заголовок вікна міститься у властивості Caption. По заголовку вікна користувач дізнається про функціональне призначення програми або поточного вікна.

Окрім заголовка у верхній частині вікна знаходиться ікона і кнопки управління станом. За умовчанням ікона така ж сама, як і ікона в проекті. Щоб змінити ікону в поточному вікні необхідно в інспекторові об'єктів вибрати властивість Icon в якому вибрати відповідний файл-малюнок. Файл повинен бути з розширенням ICO.

Для того, щоб змінити ікону в проекті, потрібно увійти до меню "Project", далі "Options...", на вкладці Application ви бачите поточний малюнок файлу проекту. Змінити його можна кнопкою "Load Icon...". Ікона проекту зберігається у файлі ресурсів з розширенням RES.

Властивість **BorderStyle**

bsDialog – Біля вікна немає ікони. Відображена тільки кнопка управління "Закрити". Розмір вікна постійний. Такий тип вікон застосовується найчастіше в діалогових вікнах, наприклад вікно запиту на збереження проекту, якщо ви намагаєтеся вийти з delphi не зробивши збереження.

bsNone – Біля вікна немає ікони, кнопок управління, заголовка. Розмір вікна постійний. Закрити таке вікно можна тільки програмно або за допомогою комбінації клавіш Alt+F4. Цей тип вікон застосовується в заставці при запуску програми. На вікні розташований компонент TImage, який містить малюнок.

bsSingle – У вікні присутня ікона, є заголовок. Кнопки управління скрутити, розвернути (відновити), закрити. Розмір вікна постійний.

bsSizeable – Тип вікна за умовчанням. Має всі елементи, вказані в попередньому значенні властивості плюс зміна розмірів вікна.

bsSizeToolWin – Цей тип вікна встановлений у вікні інспектора об'єктів. Таке вікно не має ікони, є заголовок, кнопка управління "закрити". Можна змінювати розмір вікна.

bsToolWindow – Аналогічний попередньому значенню, за винятком того, що не можна змінювати розміри вікна.

Всі вищезгадані типи властивості **BorderStyle** встановлюються тільки для працюючого застосування, тобто ми бачимо зміни в оформленні вікна тільки після запуску програми.

Властивість **BorderIcons**

Ця властивість є вкладеною. Тобто якщо ви в інспекторові об'єктів натиснете на знак +, то відкриється ще декілька підвластивостей. Властивість BorderIcons має сенс встановлювати тільки у відповідних значеннях властивості BorderStyle.

biSystemMemu – указує, відображати чи ні ікону і кнопки управління вікном.

biMinimize – відображати чи ні кнопку управління "скрутити".

biMaximize – відображати чи ні кнопку "розвернути" ("відновити").

biHelp – відображати чи ні кнопку "допомога".

Можливі два значення для цих властивостей: **true** (істинно) і **false** (помилково).

Властивість **Position**

poDefault – Windows само визначає розмір і положення вікна.

poDefaultPosOnly – довільне положення вікна.

poDefaultSizeOnly – тільки довільний розмір вікна.

poDesigned – такой-же розмір і положення, кокой був при розробці. Значення за умовчанням. Вам слід знати, що не на всіх комп'ютерах встановлений такий же екранний дозвіл, яке є у вас, і означає вікна в програмі знаходитимуться в абсолютно іншому місці.

poDesktopCenter – вікно знаходиться в центрі екрану.

poScreenCenter – вікно знаходиться в центрі екрану. Працює з багатодисплейним режимом відображення.

Властивість **WindowState**

wsMaximized – Вікно спочатку буде розгорнене у весь екран.

wsMinimized – Вікно спочатку буде згорнуто в значок.

wsNormal – Вікно має ті ж розміри, що і під час розробки. Значення за умовчанням.

Приклад

Розглянемо один приклад. Нам треба ще до запуску програми запропонувати користувачеві ввести якийсь пароль. Якщо пароль

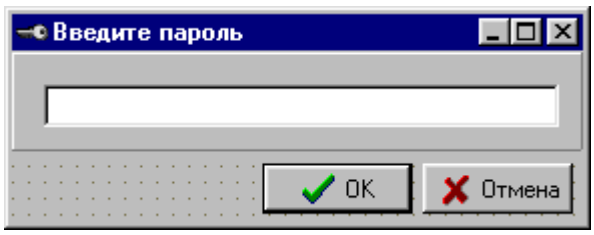
введений вірно, то відображається головне вікно проекту і користувач може працювати. Якщо пароль введений невірно, то програма негайно завершує свою роботу.

Запускаємо delphi і створюємо новий проект. У новому проекті вікно Form1 буде головним вікном проекту.

Для події створення вікна OnCreate пишемо:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Application.ShowMainForm:=false; //не відображати головне вікно додатку
end;
```

Вибираємо пункт меню "New Form" з меню "File". З'являється нове вікно Form2. Його ми використовуватимемо як введення пароля. Оформляємо його для цього належним чином.



Приклад ви бачите на рис.1.7.

Рисунок 1.7 – Форма для введення

· Зменшуємо вікно так, щоб воно за розмірами було подібне до вікон, що запитують пароль.

· Властивість *BorderStyle* встановлюємо в bsSingle. При цьому розмір вікна під час роботи програми буде постійний.

· Властивість *BorderIcons*. Підвластивості biMinimize і biMaximize встановлюємо в false. Це остаточно недопустит зміна стану вікна (згортання, розгортання на весь екран).

· У властивості *Caption* пишіть будь-яку на ваше розсуди зрозумілу фразу, наприклад "Введіть пароль".

· Властивість *Position* в значення poDesktopCenter. При цьому вікно при будь-якому екранному дозвіл завжди спочатку буде розташований посередині екрану.

· На сторінці палітри компонентів STANDART вибираємо і встановлюємо у форму компонент **TPanel**. Прибираємо у нього значення *Caption*, надаємо властивості *Align* значення alTop і остаточно змінюємо його розмір на четь більше половини вікна.

· На сторінці ADDITIONAL вибираємо компонент **TBitBtn** і встановлюємо у форму дві кнопки, нижче за панель.

· Для першої кнопки властивість Kind в bkOk, для другої *bkCancel*.

· Для першої кнопки властивість *Caption* залишаємо в для всіх зрозуміле значення Ok, для другої пишемо "Відміна".

· На сторінці STANDART вибираємо і встановлюємо на компонент Panel компонент TEdit.

 Для компонента **Edit1** змінюємо властивість PasswordChar на знак * (зірочка). Це прийнятый у всіх програмах символ введення пароля. При наборі паролю замість символів будутье відображатися зірочки.

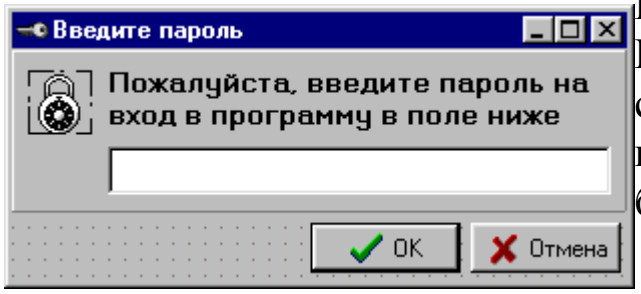


Рисунок 1.8 – Вікно ввденння паролю

· Для компоненту Edit1 прибираємо значення Caption.

· Для Form2 у властивості Icon підбираєте відповідну ікону.

Ми тільки що оформили повністю готове вікно для введення пароля. Я ж можу запропонувати вашому увагу свій варіант, який барвистіше оформлений. Ви завжди можете оформляти свої застосування по своєму. Не обов'язково дотримуватися яких-небудь правил по оформленню програми, але проте користувач не повинен втрачатися побачивши ваших "фантазій" і повинен чітко уявляти, що від нього вимагається.

Далі робимо виклик вікна введення пароля на екран. Оскільки ми з вами прибрали властивість виведення головного вікна, то після запуску програми на екрані не буде нічого. Нам же потрібно вивести вікно запиту пароля, і ми це робимо за допомогою програми DPR. Викликаємо її на екран за допомогою пункту меню "View Source" з меню "Project". У рядку до Application.Run пишемо:

```
Form2.Show;
```

Ще до запуску додатку, але після створення всіх вікон на екран виводиться вікно Form2.

Далі перемиканий редактор коди на модуль Unit2, викликаємо проектоване вікно клавішею F12, в інспекторові об'єктів для компоненту Form2 (проектована форма) створюємо реакцію на подію закриття вікна OnClose і в самій процедурі пишемо:

```
Procedure TForm2.FormClose (Sender: TObject; var Action: TCloseAction);
begin
If Form2.ModalResult=mrOk then //если користувач натиснув на кнопку Ok, то
  If Edit1.Text='Informatica' then //если набраний необхідний пароль, то
  begin
  Form1.Show; //показать головне вікно
  Exit; //выйти з цієї процедури
  end;
Application.Terminate; { якщо всі вищезгадані умови не виконуються, то зупинити програму}
end;
```

Для кнопки BitBtn1 реакція на подію OnClick

```
procedure TForm2.BitBtn1Click(Sender: TObject);
Begin
Form2.ModalResult:=mrOk; //результат роботи цього вікна
Close; //закрыть це вікно
end;
Для кнопки BitBtn2 подія OnClick
procedure TForm2.BitBtn2Click(Sender: TObject);
begin
Close; //закрыть це вікно
end;
```

Оскільки ми вікно Form2 викликаємо командою Show, а не ShowModal, те привласнення результату роботи вікна не приведе до автоматичного закриття і виклику процедури OnClose. Тому після цієї команди слідує команда закриття вікна Close.

Короткий опис роботи програми.

При створенні головного вікна, програма отримує вказівку про відсутність необхідності автоматичного відображення головного вікна після запуску.

Далі, після створення всіх вікон (у нашому випадку два) стає видимим вікно Form2. Якщо користувач натиснув на кнопку "Ok" або в полі введення пароля натиснув на клавішу Enter (!), то викликається обробник події OnClick, який занесе відповідне значення в змінну результату роботи цього вікна і викличе команду закриття Close.

У процедурі обробки закриття вікна спочатку йде перевірка результату, правильності введення пароля. Якщо одне і інша умова виконується, то поступає команда Form1.Show, яка виведе головне вікно, після якої команда Exit виведе обробник події з процедури, що тим самим виключає подальше виконання рядків цієї процедури.

Якщо по яких або причинам, всі вищезгадані умови не виконуються (користувач закрив вікно кнопкою "Відміна", закрив вікно кнопкою "Закрити" або натиснув комбінацію клавіш Alt+F4, ввів неправильний пароль), то виконується команда Termanate (терминатор, який вбиває працююче застосування :).

У програмі як пароль вводиться назва предмету, що вивчається. Якщо ви бажаєте вказати інший пароль, можна змінити відповідний рядок в програмі.

Наприкінці кожної теми ми виноситимемо основні положення, які викладені в її межах.

Основні положення теми:

✓ Для створення нового застосування виконується команда **File|New.**

✓ Для збереження - **File|Save All.**

✓ Для запуску на виконання **Run|Run**.

✓ Для перезавантаження програми команда **Run|Program Reset.**

✓ Компонент **Button** (кнопка) дозволяє задати виконання певних операцій при натисненні на неї. Основною властивістю компоненту є властивість Caption (текст на кнопці).

✓ Компонент **Shape** (фігура) використовується для приміщення на форму прямокутника, або квадрата, або круга або еліпса. Основною властивістю є властивість *Shape* (вид фігури), властивість *Brush* (задає колір і стиль штрихування).

✓ Компонент **Timer** (таймер) використовується для завдання процесів, що змінюються в часі, наприклад. Рух фігури, зміна її розмірів .Основными властивостями є властивість Enabled (включити або вимкнути таймер) і властивість Interval, задаюча швидкість процесів.

✓ Після створення додатку відразу збережіть його в потрібній теці. І протягом роботи над проектом частіше виконуйте збереження.

✓ Зручна структура тек істотно полегшує роботу над проектами.

✓ Для кожного нового проекту доцільно відводити нову теку Windows.

Тести для самоперевірки знань до теми1

Завдання 1–7 мають по чотири варіанти відповіді, серед яких лише ОДИН ПРАВИЛЬНИЙ. Виберіть правильний варіант відповіді.

1. Компонент **SHAPE**, використовуваний для введення об'єкту, належить до панелі кнопок:

А Win32
Б Additional
В Standart
Г System

Правильна відповідь: ☐

2. Компонент **BUTTON**, призначений для формування події при натисненні на кнопку, належить до панелі кнопок:

А Win32
Б Additional
В Standart
Г System

Правильна відповідь: ☐

3. Для представлення зовнішнього вигляду компоненту **SHAPE** у вигляді фігури із закругленими кінцями в назві властивості додаляється слово:

А Round
Б Circle
В Ellipce
Г Around

Правильна відповідь: ☐

4. Для завдання процесів, що змінюються в часі, використовується компонент... (введіть відповідь на англ.)

А TrackBar
Б ListBox
В Memo
Г Timer

Правильна відповідь: ☐

5. Колір і стиль штрифовки компоненту **SHAPE** визначає властивість:

А Font
Б Shape
В Style
Г Brush

Правильна відповідь: ☐

6. Одиницею вимірювання інтервалу зміни властивостей компоненту **TIMER** є :

А Хвилина
Б Секунда
В Миллисекунда
Г Година

Правильна відповідь ☐

7. Вкажіть оператор, який одночасно задає найшвидший та найбільш плавний рух об'єкту **Shape1** при включеному таймері:

А timer1.interval:=10;shape1.left:=shape1.left+10
Б timer1.interval:=10;shape1.left:=shape1.left+20
В timer1.interval:=200;shape1.left:=shape1.left+10
Г timer1.interval:=20;shape1.left:=shape1.left+50

Правильна відповідь ☐

Завдання 8–9 мають два варіанти відповіді. Треба обрати ПРАВИЛЬНИЙ, щоб приведений вислыв мав сенс.

8. Програми в Delphi пишуть на мові Object Pascal, що є подальшим об'єктно-орієнтованим розвитком широко поширеної мови Turbo Pascal.

Так | ні.

Правильна відповідь: ☐

9. Кожен компонент обов'язково видимий під час запуску програми

Так |ні

Правильна відповідь: ☐

Завдання 10–14 мають на меті встановлення відповідності. До кожного рядка, позначеного ЦИФРОЮ, доберіть відповідник, позначений БУКВОЮ, і поставте позначки

10. Установіть відповідність між основними командами і пунктами меню для їх виконання.

1 Для збереження додатку	А. Run|Run
2 Для створення нового додатку	Б. File|Run
3 Для запуску додатку на виконання	В. File|New
4 Для перезавантаження програми	Г. File|RrogramReset
	Д. File|Save all

	А	Б	В	Г	Д
1					
2					
3					
4					

11. Встановіть відповідність значень властивості Color за технологією RGB

1 rgb(0,0,255)	А. Синій
2 rgb(0,255,0)	Б. Червоний
3 rgb(255,0,0)	В. Зелений
4 rgb(0,0,0)	Г. Білий
	Д. Чорний

	А	Б	В	Г	Д
1					
2					
3					
4					

12. Встановіть відповідність між деякими властивостями компоненті їх розтлумачення

1 Caption	А. Ширина компоненту
2 Font	Б. Різновиди шрифту
3 Hint	В. Заголовок об'єкту, форми
4 Height	Г. Текст підказки
	Д. Висота компоненту

	А	Б	В	Г	Д
1					
2					
3					
4					

13 . Встановите відповідність властивостей об'єкту Shape, що змінюються.

1 shape1.width:=shape1.width*0.5	А.Збільшення висоти в 2 рази
2 shape1.left:=shape1.left+15	Б.Рух об'єкту справа наліво
3 shape1.left:=shape1.left-5	В.Зменшення ширини в 2 рази
4 shape1.top:=shape1.top-15	Г. Рух об'єкту знизу уверх
	Д.Рух об'єкту зліва направо

	А	Б	В	Г	Д
1					
2					
3					
4					

14. Встановите відповідність між компонентами та розтлумаченням їх

1	BUTTON
2	LABEL
3	SHAPE
4	TIMER

А. Кнопка
Б. Фігура
В. Таймер
Г. Мітка
Д.Список

	А	Б	В	Г	Д
1					
2					
3					
5					

Питання для самоконтролю

1. Яка команда виконується для створення нового проекту?
2. Яка команда виконується для запуску на виконання тільки-но створеного проекту?
3. Яким чином правильно виконати збереження створеного проекту?
4. Кратко охарактеризуйте наступні візуальні компоненти: Button, Shape,Timer.
5. Які основні властивості компонентів Button, Shape,Timer?
6. Яким чином визначити форму, яку набуває компонент Shape?
7. Охарактеризуйте дві підвластивості властивості Brush компоненту Shape.

2 РОЗРОБКА ПРОГРАМИ ЛІНІЙНОГО ОБЧИСЛЮВАЛЬНОГО ПРОЦЕСУ В СЕРЕДОВИЩІ DELPHI

Лінійний алгоритм - це такий процес, в якому всі операції виконуються послідовно одна за одною.

Постановка завдання:

Початкові дані : *a, b-дробные* числа; *c, d* – целые.

Визначити: $x:=sin(a+2\cdot b);$

$y:=e^3+ (c+3\cdot d)^{0.5;}$

$z:= (x+y)^2$ і вивести значення на екран.

Розташуємо на формі компоненти, необхідні для роботи. Цифрою 1 позначимо компонент **LABEL**, який використовували для ввода пояснювальних надписів (a=, b=…). Для ввода *a, b, c* используем компонент **Edit** (цифра2). Значення змієнної *a* буде вводитися в Edit1, значення *b* - в Edit2 і *c* -в Edit3 (рис.2.1). Цифрою 3 – кнопка **BUTTON**, при натискуванні по якій почнуться підрахунки, 4 – компонент **PANEL** для виведення результатів.

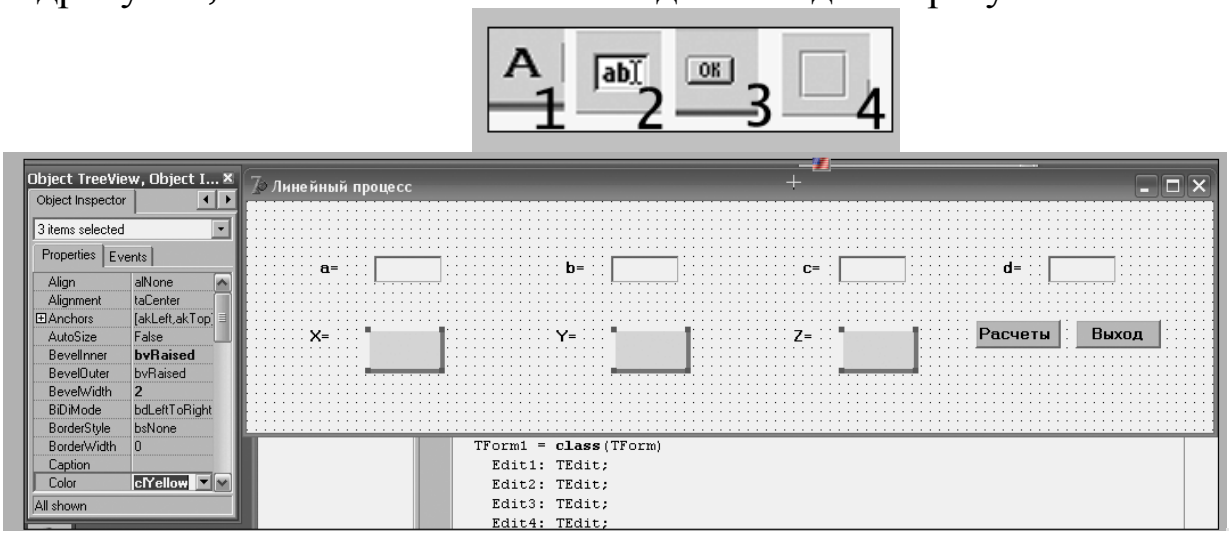


Рисунок 2.1 – Форма с необхідними компонентами

EDIT - компонент, який використовується для введення/виводу символьного терміну. Основною властивістю є властивість Text, спочатку в цьому вікні відображається ім'я компоненту, якщо вікно при введенні інформації повинне бути порожнім, його просто стирають; якщо є стартові початкові дані, то їх можна ввести в цій властивості.

Для зручності роботи з програмою, введемо пояснюючі написи. Для цього використовують компонент **Label.**

LABEL – компонент, призначений для відображення текстової інформації. Інформація записується у властивості Caption (у нашому прикладі це написи виду a=; b=; c= та інші).

Для введення інформації можна використовувати компонент **Trackbar**. Він належить до панеди кнопок Win32. Це елемент управління з повзунком, який користувач може переміщати за допомогою миші. До його основних властивостей відносяться:

Min, Max – мінімально і максимально допустимі значення, в яких змінюється інформація, що відображається на повзунку.

Orientation- визначають горизонтальну (вибрати із списку *trHorizontal*) або вертикальну (*trVertical*) орієнтацію повзунка.

Frequency – частота відображення рисок на шкалі.

TickMarks- положення шкали (*tmBoth* – снизу і зверху; *tmBottomRight* внизу повзунка; *tmTopLeft* – сверху).

Position – основна програмована властивість, що визначає позицію розташування повзунка.

Для виведення інформації в цій програмі використовуватимемо компонент **Panel.**

PANEL - компонент, який використовується для виведення результатів. Основною для нього є властивість Caption, яка і використовується для виведення даних.

Окрім цих об'єктів на екран поміщена вже відома вам кнопка Button з написом «Розрахунки».

Оскільки поставлене завдання припускає операції з числами, насамперед слід організувати введення інформації. У середовища Delphi є важлива властивість: вся інформація, що вводиться, сприймається як текстова, навіть при введенні чисел. Для проведення розрахунків інформацію, що спочатку вводиться, перетворюють в числа. Проводять розрахунки, а потім отримані результати перетворять в текстову інформацію для виводу на екран.

Двічі натиснемом по кнопці «Розрахунки» для компіляції процедури. Це приведе до генерації тексту

```
procedure TForm1.Button1Click(Sender: TObject);
begin
....
end;
```

Після **begin** введемо текст програми:

```
a:=strtofloat(edit1.text);
b:=strtofloat(edit2.text);
c:=strtoint(edit3.text).
```

Функція **StrToFloat** використовується для перетворення рядка в число з плаваючою комою (дріб), **StrToInt** – аналогічно, але в ціле.

Оператор **a:=strtofloat(edit1.text)** показує, що значення змінної a буде зчитано з компоненту Edit1 (як ви пам'ятаєте, введення даних відбувається за рахунок властивості text). Перетворено в дробове число.

Для прочитування інформації з компоненту **TrackBar** клацнемо по ньому двічі. Автоматично скомпілюється процедура TrackBar1Change. При зміні позиції повзунка в змінну d прочитується значення цієї позиції. Нами організований вивід на екран показника повзунка. Для цього використана мітка Label і оператор **label7.Caption:=inttostr(d)**.

Функція **IntToStr** виконує перетворення цілого числа в його текстовий еквівалент.

```
procedure TForm1. TrackBar1Change(Sender: TObject);
begin
        d:= TrackBar1.position;
        label7.Caption:=inttostr(d);
end;
```

Для проведення розрахунків дописуємо текст procedure TForm1.Button1Click. Для розрахунків використовують оператора привласнення, який має формат

Змінна:=вираз;

де в лівій частині указується ім'я змінної, яка обчислюється;

справа – вираз для виконання обчислень.

Delphi дозволяє використовувати шість арифметичних операцій:

+ - складання; **-** віднімання; ***** множення; **/** ділення; **div** - цілочисельне ділення; **mod** - залишок від цілочисельного ділення. Окрім цього використовується парна кількість круглих дужок () і стандартні математичні функції.

У табл.2.1 приведені найбільш часто використовувані математичні функції.

Величина кута тригонометричних функцій повинна бути виражена в радіанах. Для перетворення величини кута з градусів в радіани використовується формула $\frac{a \cdot Pi}{180}$ де: a - величина кута в градусах; 3.1415926 - число π.

Запрограмовані формули виглядають таким чином:

```
x:=sin(a+2*b);
y:=exp(3)+sqrt(c+3*d)
z:=sqr(x+y);
```

Таблиця 2.1 - Найбільш часто використовувані математичні функції

Функція	Значення	Функція	Значення
Abs (n)	Модуль числа n	Ln(n)	Натуральний логарифм n
Sqrt (n)	Квадратний корінь з n	Trunc(n)	Ціла частина дробового числа, отримана після відсікання дробової частини
Sqr (n)	Квадрат n		
Sin (n)	Синус n	Round (n)	Ціла частина дробового числа, отримана після округлення *n*
Cos (n)	Косинус		
Arctan (n)	Арктангенс n	Rardom(n)	Випадкове ціле число в діапазоні від 0 до n-1 (перед зверненням до неї необхідно викликати функцію ***Randomize***, яка виконає ініціалізацію програмного генератора випадкових чисел)
Exp(n)	Експонента n		
Exp(b*ln(a))	Значення ab		
Pi	Значення π=3.1415926		

При програмуванні формули слід пам'ятати, що спочатку виконуються всі розрахунки функцій, потім операції множення і ділення в порядку їх проходження зліва направо, і тільки потім складання і віднімання в тому ж порядку зліва направо. Це комп'ютерний принцип **пріоритету операцій**. Щоб краще з'ясувати поняття пріоритету операцій, давайте розглянемо простий математичний вираз 2*2+32-5+9. Пригадаємо шкільні роки і спробуємо провести обчислення. Що ми робимо на початку? Умножаємо два на два і запам'ятовуємо, що результат рівний чотирьом. Потім зведемо в квадрат три, отримаємо дев'ять. Формула прийме вигляд 4+9-5+9. А далі послідовно 4+9=13, потім віднімемо з тринадцяти п'ять 13-5=8 і в кінці додамо 9. Результат рівний 17. В принципі ми зараз працювали як комп'ютер, а, якщо точніше, то комп'ютер зробили під нас. Спочатку були знайдені дві старші операції: піднесення до ступеня і множення, ми порахували і запам'ятали результат або записали його на листі паперу, потім

зліва направо проводили всі операції складання і віднімання, підряд, в тому порядку, в якому вони слідують у формулі.

Круглі дужки, як і в математиці, змінюють пріоритет операцій.

Тепер потрібно організувати вивід на екран в компоненти **Panel** і компонент **TrackBar**. Вивід на екран можна організувати декількома способами. *Перший* спосіб полягає у використанні функції **FloatToStr** (перетворення дробового числа в рядок і має вигляд

panel2.caption:=floattostr(z); - вивід в компонент Panel. Кількість знаків, що виводяться після коми рівне 11.

Другий спосіб полягає у використанні функції **str**. Функція має формат:

str([число]: [кількість символів, що виводяться]: [з них після коми], [ім'я рядка, в яке число перетвориться])**.**

Цей варіант дозволяє видавати на екран число із заданим ступенем точності.

```
str(x:5:2,ss);
panel1.caption:=ss;
```

Третій спосіб – вивід в компонент **trackbar1**.

```
trackbar1.position:=trunc(y);
label11.Caption:=floattostr(trunc(y));
```

Для виводу необхідне перетворення числа з відкиданням його дробової частини. У компонент **label11** виводиться набутого значення для відображення його величини на екрані.

Змінними називаються параметри програми, значення яких можуть змінюватися в процесі її виконання. Всі використовувані в програмі змінні повинні бути ідентифіковані й оголошені з вказівкою їх типів. У розділі **Var** конкретної процедури описуються змінні, які будуть використані тільки в ній.

Дійсні змінні (числа з плаваючою крапкою) оголошуються за допомогою службового слова *Real*, цілі - за допомогою службового слова *Integer*. Оголосити змінну, це означати зарезервувати в пам'яті машини елемент пам'яті для зберігання будь-якого числа. Звернутися до цього осередку можна по імені (ідентифікатору), яке вибране вами в розділі оголошення змінних.

У табл2.2 приведені варіанти опису даних. У нашому випадку опис має вигляд:

```
var ss:string;  a,b,x,y,z:real;   c:integer;
```

Змінна *d* використовується в двох процедурах. Її опис поміщають в розділ

public d:integer;

Табліця 2.2 - Варіанти опису типу даних

Тип	Розмір у байтах	Діапазон значень	Тип	Розмір в байтах	Діапазон значень
Byte	1	0÷255	Real	$\pm 5.0 \cdot 10^{-324} \div 1.7 \cdot 10^{308}$	8
Word	2	0÷65535	Single	$\pm 1.5 \cdot 10^{-45} \div 3.4 \cdot 10^{38}$	4
LongWord	4	0÷4294967295	Double	$\pm 5.0 \cdot 10^{-324} \div 1.7\ 10^{308}$	8
ShortInt	1	-128÷127	Char	1 символ	1
SmallInt	2	-32768÷32767	String	Максимальная довжина 255 символів	
Integer	4	-2147483648 ÷2147483647			

Лістинг процедури, яка відбувається при клацанні по кнопці **«Розрахунки»,** має наступний вигляд:

```
procedure TForm1.Button1Click(Sender: Tobject);
var ss:string;  a,b,x,y,z:real;  c:integer;
begin
      a:=strtofloat(edit1.text);
      b:=strtofloat(edit2.text);
      c:=strtoint(edit3.text);
      x:=sin(a+2*b);
      y:=exp(3)+sqrt(c+3*d);
      z:=sqr(x+y);
        str(x:5:2,ss);
        panel1.caption:=ss;
        trackbar1.position:=trunc(y);
        label11.Caption:=floattostr(trunc(y));
        panel2.caption:=floattostr(z);
end;
```

Лістинг процедури, яка відбувається при переміщенні повзунка, має наступний вигляд:

```
  procedure TForm1.sChange(Sender: TObject);
  begin
d:= s.position;
       label7.Caption:=inttostr(d);
  end;
```

На рис.2.2 приводиться копія екрану, на якому демонструється робота програми.

Зверніть увагу на значення функції *z*. Якщо немає необхідності в такій великій кількості знаків після коми, то функцію **FloatToStr** не використовують для виводу інформації.

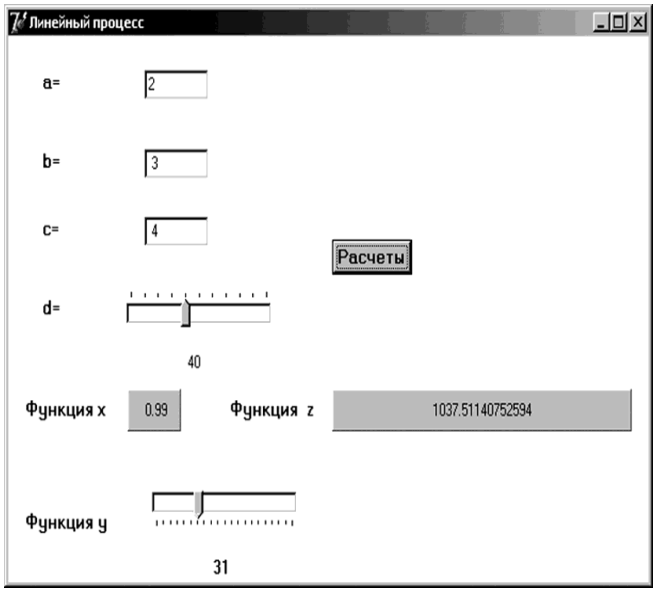


Рисунок 2.2 – Зовнішній вид форми

На практиці рідко зустрічаються завдання, алгоритм вирішення яких є лінійним.

Точки алгоритму, в яких виконується вибір подальшого ходу програми, називаються точками вибору. Вибір кроку рішення задачі здійснюється залежно від виконання деякої умови. У повсякденному житті умова зазвичай формулюється у вигляді питання, на яке можна відповісти **та** чи **ні.** Наприклад:

- Величина дискримінанта позитивна ?
- Відповідь правильна?
- Сума покупки більше 500 гривень?

У програмі умова — це вираз логічного типу, який може приймати одне з двох значень: **True** (істина) або **False** (брехня).

У мові Delphi є шість *операторів порівняння* (>більше < менше = рівно, <> не рівне, >= більше або рівно, <= менше або рівно).

З простих умов за допомогою *логічних операторів*: **and** — «логічне І», **or** – «логічне АБО» і **not** – «ні» можна будувати складні умови.

Оператор **if** дозволяє вибрати один з двох можливих варіантів розвитку програми. Вибір здійснюється залежно від виконання умови. У загальному вигляді Оператор **if** записується так:

if умова **then** оператор1 **else** оператор2**.**

Якщо умова **виконується**, то діє оператор 1 **інакше** оператор 2.

Наприклад, необхідно запрограмувати наступну умову:

$$z = \begin{cases} \dfrac{a\sin(x)}{x} & x \le 0 \\ \dfrac{x}{(x-6)} & x > 0 \end{cases}$$

If x<=0 then z:=a*sin(x)/x else z:=x/(x-6);

Залежно від введеного значення *X* розрахунок проводиться по першій формулі, якщо *X* менше нуля і по другій, якщо введене значення більше або рівно нулю.

За правилами мови після слів **then** або **else** можна ставити один тільки один будь-який оператор. Це може бути оператор привласнення, виводу на екран, введення даних, ще один умовний оператор і т.д. Якщо за умовою завдання після **then** або **else** потрібно поставити більше одного оператора, то для реалізації таких ситуацій використовують складеного оператора. Складений оператор складається з групи операторів, поміщених в операторні дужки **begin** і **end.**

Допустимі вкладені оператори **if**. Наприклад, виконувана последова-тельность дій залежно від кольору світлофора може бути запрограмована таким чином:

If svet=’зеленый’ **then** showmessage(‘переходимо вулицю’) **else if** svet=’желтый’ **then** showmessage(‘чекаємо’) **else** showmessage(‘стоїмо’);

Використаний в прикладі оператор **showmessage** використовується для виведення рядка тексту, причому рядок поміщається в одинарні лапки.

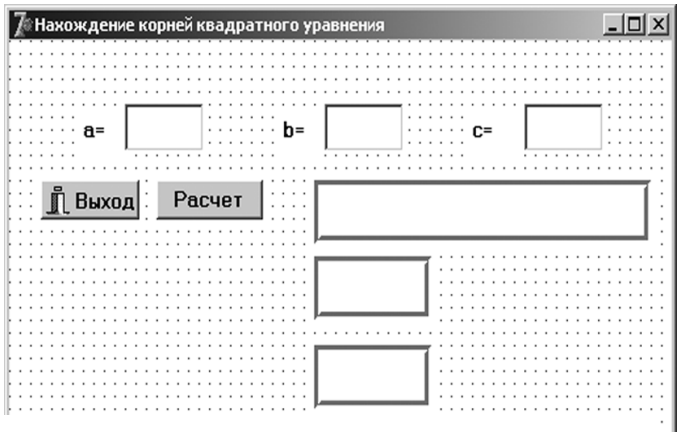


Рисунок 2.3 – Загальний вид форми для рішення квадратного рівняння

Розглянемо процедуру знаходження коренів квадратного рівняння, яке в загальному вигляді записується: $ax^2+bx+c=0$. Для введення даних використані три компоненти Edit. Для виведення результатів - три компоненти Panel (рис.2.3).

Текст програми виглядає таким чином:

```
procedure TForm1.Button1Click(Sender:TObject); {розрахунок проводиться при клацанні по кнопці «Розрахунок»}
  var ss:string;  a,b,c,d,x1,x2:real; {опис використовуваних в програмі змінних: a, b, з – коефіцієнти рівняння; d-дискримінант; x1,x2 – коріння рівняння }

begin
   a:=strtofloat(edit1.text); {Введення початкових даних}
   b:=strtofloat(edit2.text);
   c:=strtoint(edit3.text);
   d:=b*b-4*a*c;  {Розрахунок дискримінанта}
   str(d:1:2,ss);
      panel1.caption:='дискриминант ='+ss; {Виведення дискримінанта на екран, причому окрім цього виводиться текст; текстові змінні з'єднуються знаком плюс для виводу в один рядок}
     if d>0 then begin x1:=(-b+sqrt(d)) /2/a;  x2:=(-b-sqrt(d)) /2/a;
        {Якщо дискримінант більше нуля, визначаються обидва корені}
     panel2.Visible:=true;  panel3.Visible:=true; {Первинний обидва компоненти Panel на екрані не відображаються, оскільки невідомо, чи є у рівняння коріння, якщо вони є, то їх виводять на екран, для цього властивість компоненту Visible вважається рівним true }
      str(x1:1:2,ss);
      panel2.caption:='x1 ='+ss;    {Виведення першого кореня в Panel2}
      str(x2:1:2,ss);
      panel3.caption:='x2 ='+ss;    {Виведення другого  кореня в Panel3}
       end else
     if d=0    then begin x1:=-b/2/a;  str(x1:1:2,ss); {Якщо корінь один, то його обчислюють і виводять на екран, другий компонент Panel стає невидимим}
     panel2.Visible:=true;   panel3.Visible:=false;
     panel2.caption:=' Один корінь x ='+ss;
                  end  else  begin  panel2.Visible:=true; panel2.Width:=120; {При виведенні даних змінюється ширина компоненту Panel залежно від довжини тексту, що виводиться}
     panel2.Width:=180; panel3.Visible:=false; panel2.caption:=' Дійсного коріння немає'; {Якщо коріння немає, то це повідомлення виводиться на екран, при цьому міняється ширина компоненту Panel }
        end;      end;
```

На рис.2.4 приводяться екрани розрахунку коренів рівняння для від’ємного, нульового та додатнього дискримінантів.

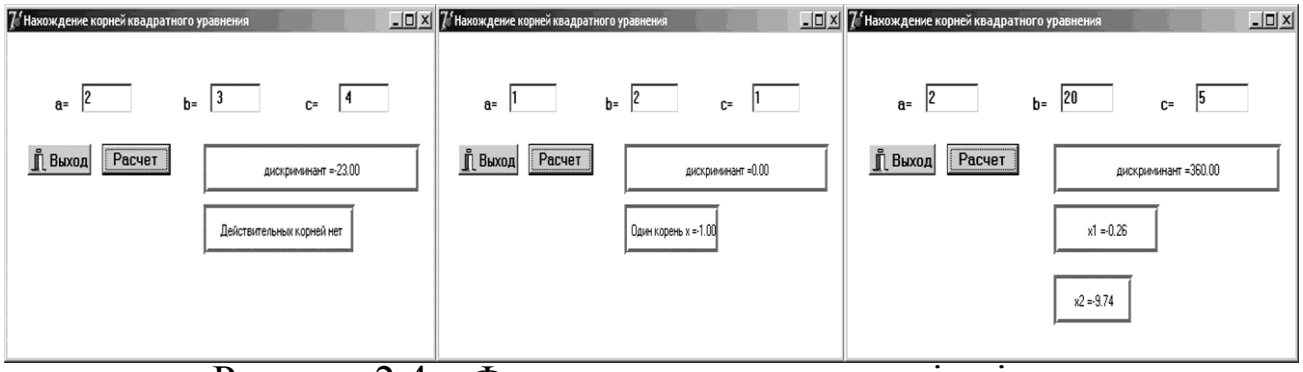


Рисунок 2.4 – Форми розрахунку коренів рівняння для від'ємного, нульового та додатнього

При записі складних умов важливо враховувати те, що логічні оператори мають вищий пріоритет, чим оператори порівняння, і тому прості умови слід брати в дужки. Наприклад, хай умова надання знижки сформульована таким чином: «Знижка надається, якщо сума покупки перевищує 400 крб. і день покупки — воскресіння», Якщо день тижня позначений як змінна Day цілого типу, і рівність її значення одиниці відповідає воскресінню, то умову надання знижки можна записати:

If (Summa > 4300) **and** (Day = 1) **then** Showmessage ('Знижка') **else** Showmessage (' Немає знижки');

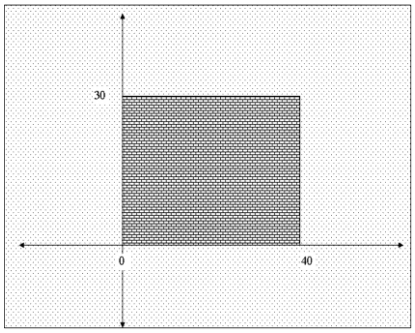


Рисунок 3.3 – Схема розміщення прямокутника в системі координат

Хай залежно від значення координати точки (x, y), що вводиться, слід видати на екран повідомлення: крапка потрапила в прямокутник чи ні (рис.3.3). Оператор виводу має вигляд:

If (x>0) and (x<40) and (y<30) and (y>0) **then** showmessage ('Крапка знаходиться усередині прямокутника') **else** showmessage ('Крапка поза прямокутником');

Вибір в точці розгалуження алгоритму чергового кроку програми може бути реалізованим за допомогою оператора **case**.

Якщо оператор **if** дозволяє вибрати один з двох можливих варіантів, інструкція **case** — один з декількох.

Множинний вибір може бути реалізований за допомогою вкладених одна в іншу інструкцій **if**. Такий підхід не завжди зручний, особливо в тому випадку, якщо кількість варіантів ходу програми велика.

У мові Delphi є оператор **case**, який дозволяє ефективно реалізувати множинний вибір. У загальному вигляді вона записується таким чином:

```
case [Селектор] of
    список1: оператор  1  ;
    список 2:  оператор 2;
    список N: оператор N
else
оператор
end;
```

де:

- ✓ Селектор — вираз, значення якого визначає подальший хід виконання програми (тобто послідовність інструкцій, яка буде виконана); Селектор повинен мати порядковий тип. Порядковими типами називаються такі типи даних, у яких значення впорядковані і для кожного поточного значення можна вказати попереднє і наступне. Прикладом є дані цілого типу або дані типу char. Відзначимо, що така особливість зменшує область дії оператора **case**, оскільки як селектор не можна використовувати дробові числа, текстові рядки, на відміну від оператора **if**, у якого немає обмеження вказаного вигляду.
- ✓ Список N — список констант. Якщо константи є діапазоном чисел, то замість списку можна вказати першу і останню константу діапазону, розділивши їх двома крапками. Наприклад, список 1, 2, 3, 4, 5, 6 може бути замінений діапазоном 1..6.

Виконується інструкція **case** таким чином:

1. Спочатку обчислюється значення виразу-селектора.
2. Значення виразу-селектора послідовне сравни-вается з константами із списків констант.
3. Якщо значення виразу співпадає з константою із списку, то виконується відповідна цьому списку група інструкцій. На цьому виконання інструкції **case** завершується.

4. Якщо значення виразу-селектора не співпадає ні з однією константою зі всіх списків, то виконується послідовність інструкцій, наступна за else.

Синтаксис інструкції **case** дозволяє не писати **else** і відповідну послідовність інструкцій. В цьому випадку, якщо значення виразу не співпадає ні з однією константою зі всіх списків, то виконується наступна за case інструкція програми.

```
case n   of
1,2,3,4,5: day:='Рабочий день. ' ;
6: day:='Суббота!';
7: day:='Воскресенье!';
end;
```

При роботі конструкції вибір значення порядкової змінної порівнюється послідовно зі всіма можливими значеннями, і як тільки знайдеться значення, якому дорівнює порядкова змінна, виконується гілка операторів, порівняння припиняються і конструкція, що управляє, припиняє свою роботу. Тому при складанні програм з використанням конструкції, що управляє, "вибір", раніше необхідно розташовувати "строгіші" послідовності операторів.

Зі всіх можливих гілок операторів конструкції вибір виконується тільки одна гілка, навіть якщо значення порядкової змінної співпадає з декількома можливими значеннями, що відповідають за різні гілки операторів. Виконується та гілка, яка розташована раніше.

Допустимо, змінна *m* зберігає номер місяця, тоді за допомогою **case** легко організувати розрахунок кількості днів в кожному місяці:

```
case m_ of
1,3,5,7,8,10,12: day:=31;
4,6,9.11: day:=30;
2: if  year mod 4 = 0 then day:=29 else day;=28;
 end;
```

Якщо номер місяця рівний 1 або 3, або 5, або 7., то кількість днів цього місяця рівна 31. Аналогічно діє перевірка на четвертий місяць, шостий і останні. Виняток становить лютий. У нім може бути 28 або 29 днів залежно від того, чи є рік високосним чи ні.

Раніше розглядалися арифметичні операції, до яких окрім звичайних складання, віднімання, ділення і множення відносяться операції **div** і **mod**. **Div** – цілочисельне ділення, при якому дробова частина відкидається без округлення. Наприклад, 11 **div** 3 =3; 27 **div** 5 =5. Операція **mod** обчислює ту частину, яка не ділилася при цілочисельному діленні. 11 **mod** 3=2, 27 **mod** 5=2.

Ознакою високосного року є ділення його порядкового номера на чотири без залишку. 2000 рік був високосним і число 2000 ділиться без залишку на чотири. Це ж відноситься і до 2008 року, а ось 2007 рік високосним не був, число 2007/4=501.75. Таким чином, якщо залишок від цілочисельного ділення рівний нулю, то рік був високосним.

Ця перевірка реалізована за допомогою оператора If

```
if year mod 4 = 0 then day:=29 else day;=28;
```

якщо залишок від ділення на чотири рівний нулю, то в лютому високосного року 29 днів, інакше 28.

Основні положення теми:

✓ Для ввода|вывода інформації використовують компоненти Edit, Panel і TrackBar.

✓ Вся інформація, що вводиться, повинна бути перетворена з тексту в числові дані відповідного типу для подальших розрахунків.

✓ Вся інформація, що виводиться, повинна бути перетворена з числових типів в текст для подальшого виводу на екран.

✓ При розрахунках формул використовується оператор привласнення.

✓ При програмуванні формул можна використовувати числа, змінні, функції, знаки операцій і парні круглі дужки.

✓ При програмуванні слід враховувати пріоритет операцій.

✓ Для організації різного виду дій залежно від перевірки умов використовується оператор **If**. Він має формат: ***if*** *умова* ***then*** *оператор1* ***else*** *оператор2***.**

✓ Після слів **then** і **else** дозволяється використовувати тільки один оператора. Якщо потрібно виконати множинну послідовність дій, оператори поміщаються в операторні дужки **begin** оператор1, оператор2, оператор3. **end .**

✓ Допустимі вкладені оператори **If .**

✓ При програмуванні умов дозволяється використовувати шість операцій порівняння і трьох логічних операторів.

✓ Для організації множинного вибору використовують оператора **case**. Оператор **case** дозволяє використовувати як селектор дані тільки порядкового типу.

Тести для самоперевірки знань до теми 2

Завдання 1–9 мають по чотири варіанти відповіді, серед яких лише ОДИН ПРАВИЛЬНИЙ. Виберіть правильний варіант відповіді.

1. Компонент **Track Bar**, використовуваний для введення інформації, належить до панелі кнопок:

а) Additional
б) Format
в) Win32
г) Standart

2. У розрахунковій формулі пріоритет операції змінюють:

а) Знаки операцій
б) Круглі дужки
в) Фігурні дужки
г) Квадратні дужки

3. Функція, що перетворює строковий тип у цілочисельний:

а) Strtoint
б) Str
в) Strtofloat
г) Floattostr

4. Функція, що перетворює строковий тип у число з плаваючою комою:

а) Strtoint
б) Str
в) Strtofloat
г) Floattostr

5. Для формування події при натисканні на кнопку використовується компонент:

а) Button
б) Label
в) Edit
г) Panel

6. Основною властивістю компонента **Edit** є властивість:

а) Caption
б) Color
в) Text
г) Font

7. Оператор **If** у загальному виді записується в такий спосіб:

а) If<умова> else <оператор1>then<оператор2>;
б) If<умова> then <оператор1>else<оператор2>;
в) If<умова> then <оператор1>; else<оператор2>;
г) If<умова>; then <оператор1>; else<оператор2>;

8. Якщо в будь-яку гілку розгалуження потрібно вставити декілько операторів, то вони поєднуються в один оператор, розміщений в операторних дужках **begin** і **end**, що називається:

а) складний
б) складений
в) простій
г) множинний

Завдання 9–11 мають два варіанти відповіді. Треба обрати ПРАВИЛЬНИЙ, щоб приведений вислів мав сенс.

9. При розрахунках формул використовується оператор присвоювання, який має наступний формат:

Змінна:=обчислення; Так | Ні

10. Коментарі - це будь-яка послідовність символів, укладена у фігурні дужки, що пояснює програму, але не впливає на хід її виконання.

Так | Ні

11. Оператор циклу **For** застосовують у випадку, коли число повторень послідовності дій попередньо невідомо.

Так | Ні

Завдання 12–15 мають на меті встановлення відповідності. До кожного рядка, позначеного ЦИФРОЮ, доберіть відповідник, позначений БУКВОЮ.

12. Установите відповідність між функціями, які дозволяють виводити інформацію на екран, та їх ротлумаченням:

1. Round
2. Str
3. Trunc

А. Функція, що дозволяє виводити на екран число з заданим ступенем точності
Б. Функція, що перетворить число з відкиданням його дробової частини
В. Функція, дозволяє виводити на екран інформацію будь-якого формату
Г. Функція, що перетворить число з округленням його дробової частини

	А	Б	В	Г
1				
2				
3				

13. Установите відповідне положення шкали для компонента **Track Bar**, обумовлене властивістю **Tick Marks**:

1. tm Both
2. tm Bottom Right
3. tm Top Left

А. Унизу повзуна
Б. Знизу і зверху
В. Ліворуч повзуна
Г. Зверху повзуна

	А	Б	В	Г
1				
2				
3				

14. Установите відповідність між назвою операторів і видами циклів, утворених за їх допомогою:

	А	Б	В	Г
1				
2				
3				

1. REPEAT
2. FOR
3. WHILE

А. З відомим числом повторень
Б. З випадковим числом повторень
В. З постумовою
Г. З передумовою

15. Установите відповідні символи для використання наступних арифметичних операцій Delphi:

	А	Б	В	Г	Д	Е	Ж
1							
2							
3							
4							
5							
6							

1. +
2. -
3. *
4. /
5. mod
6. div

А. Піднесення до ступеню
Б. Залишок від ділення
В. Цілочисельне ділення
Г. Віднімання
Д. Складання
Е. Множення
Ж. Ділення

Питання для самоконтролю

1. Кратко охарактеризуйте наступні візуальні компоненти: Edit, Label, Button.
2. Які основні властивості компонентів Edit, Label, Button?
3. Що означають функції: StrToFloat, StrToInt, FloatToStr, IntToStr?
4. Назвіть відомі вам арифметичні операції та функції.
5. Розкажіть про принцип пріоритету операцій
6. Яким чином об’являються цілі змінні? змінні ?
7. Як визначити значення змінної?
8. Опишите оператор привласнення й правила його використання.
9. Для чого призначений умовний оператор?
10. Які дві форми запису умовного оператора?
11. Чи може умовний оператор містити в собі інших умовних операторів?
12. Коли і як застосовується складений оператор?
13. Для чого призначений оператор вибору?
14. Чи можуть виконатися декілька гілок оператора "вибір" за один раз?
15. Скільки операторів можна написати після можливого значення змінної?
16. Чи може відразу декілька констант фігурувати як можливе значення змінної?

3 РОЗРОБКА ПРОГРАМ З ВИКОРИСТАННЯМ ЦИКЛІЧНИХ ПРОЦЕСІВ. ПРОГРАМИ ОБРОБКИ ОДНОВИМІРНИХ МАСИВІВ

Алгоритм, в якому є послідовність операцій, яка повинна бути виконана кілька разів, називається циклічним, а сама послідовність операцій іменується циклом. У програмі цикл може бути реалізований за допомогою інструкцій **for**, **while** і **repeat**.

3.1 Реалізація циклу While

Завдання 1. Задані початкове, кінцеве значення *X* і крок. Визначити для кожного *X* значення, визначуване за формулою $y= 3x^2 -9$ і вивести значення *X* і відповідні значення *Y* на екран.

На рис.3.1 показана форма з компонентами (частина 1), які в ній використовуються. Для введення початкового, кінцевого значення та кроку змінювань використані за звичаєм компоненти **Edit**, а ось для виведення результатів нами був обраний новий компонент **Memo** зі стандартної панелі компонентів. Для введення заголовку використана власти-вість Lines (частина 2 рис.3.1), при натиску-ванні на рядок з цією властивістю відкрива-ється вікно, в котре можна вносити будь-який текст (частина 3 рис.3.1).

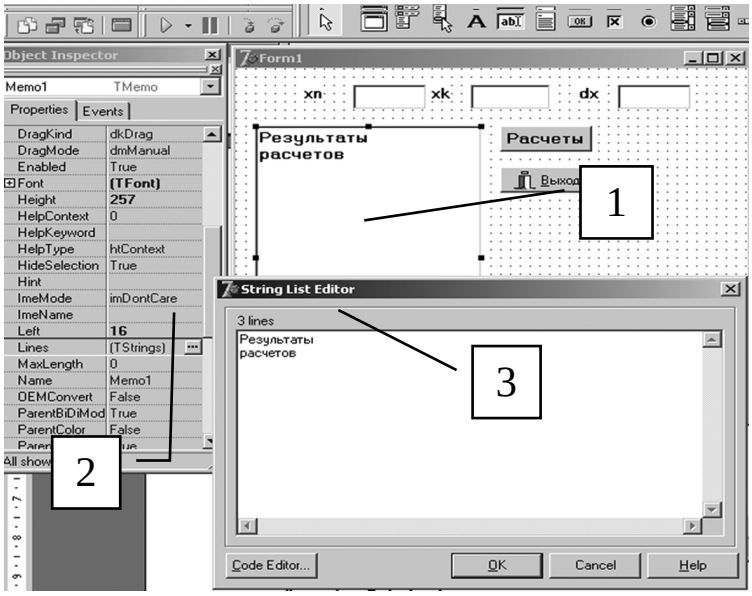


Рисунок 3.1 – Створення форми з використанням компоненту **Memo**

Припустимо, що потрібно провести розрахунки від -10 до 10 з кроком, рівним 2. Якби ми проводили розрахунки вручну, то спочатку узяли б значення, рівне -10 і визначили величину *y*, потім додали б крок, для набутого значення *x=-8* визначили відповідне *y* і так доти, доки *x* не перевищить значення, рівне 10.

Для організації такого обчислювального процесу використовується оператор циклу з передумовою **while.**

Цикл WHILE (англ. "*доки*") - цикл, в якому умова знаходиться перед тілом цикла, а сам цикл виконується дотиди, доки умова не стає помилковою.

```
procedure TForm1.Button1Click(Sender: TObject);
var x,dx,y,xn,xk: real;     {Опис локальних змінних}
begin
xn := strtofloat(edit1.Text);
dx:=strtofloat(edit3.Text); {Введення початкових даних}
xk:= strtofloat(edit2.Text);
x:=xn;        {Вважаємо поточне значення x рівним початковому значенню }
while x<=xk do begin     {Умова роботи циклу}
y := 3*sqr(x)-9;     {Розрахунок за формулою}
 memo1.lines.add('x='+floattostr(x)+'          y='+floattostr(y));
 {Виведення в компонент Мемо результатів розрахунку y}
x :=x + dx;  {Додавання кроку до попереднього значення X} end;    end;
```

Оператор **while** використовується в тому випадку, коли деяку послідовність дій треба виконати кілька разів, причому необхідна кількість повторень циклу під час розробки програми невідома і може бути визначена тільки в процесі роботи програми.

Типовими прикладами використання циклу **while** є обчислення із заданою точністю, пошук в масиві або у файлі.

У загальному вигляді оператор **while** записується таким чином:

while [умова] **do begin** {оператори, яких потрібно виконати кілька разів} **end**

де [умова] — вираз логічного типу, який визначає умову виконання інструкцій циклу, програмується за тими ж принципами, що й умови в умовних операторах.

Виконання оператора **while**:

1. Спочатку перевіряється умова.
2. Якщо умова не виконується (рівна ***False***), то на цьому робота циклу і оператора **while** завершується.
3. Якщо умова виконується (рівна ***True***), то послідовно виконуються розташовані між **begin** і **end** оператори тіла циклу.то послідовно виконуються розташовані між **begin** і **end** оператори тіла циклу. Після цього знову перевіряється виконання умови. Якщо умова

виконується, то оператори циклу виконуються ще раз. І так до тих пір, доки умова не стане помилковою (False).

Після оператора **Do** дозволяється ставити тільки один оператор. Якщо їх в циклі необхідно поставити декілька, то використовують операторні дужки **begin** і **end**, утворюючи тим самим ***складеного*** оператора.

Особливістю циклу з передумовою є те, що він може не виконатися жодного разу - це відбудеться, якщо вказана умова спочатку буде помилковою. При цьому, цикл може і стати "вічним" - якщо умова ніколи не прийме значення False. Саме тому слід стежити за тим, щоб були завжди присутні умови для завершення роботи циклу.

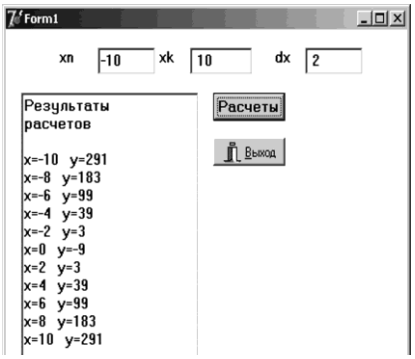


Рисунок 3.2 – Результати виконання програми

На рис.3.2 приведена копія екрану з результатами виконання програми. Розглянемо оператор виведення даних на екран. При виведенні результатів використовується компонент **Memo1**, подія **lines.add** - для додавання нових рядків при виведенні інформації:

```
memo1.lines.add('x='+floattostr(x)+'y='+floattostr(y));
```

Та частина інформації, яка розміщена в лапках, виводиться на екран без змін, **FloatToStr(x)** – виводить само значення X, знак «+» використаний для об’єднання текстових фрагментів.

Відмінною особливістю оператора **While** є, по-перше, те, що спочатку йде перевірка умови входу в цикл, а тільки потім його виконання і, по-друге, у ролі параметрів циклу (початкового, кінцевого значення і кроку циклу), можна використовувати будь-які значення.

3.2 Реалізація циклу Repeat

Наступним оператором циклу є оператор **Repeat**, який називають оператором циклу з постумовою. Спочатку хоча б один раз виконується цикл і тільки потім здійснюється перевірка на можливість його повторення.

У загальному вигляді оператор **repeat** записується таким чином:

Repeat { оператори} **until** [умова]

де [умова] — вираз логічного типу, що визначає умову завершення циклу. Всі оператори, які необхідно виконати в циклі розміщують між словами **Repeat** та **Unti1.** На відміну від оператора **While** не потрібно використання операторних дужок **begin** і **end**.

Виконання оператора **Repeat**:

1. Виконуються оператори, наступні за службовим словом Repeat
2. Після цього перевіряється умова.
3. Якщо умова помилкова, то відбувається повернення до виконання операторів, наступних за службовим словом Repeat і знову перевіряється умова.
4. Якщо умова істинна, то виконання тіла циклу припиняється.

У "жаргонному" перекладі оператор циклу з постумовою "звучить" так: *Повторювати тіло циклу поки не виконається умова*.

У циклі Repeat тіло циклу виконується принаймні один раз.

Завдання 2. Змінимо постановку завдання, яке ми розглядали при вивченні оператора **While.** Провести розрахунки *y* за формулою $y=3\sqrt{x}-9$ причому розрахунки припинити, як тільки значення ***x*** стане від'ємним (<0).

Ми не приводимо зовнішній вигляд форми, оскільки вона виконана аналогічно. Текст програми приводиться нижче. Усередині циклу виконуються три оператори: розрахунок *y*, виведення його значень на екран і зміни значення *x* із заданим кроком.

```
procedure TForm1.Button1Click(Sender: TObject);
var x,dx,y,xn: real;
begin
xn := strtofloat(edit1.Text);
dx:= strtofloat(edit3.Text);
x:=xn;
repeat
y := 3*sqrt(x)-9;
 memo1.lines.add('x='+floattostr(x)+'   y='+floattostr(y));
x :=x - dx;
 until x<0;  end;
```

3.3 Реалізація циклу за допомогою оператору For

Оператор **For** використовується в тому випадку, якщо деяку послідовність дій потрібно виконати кілька разів, причому кількість повторень заздалегідь відома.

У загальному вигляді оператор **for** записується у вигляді:

For [лічильник] := [нач_знач] **to** [кон_знач] **do begin**
{ оператори} **end**

де:

[лічильник] - змінна-лічильник кількості повторень інструкцій циклу;

[поч_знач] - вираз, який визначає початкове значення лічильника циклів;

[кін_знач] - вираз, який визначає кінцеве значення лічильника циклів.

Змінна-лічильник, початкове і кінцеве значення повинні бути цілого типу. Кількість повторень інструкцій циклу можна обчислити за формулою:

[кін_знач] -[поч_знач]+ 1.

Виконання оператора **For**:

1. Обчислюються і запам'ятовуються початкове і кінцеве значення параметра циклу, які можуть бути представлені у вигляді конкретного значення (в цьому випадку немає необхідності в обчисленнях) або у вигляді виразу, значення якого обчислюється на початку виконання.
2. Параметру циклу ***i*** привласнюється початкове значення.
3. Значення параметра циклу і порівнюється з кінцевим значенням. Оператор "тіло циклу" виконуватиметься при виконанні наступного условия:i<=кінцевого значення. Інакше відбувається припинення виконання циклічного оператора.
4. Параметру циклу привласнюється наступне більше значення.
5. Виконується пункт 3 даної схеми.

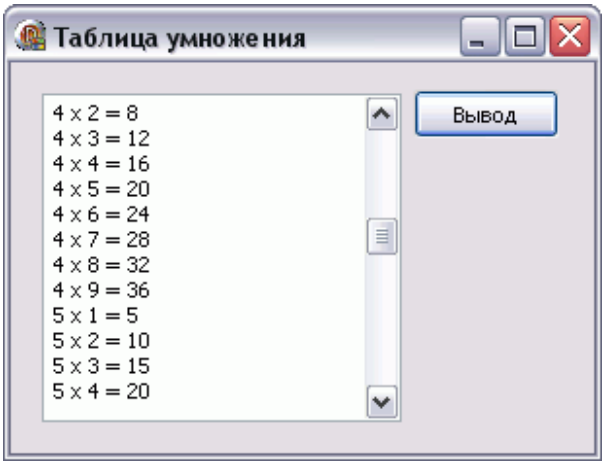
Якщо між **begin** і **end** знаходиться тільки один оператор, то слова **begin** і **end** можна не писати.

3.4 Вкладені цикли

Що таке вкладені цикли, зрозуміло з назви - це цикли, які вкладені в інші цикли. Наприклад, якщо один цикл дозволяє вивести лише ряд чисел, то 2 цикли, один з яких вкладений в інший, дозволять вивести цілу таблицю значень.

Ніяких спеціальних конструкцій для вкладених циклів не немає. Все працює точно також. Змінні-лічильники циклів, як правило, називають буквами I, J, K, хоча назва, звичайно, може бути будь-яке.

Простий приклад застосування вкладеного циклу - виведення таблиці множення. Спершу продумаємо алгоритм: для виведення таблиці для одного конкретного числа (наприклад, для 5), потрібно створити цикл, який пройде значення від 1 до 9 і виведе твір числа 5 на кожне з цих чисел. А щоб вивести таблицю для самих чисел від 1 до 9, потрібний ще один такий же цикл.




```
Procedure TForm1.Button1Click (Sender: TObject);
  var i,j: Integer;
  begin
    Memo1.Lines.Clear;
    for i := 1 to 9 do
      for j := 1 to 9 do
Memo1.Lines.Add(IntToStr(i)+' x '+IntToStr(j)+' = '+IntToStr(i*j))
  end;
```

Рисунок 3.3 – Результати роботи програми та її програмний код

Завдання 3. Ввести масив *X*, заздалегідь задавши кількість його елементів. Визначити елементи масиву за формулами:

$$y_i = 2\sqrt{x_i}, при\_x_i > 0$$

$$y_i = 30 \sin x_i, при\_x_i \le 0$$

Знайти елементи масиву Z як більшего значення елементів масивів X та У. Знайти суму елементів масиву У і добуток його ненульових елементів.

Масив - це структура даних, що дозволяє зберігати під одним ім'ям сукупність даних будь-якого, але обов'язково однакового

типу. Масив характеризується ім'ям, типом інформації, що зберігається, і кількістю його елементів.

Масиви бувають одновимірними та багатовимірними. В математиці та інформатиці масив називають одновимірним, якщо для отримання доступу до його елементів достатньо однієї індексної змінної.

Масив, як і будь-яка змінна програми, перед використанням повинен бути оголошений в розділі опису змінних. У загальному вигляді опис масиву виглядає таким чином:

Ім'я: **array** [нижній_індекс **..** верхній_індекс] **of** тип

де:

ім'я — ім'я масиву;

array — зарезервоване для масиву слово мови Delphi;

нижній_індекс і **верхній_індекс** — цілі константи, які визначають діапазон зміни номерів елементів масиву і, неявно, кількість елементів (розмір) масиву;

тип — тип елементів масиву.

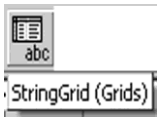
Величини, що позначають нижній верхній індекси,в квадратних дужках розділяються двома крапками. Тип елементів, що входять до складу масива, може бути будь-яким, а тип індексів масиві може бути тільки простим, найчастіше – це цілі числа.

Приклади оголошення масивів:

```
temp:array[1..50] of real;
koef:array[0..4]of integer;
nik:array[1..20] of string;
```

3.5 Робота з масивами з використанням компоненту StringGrid

Для роботи з масивами найчастіше використовують компонент StringGrid, який знаходиться на сторінці ADDITIONAL.



Компонент **StringGrid** є таблицею, що містить рядки. Дані таблиці можуть бути тільки для читання або редагування. Таблиця ділиться на дві частини - фіксовану і робочу. Фіксована служить для показу заголовків стовпців/рядів і для ручного управління їх розмірами. Зазвичай фіксована частина займає крайній лівий стовпець і самий верхній ряд таблиці, проте за допомогою властивостей FixedCols і FixedRows можна задати іншу кількість фіксованих стовпців і рядів (якщо ці властивості дорівнюють 0, то таблиця не містить фіксованої зони). Робоча

частина - це решта частина таблиці. Вона може містити довільну кількість стовпців і рядів, більш того, ці величини можуть змінюватися програмно. Робоча частина може не уміщатися цілком в межах вікна компоненту, в цьому випадку в нього автоматично поміщаються потрібні смуги прокрутки. При прокрутці робочої області фіксована область не зникає, але міняється її вміст - заголовки рядків і рядів.Таким чином, можна задати заголовки стовпців і рядків, постійно присутні у вікні компоненту. Кожному елементу таблиці може посталений у відповідність деякий об'єкт.

Основні властивості компоненту

Cells - комірка, що містить індекси рядка і стовпця.
ColCount, RowCount- визначають число стовпців і рядків.
FixedCols, FixedRows – визначають число не прокручуваних стовпців і рядків. Їх використовують для заголовків.
FixedColor - колір фону фіксованих осередків.
Scrollbars - наявність смуг прокрутки.
Options дозволяє задати безліч властивостей:
GoFixedVertline, GoFixedHorsline - наявність роздільників у фіксованих осередках;
GoVertline, GoHorsline - наявність роздільників в звичайних осередках;
GoColSizing, GoRowSizing- можливість для користувача за допомогою миші змінювати розміри;
GoColMoving, GoRowMoving- можливість переміщати стовпці і рядки;
GoEditing- можливість редагувати вміст таблиці.

Центральною властивістю компоненту є **Cells** - двомірний масив комірок, кожна з яких може містити довільний текст. Кількість осередків по кожному вимірюванню зберігає пара властивостей **Colcount** (кількість стовпців) і **RowCount** (кількість рядів). Значенія цих властивостей і, отже, розміри таблиці можуть мінятися як на етапі розробки програми, так і в ході її роботи, проте їх значення повинні бути як мінімум на одиницю більше відповідно значень у властивості **FixedCols** і **FixedRows**, що визначають розміри фіксованої зони.

Конкретна комірка визначається парою чисел - номером стовпця і номером ряду, на перетині яких він знаходиться

(нумерація починається з нуля). Властивість **Cells** має тип ***String***, тому програма може легко прочитати або записати вміст потрібної комірки.

Перша комірка, розташована у верхньому лівому кутку таблиці має номер 0,0 (рис.3.4); наступний зліва направо осередок нумерується як 1,0; потім 2,0.

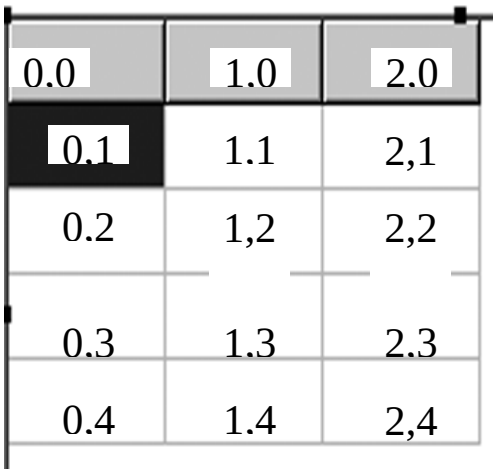


Рисунок 3.4 - Нумерація комірок в об'єкті **Stringgrid**

Першим в нумерації слідує номер стовпця починаючи з нуля і далі, другим номером йде номер рядка, що також нумерується від нуля.Наприклад:

```
Cells [1,1] := 'Лівий верхній осередок робочої зони';
```

Для вирішення поставленого вище завдання на форму поміщений об'єкт **StringGrid**, причому для нього задана кількість стовпців **Colcount** рівною трьом, кількість рядків **RowCount** спочатку задається рівною п'яти. Оскільки передбачалося, що в першому рядку знаходитимуться заголовки, тобто назви масивів, які вводяться і які обчислюються, та властивість *Fixedcols=0* (немає стовпця як заголовка) і *Fixedrows=1* (один рядок використовується як заголовок). Вміст осередків можна редагувати. Для цього в таблиці викликається підвластивість **GoEditing**, що знаходиться у властивості **Options** :*GoEditing=true.* Це робиться для того, щоб в таблиці при запуску програми можна було вводити початкові дані. Решта властивостей (кольори, шрифт.) задається на свій розсуд.

Для оптимальної роботи з програмою організовується стандартне меню. Для введення меню використовується компонент **MainMenu** панелі **STANDARD**, який розміщується на екрані в будь-якому місці, оскільки при запуску програми він не

відображається. Замість нього у верхньому рядку екрану з'являється рядок меню, як і в стандартному вікні Windows. При подвійному натискуванні по кнопці ***MainMenu*** з'являється вікно для введення пунктів меню. Пунктирні прямокутники призначені для введення (властивість ***Caption***) назв пунктів меню (рис.3.5). Переміщаючи курсор по пунктирним прямокутникам то вниз, то ліворуч, створюються додаткові пункти меню.

Рисунок 3.5 – Створення меню з використанням компоненту **MainMenu**

Зверніть увагу на властивість **ShortCut**, яка використовується для завдання клавіш швидкого виклику, відкривши список цієї властивості ви отримуєте можливість вибрати одну з пропонованих комбінацій клавіш.

На рис.3.6 показана форма з винесеними на неї об'єктами **Stringgrid**, **MainMenu** і трьома компонентами **Edit**, які використовуватимуться для введення кількості елементів в масиві X і виведення результатів визначення суми і твору елементів.

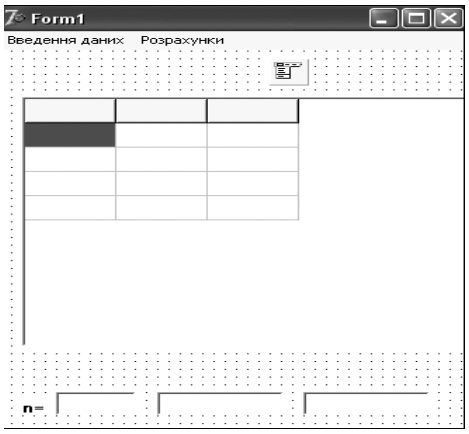


Рисунок 3.6 – Загальний вид створеної форми

Якщо ми хочемо, щоб відразу при завантаженні форми з'явилася таблиця із заголовками, то двічі клацаємо по самій формі і в скомпільованій процедурі вводимо текст

```
procedure TForm1.FormCreate(Sender: TObject);
begin
stringgrid1.Cells[0,0]:='X'; {Завдання заголовка нульового стовпця}
stringgrid1.Cells[1,0]:='Y'; {Завдання заголовка першого стовпця}
stringgrid1.ColWidths[0]:=50; {Завдання ширини нульового стовпця}
stringgrid1.ColWidths[1]:=150; {Завдання ширини першого стовпця}
end;
```

Пункт меню **Введення/Кількість елементів** (рис.3.6) використовується для введення в компонент **Edit1** передбачуваної кількості елементів і зміни кількості рядків таблиці у відповідність з введеним значенням.

```
        procedure TForm1.N2Click(Sender: TObject);
        begin
N:=strtoint(edit1.text);   Введення кількості елементів}
stringgrid1.RowCount:=n+1; {Кількість рядків в таблиці більше на одиницю введеного числа, оскільки один рядок відводиться під заголовок}
        end;

        procedure TForm1.N3Click(Sender: TObject);
        var i:integer;
        begin
        for i:=1 to n do
        x[i]:=strtoint(stringgrid1.Cells[0,i] );
        end;
```

Пункт меню **Введення/Масив *X*** використовується для введення елементів масиву *X*. Зверніть увагу на те, що кожен елемент масиву вводиться окремо. Відбувається прочитування поелементно вмісту кожної комірки нульового стовпця і збереження зчитуваного вмісту в масиві.

```
        procedure TForm1.N3Click(Sender: TObject);
        var i:integer;       {Опис локальних змінних}
        begin
        for i:=1 to n do  Організація циклу для виконання послідовності дій запису лічених значень, що повторюється, в масив}
        x[i]:=strtoint(stringgrid1.Cells[0,i]);
        end;
```

Пункт меню **Розрахунки/Масив *У*** використовується для обчислення елементів масиву *У*. У циклі відбувається обчислення елементів масиву *У*, перетворення їх в строковий тип і запис в перший стовпець таблиці.

```
        procedure TForm1.N4Click(Sender: TObject);
        var i:integer; xx:string;        Опис локальних змінних}
        begin
for i:=1 to n do   begin{Організація циклу для виконання послідовності обчислень, що повторюється }
        if x[i]>0 then y[i]:=2*sqrt(x[i])
```

```
        else y[i]:=30*sin(x[i]);
  str(y[i]:1:2,xx);     {Перетворення в строковий тип}
        stringgrid1.Cells[1,i]:='y['+inttostr(i)+']='+xx;     {Заповнення
значеннями масиву У першого стовпця таблиці}
        end;  end;
```

Пункт меню **Розрахунки/Масив *Z*** використовується для обчислення елементів масиву *Z*. У циклі відбувається обчислення елементів масиву *Z*, перетворення їх в строковий тип і запис в другий стовпець таблиці.

```
        procedure TForm1.Z1Click(Sender: TObject);
        var i:integer;xx:string;  Опис локальних змінних}
        begin
stringgrid1.Cells[2,0]:='Z';      {Створення заголовка другого
стовпця}
        stringgrid1.ColWidths[2]:=150;
        for i:=1 to n do   begin
        if x[i]>y[i] then z[i]:=x[i] else z[i]:=y[i];     Обчислення
елементів масиву Z}
str(z[i]:1:2,xx);       {Перетворення в строковий тип}
stringgrid1.Cells[2,i]:='z['+inttostr(i)+']='+xx;{Заповнення
значеннями масиву Z стовпця таблиці}end;   end;
```

Пункт меню **Розрахунки/Сума** використовується для обчислення суми елементів масиву *У*. Для обчислення суми спочатку встановлюють її значення рівним нулю. Це дозволяє виконувати обчислення багато разів, не додаючи до попереднього значення наступні. У циклі відбувається накопичення значень суми. Спочатку *s=0*, потім до цього значення додається перше значення елементу масиву *У*, т.е *s:=s+y[1]*, потім до вже існуючого значення додається ще одне *s:=s+y[1]+y[2]* і так n разів, які працює цикл.

```
        procedure TForm1.N6Click(Sender: TObject);
        var i:integer; xx:string;  s:real;   { Опис локальних змінних}
        begin s:=0;    {Установка початкового значення суми
рівним нулю}
         for i:=1 to n do
         s:=s+y[i];   Накопичення суми}
         str(s:1:2,xx); {Перетворення в строковий тип}
edit2.Text:='s='+xx;{Виведення результату.} end;
```

Пункт меню **Розрахунки|Добуток** організований для обчислення добутку ненульових елементів масиву У. Для обчислення добутку спочатку встановлюють його значення рівним

одиниці. Якби, як і у попередньому випадку, значення встановили рівним нулю, то це привело б до нульового результату обчислень, оскільки множення на нуль дає в результаті нуль. У циклі відбувається накопичення значень добутку. Спочатку p=1, потім це значення перемножується на перше значення елементу масиву У, тобто p:=p*y[1], потім вже існуюче значення перемножується на ще одне p:=p*y[1]*y[2] і так *n* разів, доки працює цикл. Відмінність від попередньої процедури полягає в тому, що заздалегідь відбувається перевірка на рівність нулю елементів.

```
procedure TForm1.N7Click(Sender: TObject);
var i:integer; xx:string; p:real;   { Опис локальних змінних}
beginp:=1;   for i:=1 to n do
if y[i]<>0        {Добуток тільки ненульових елементів}
then  p:=p*y[i];        {Накопичення добутку}
str(p:1:2,xx);    {Перетворення в строковий тип}
edit3.Text:='p='+xx; {Виведення результату в компонент
Edit} end;
```

Частина змінних була описана як локальні в кожній процедурі. Це означало, що вони будуть використані тільки в цій процедурі. Параметр змінної циклу *i* винен обов'язково описаний в тій процедурі, в якій він використовується. Масиви *X, Y, Z* описуються як глобальні, оскільки вони використовуються в декількох процедурах.

```
public    n:integer;
   x:array[1..20] of integer; {Опис цілочисельного масиву X}
      y,z:array[1..20] of  real; {Опис речових масивів Y і Z}
```

На рис.3.7 представлені результати роботи програми. Значення *n* було введене рівним шести, що привело до збільшення кількості рядків в таблиці.

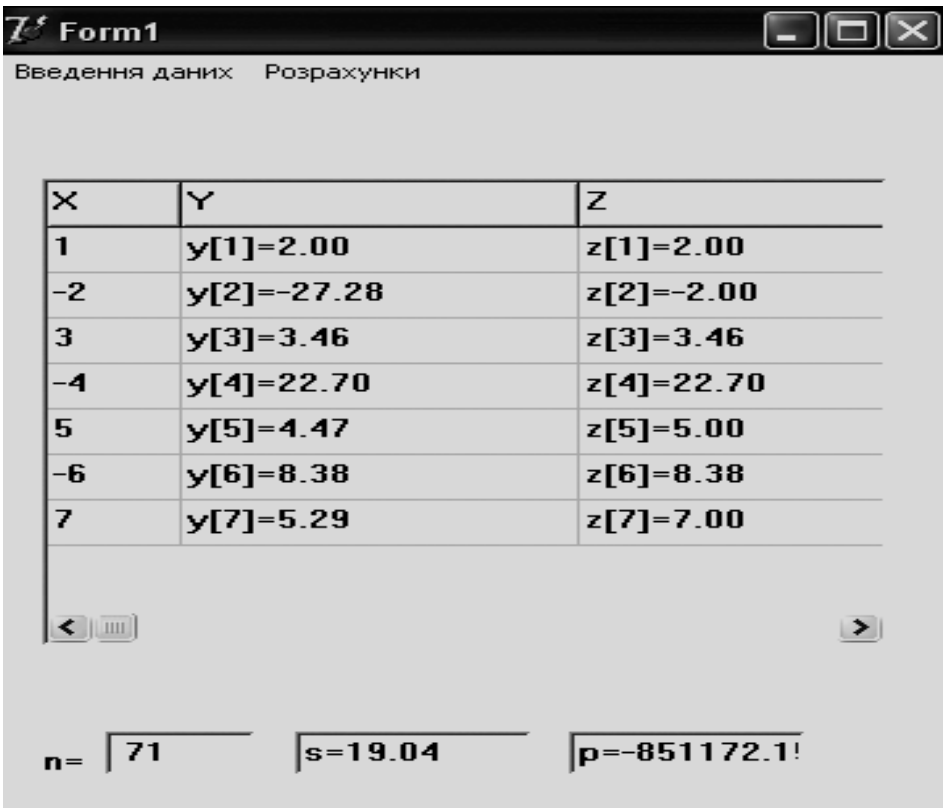


Рисунок 3.7 - Результати роботи програми

Після введення масиву *X* сформовані два інших масиви і розраховані сума всіх елементів масиву У, а також добуток всіх ненульових елементів масиву У.

3.6 Переривання і продовження циклу

Примітка: *то, про що піде мова далі, застосовно не тільки до циклу по змінній, але і до циклів з передумовою і постумовою.*

Не завжди виконання фіксованого числа ітерацій приводить до потрібного результату. Іноді в процесі виконання можуть виникнути ситуації, коли цикл логічно було б завершити, не виконуючи його до кінця, тобто потрібно просто виключити всі подальші ітерації. Така можливість існує - для цього необхідно викликати команду **BREAK** або **EXIT**.

Оператор **Break** виконується для переривання будь-якого циклу, організованого операторами **For**, **Repeat** або **While** Дана команда завершує цикл, який виконується в даний момент, і продовжує виконання програми. При цьому поточна ітерація не

виконується до кінця - переривання відбувається саме в тому рядку, де вказана команда *Break*.

```
procedure TForm1.N3Click(Sender: TObject);
        var i,n: integer;   Опис змінних i,n}
X:array[1..20] of integer;{Опис цілочисельного масиву X}
        begin
        for i:=1 to n do
x[i]:=strtoint(stringgrid1.Cells[0,i]);{Введення цілочисельного масиву X}
        if x[i]>0 then
        begin edt1.text:=inttostr(x[i]); break;
        end; {Якщо елемент масиву X більше нуля, він видається в компонент Edit1 і виконання циклу припиняється} end;
```

Оператор **Exit** используется для прерывания не только цикла, но и всей программы. Например, найти первый положительный элемент массива *X*.

Зауваження: якщо цикл, виконання якого уривається командою *Break*, вкладений в інший цикл, то "зовнішній" цикл продовжить своє виконання, тобто команда *Break* зупиняє тільки один цикл, а не все що є.

Завершення циклу - екстрений метод. Іноді ж потрібно просто пропустити поточну ітерацію і перейти до наступної. Уручну це можна зробити, уклавши всі команди в блок умовного оператора, проте такий здатний незручний і лише захаращує код. Саме тому існує команда продовження циклу і називається вона абсолютно логічно - **CONTINUE**. Ця команда "примушує" цикл тут же перейти до наступної интерации, не продовжуючи виконання поточної.

3.7 Символи і рядки

Комп'ютер може обробляти не тільки чисельну інформацію, але і символьну. Мова Delphi оперує з символьною інформацією, яка може бути представлена як окремими символами, так і рядками (послідовністю символів).

Для зберігання і обробки символів використовуються змінні типу *Ansichar* і *wideChar*. Типом Ansichar є набір ANSI-символов, з якому кожен символ кодується восьмирозрядним двійковим числом (байтом). Типом wideChar є набір символів в кодуванні Unicode, в якому кожен символ кодується двома байтами.

Для забезпечення сумісності з попередніми версіями підтримується тип Char, еквівалентний AnsiChar.

Значенням змінної символьного типу може бути будь-який символ, що відображається:

- буква російського або латинського алфавітів;
- цифра;
- розділовий знак;

І спеціальний символ, наприклад, "новий рядок".

Змінна символьного типу повинна бути оголошена в розділі оголошення змінних. Інструкція оголошення символьній змінній в загальному вигляді виглядає так:

```
Ім'я: char;
```

де:

- *ім'я* — ім'я змінної символьного типу;
- *char* — ключове слово позначення символьного типу.

Приклади:

```
otv: char;
```
та
```
ch: char;
```

Як і будь-яка змінна програми, змінна типу char може набути значення в результаті виконання інструкції привласнення. Якщо змінна типу char набуває значення в результаті виконання операції привласнення, то праворуч від знаку := повинен стояти вираз типу char, наприклад, змінна типу char або символьна константа — символ, ув'язнений в лапки.

В результаті виконання інструкцій
```
c1 := '*';
c2 := c1;
```
змінна *c1* набуває значення привласненням значення константи, а змінна *c2* — привласненням значення змінній cl (передбачається, що змінні *c1* і *c2* є змінними символьного типу).

Змінну типу char можна порівняти з іншою змінною типу char або з символьною константою. Порівняння засноване на тому, що кожному символу поставлено у відповідність число (Додаток А), причому символу 'о' відповідає число менше, ніж символу У, символу 'А' — менше, ніж 'в', символу V — менше, ніж а. Таким чином, можна записати:

```
'0'<'1'<..<'9'<..<'A'<'B'<..<'Z'<'a'<'b'<..<'z'
```

Символам російського алфавіту відповідають числа більші, ніж символам латинського алфавіту, при цьому справедливо наступне:

'А'<'б'<'в'<..<'Ю'<'Я'<'а'<'б'<'в'<...<'э'<'ю'<'я'

У тексті програми замість символу можна вказати його код, поставивши перед числом оператора #. Наприклад, замість константи 'в' можна записати #193. Такий спосіб запису, як правило, використовують для запису службових символів або символів, які під час набору програми не можна ввести з клавіатури. Наприклад, часто використовуваний при записі повідомлень символ "новий рядок" записується так: #13.

У програмах обробки символьної інформації часто використовують функції **chr** і **Ord**. Значенням функції **chr** є символ, код якого вказаний як параметр. Наприклад, в результаті виконання інструкції c:=chr(32) змінної ***з*** буде привласнено значення пропуск. Функція **ord** дозволяє визначити код символу, який передається їй як параметр. Наприклад, в результаті виконання інструкції k:=ord('*') змінна до міститиме число 42 — код символу *.

Програма, текст якої приведений нижче, виводить таблицю кодування букв російського алфавіту. Вид вікна програми представлений на рис. 3.8.

Основну роботу виконує процедура обробки події OnActivate, яка формує і виводить в полі влучні (Label1) таблицю. Подія OnActivate відбувається при активізації форми додатку, і тому процедура TForm1.FormActivate виконується автоматично, відразу після появи форми на екрані.

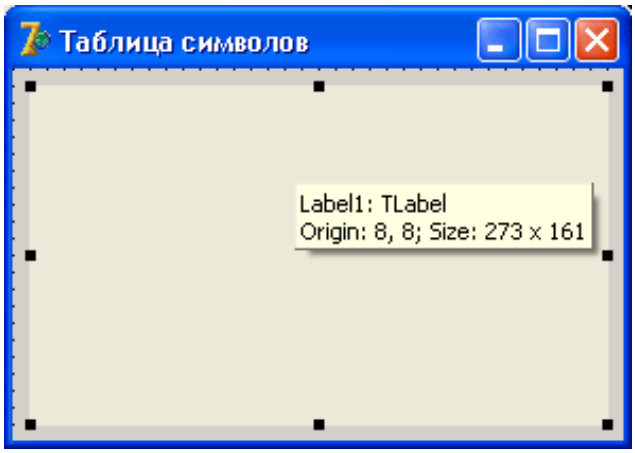


Рисунок 3.8 - Форма додатку під час розробки

```
unit tablsim_;
interface
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics
Controls, Forms, Dialogs, StdCtrls;
type
TForm1 = class(TForm)
Label1: TLabe1;
procedure FormActivate(Sender: TObject); private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.FormActivate(Sender: TObject);
var
st:string; { таблиця формується як рядок символів}
dec: byte;  { код символу}
i,j:integer;  { номер рядка і колонки таблиці }
begin
st:='';
dec:=192;
for i:=0 to 15 do {шістнадцять рядків }
begin
dec:=i + 192;
for j:=1 to 4 do {чотири колонки }
begin
st:=st+chr(dec)+'-'+IntToStr(dec)+' '; dec:=dec + 16;
end;
st:=st + #13; { перехід до нового рядка екрану}
end;
Label1.caption:=st;
end;
end.
```

Вид вікна додатку під час роботи приведений на мал. 3.9.

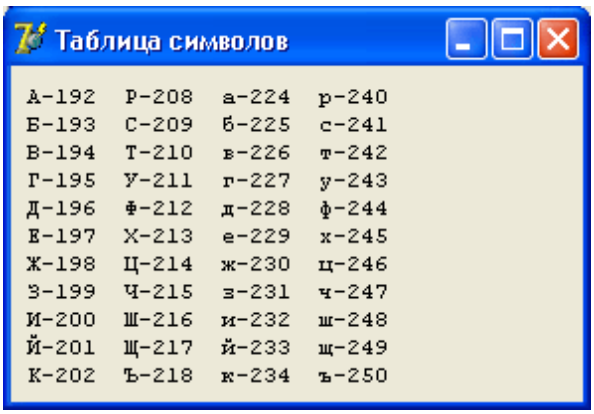


Рисунок 3.9 - Форма додатку під час роботи

Форма додатку **Таблиця символів** містить тільки один компонент - поле мітки (Label1l). Для того, щоб колонки таблиці мали однакову ширину, властивості Label1.Font.Name слід привласнити ім'я шрифту, у якого всі символи мають однакову ширину, наприклад, courier New cyr.

Рядки можуть бути представлені наступними типами: *shortstring (*короткий рядок*), Longstring* (довгий рядок) и *widestring* (широкий рядок). Розрізняються ці типи гранично допустимою довжиною рядка, способом виділення пам'яті для змінних і методом кодування символів.

Змінній типу *shortstring* пам'ять виділяється статично, тобто до початку виконання програми, і кількість символів такого рядка не може перевищувати 255. Змінним типу Longstring і widestring пам'ять виділяється динамічно — під час роботи програми, тому довжина таких рядків практично не обмежена.

Робота з рядками Delphi дозволяє витягувати з рядка необхідну інформацію і представити її в потрібному вигляді. Система надає весь спектр необхідних функцій для роботи **з** рядками Delphi і перетворення рядків Delphi в необхідні формати:

- числовий формат, цілий і дріб з плаваючою крапкою;
- формат часу, дати, дати-часу;
- перетворення символів до верхнього або нижнього регістра;
- порівняння рядків, пошук в рядку і копіювання підрядка;

  і багато інших...

Безпосередньо **самі** рядки Delphi підтримують єдину операцію, так звану **операцію конкатенації**, тобто приєднання. Не дивлячись на "наукову" назву, операція конкатенації виконує дуже

просту процедуру. За допомогою операції конкатенації один рядок приєднується до іншого.

Крім перерахованих вище типів можна застосовувати універсальний строковий тип *String*, який еквівалентний типу *Shortstring*.

Змінна строкового типу повинна бути оголошена в розділі оголошення змінних. Інструкція оголошення в загальному вигляді виглядає так:

```
Ім'я: String;
```

або

```
Ім'я: String [довжина]
```

де:

- ***ім'я*** — ім'я змінної;
- ***string*** — ключове слово позначення строкового типу;
- ***довжина*** — константа цілого типу, яка задає максимально допустиму довжину рядка.

Приклад оголошення змінних строкового типу:

```
name: string[30];
buff: string;
```

Якщо в оголошенні строковій змінній довжина рядка не вказана, то її довжина задається рівною 255 символам, тобто оголошення stroka: **string [255];** і stroka: **string;** еквівалентні.

У тексті програми послідовність символів, що є рядком (строковою константою), полягає в одинарні лапки. Наприклад, щоб привласнити строковій змінній parol значення, потрібно записати:

```
parol:= 'Великий секрет'; або
parol:= '2010';
```

Слід звернути увагу, що інструкція parol:=2010; невірна, оскільки тип константи не відповідає типу змінної. Під час компіляції цієї інструкції буде виведено повідомлення: incompatible types: 'Char' and 'Integer' (типи Char і Integer несумісні)(Додаток .

Використовуючи операції =, <, >, <=, >= і про, змінну типу string можна порівняти з іншою змінною типу string або із строковою константою. Рядки порівнюються посимвольний, починаючи з першого символу. Якщо всі символи порівнюваних рядків однакові, то такі рядки вважаються рівними. Якщо в однакових позиціях рядків знаходяться різні символи, більшим

вважається той рядок, у якого в цій позиції знаходиться символ з великим кодом. У табл. 3.1 приведені приклади порівняння рядків.

Таблиця 3.1-Порівняння рядків

Рядок 1	Рядок 2	Результат порівняння
Василенко	Василенко	Рядки рівні
Алексєєв	Петров	Рядок 1 менше рядка 2
Кучеренко	Кучер	Рядок 1 більше рядка 2

Окрім операції порівняння, до строкових змінних і констант можна застосувати операцію складання, в результаті виконання якої виходить новий рядок. Наприклад, в результаті виконання інструкцій

```
f irst__name: ='Сергій' ;
last_name:='Сергієнко';
ful_name:=first_name+last_name;
```

змінна ful_name набуде значення 'Сергій Сергієнко'.

Операції з рядками.У мові Delphi є декілька корисних при роботі з рядками функцій і процедур. Нижче приведений їх короткий опис і приклади використання (табл.3.1).

Функція **length** повертає довжину рядка. У цієї функції один параметр — вираз строкового типу. Значенням функції length (ціле число) є кількість символів, з яких складається рядок.

Наприклад, в результаті виконання інструкцій

```
n:=length('Іванов');
m:=length('Проспект Гурова');
```

значення змінних n і m буде рівне 6 і 15.

Процедура **delete** дозволяє видалити частину рядка. У загальному вигляді звернення до цієї процедури виглядає так:

```
delete(рядок, р, п);
```

де:

- *рядок* — змінна або константа строкового типу;
- *р* — номер символу, з якого починається підрядок, що видаляється;
- *п* — довжина підрядка, що видаляється.

Наприклад, в результаті виконання інструкцій

```
p:='Місто Санкт-Пітербург';
delete(s,7,6);
```

значенням змінної ***s*** буде рядок ' *місто Пітербург*'.

Функція **pos** дозволяє визначити положення підрядка в рядку. У загальному вигляді звернення до функції виглядає так:

pos (підрядок,рядок) ;

де *підрядок* — строкова константа або змінна, яку треба знайти в строковій константі або змінній рядок.

Наприклад, в результаті виконання інструкції

p := pos('Пі','санкт-петербург');

значення змінної *p* буде рівне *7*. Якщо в рядку немає шуканого підрядка, то значення функції **pos** буде дорівюватися нулю.

Нижче приведена інструкція **while**, в результаті виконання якої віддаляються початкові пропуски з рядка st.

while(pos(' ',st)= 1) and(length(st)> 0) do delete (st,1,1);

Пропуски видаляє інструкція delete (st, i, i), яка виконується в циклі до тих пір, доки першим символом рядка є пропуск (в цьому випадку значення pos (' ',st) дорівнює одиниці). Необхідність перевірки умови **length (st) > 0** пояснюється можливістю того, що введений рядок складається тільки з пропусків.

Функція **copy** дозволяє виділити фрагмент рядка. У загальному вигляді звернення до цієї функції виглядає так:

copy(рядок, р, п)

де *рядок* — вираз строкового типу, що містить рядок, фрагмент якого треба отримати;

p — номер першого символу, з якого починається підрядок, що виділяється;

n — довжина підрядка, що виділяється. Наприклад, в результаті виконання інструкцій

st:= 'Інженер Будько'; fam:=copy(st, 9, 6);

значенням змінної fam буде рядок 'Будько'.

Наступні функції працюють з параметрами повертаємого рядка.

Таблиця 3.1 - Короткий опис функцій, з якими працюють при роботі з рядками

Функція	Пояснення
AnsiLoverCase(const S: String): String	Повертає рядок S, перетворений у **нижній** регістр.
AnsiUpperCase(const S: String): String	Повертає рядок S, перетворений у **верхній** регістр.
Length(const S: String): Integer	Повертає кількість символів в рядку **S.**

Trim(const S: String): String	Видаляє з рядка **S** початкові та кінцеві пробіли та керуючі символи.
TrimLeft(const S: String): String	Видаляє з рядка **S** початкові та керуючі символи.
TrimRight(const S: String): String	Видаляє з рядка **S** кінцеві пробіли та керуючі символи.
AnsiCompareStr(const S1, S2: String): Integer	Порівнює два рядка **S1** і **S2** з урахуванням регістра символів. Повертає значення <0 якщо **S1**, 0 якщо **S1=S2**, >0 якщо **S1>S2**
AnsiCompareText(const S1, S2: String): Integer	Порівнює два рядка **S1** і **S2** без урахування регістра символів. Повертає значення <0 якщо **S1**, 0 якщо **S1=S2**, >0 якщо **S1>S2**
Pos(Substr: String; Str: String): Integer	Повертає позицію (індекс) першого вхождення **Substr** в рядку **Str**. Якщо **Substr** немає в **Str**, повертає 0.
Insert(Source: String; var S: String; Index: Integer): Integer	Вставляє рядок **Source** в рядок **S**, починаючи з номера символа, рівного **Index**
Delete(var S: String; Index, Count: Integer): Integer	Видаляє з рядка **S** підрядок, який починається з номера символа, рівного **Index**, і містить до **Count** символів.
opy(S: String; Index, Count: Integer): Integer	Повертає підрядок рядка **S**, починаючи з номеру символу, рівного **Index** і містить до **Count** символів.

Основні положення теми:

1. Для організації циклічних процесів використовуються оператори циклу **Repeat** (цикл з умовою поста) і **While** (цикл з передумовою.
2. Після слова **do** в операторові **While** дозволяється використовувати тільки один оператора. Якщо потрібно виконати множинну послідовність дій, оператори поміщаються в операторні дужки begin оператор1, оператор2, оператор3. end .
3. Оператор **For** використовується для організації циклу з відомим числом повторень, при цьому параметри циклу (початкове і кінцеве значення, змінна циклу) повинні бути описані як цілі.

4. Як параметри циклу в операторах **While** і **Repeat** можуть виступати як цілі, так і речові числа.
5. Для виведення інформації в декілька рядків можна використовувати компонент **Memo**.
6. Для роботи з масивами зручно використовувати компонент **StringGrid** (таблицю)**.**
7. Для організації меню використовують компонент **MainMenu**.
8. Для екстреного виходу з циклу застосовують оператора **Break** (тільки припинення циклу і продовження виконання основної програми) і оператор **Exit** (припинення виконання не тільки циклу, але і підпрограми).
9. Для зберігання і обробки символів використовуються змінні типу *Ansichar* і *wideChar*. Типом Ansichar є набір ANSI-символов, з якому кожен символ кодується восьмирозрядним двійковим числом (байтом). Типом wideChar є набір символів в кодуванні Unicode, в якому кожен символ кодується двома байтами. Для забезпечення сумісності з попередніми версіями підтримується тип Char.
10. Якого типу змінні використовуються для зберігання і обробки символів?
11. Рядки можуть бути представлені наступними типами: *shortstring (*короткий рядок*), Longstring* (довгий рядок) и *widestring* (широкий рядок). Розрізняються ці типи гранично допустимою довжиною рядка, способом виділення пам'яті для змінних і методом кодування символів.
12. **У** мові Delphi є декілька корисних при роботі з рядками функцій і процедур. Функція **length** повертає довжину рядка. Процедура **delete** дозволяє видалити частину рядка. Функція **pos** дозволяє визначити положення підрядка в рядку. Функція **copy** дозволяє виділити фрагмент рядка.

Тести для самоперевірки знань до теми 3

Завдання 1–4 мають по чотири варіанти відповіді, серед яких лише ОДИН ПРАВИЛЬНИЙ. Виберіть правильний варіант відповіді.

1. Укажіть правильно оформлений опис одномірного масиву:

а) massiv:array [2.5..10.5] of real;
б) temp:array (20..800) of integer;
в) v;array [1...10] of real;
г) koef:array [0..9] of integer;

2. Яку опцію властивості **Option** варто встановити для здійснення редагування вмісту таблиці:

а) goEditing
б) gocolmoving
в) gorowmoving
г) gocolsizing

3. Укажіть неправильно оформлений опис двовимірного масиву:

а) X:array[0..10,0..10] of integer;
б) type mas = array[0..3] of integer; var X:array[mas] of integer;
в) var X:array[0..2,0..2] of integer= ((0,1,2),(3,4,5),(6,7,8));
г) type mas = array[0..3] of integer; var X:array[0..10] of mas;

4. Виберіть правильний з погляду синтаксису варіант оператора **While**:

а) while begin <умова>do <оператори>end;
б) while <умова>do begin<оператори>end;
в) while <оператор1>do<умова>;
г) while <умова>;do<оператор1>;

Завдання 5–8 **мають два варіанти відповіді. Треба обрати ПРАВИЛЬНИЙ, щоб приведений вислів мав сенс.**

5. На головній діагоналі матриці розташовуються елементи, у яких не співпадають номери рядків і стовпців.

Так | Ні

6. Кількість рядків прямокутної матриці не дорівнює кількості стовпців.

Так | Ні

7. Елементи квадратної матриці, які розташовуються вище головної діагоналі мають номера рядків менше номерів стовпців.

Так | Ні

8. У загальному виді оператор **Repeat** має наступний формат:

Repeat;
<оператор1>;
<оператор2>;
<оператор3>;
until <умова>;

Так | Ні

Завдання 9–11 мають на меті встановлення відповідності. До кожного рядка, позначеного ЦИФРОЮ, доберіть відповідник, позначений БУКВОЮ.

9. Установите відповідність назви і позначення основних властивостей компонента **StringGrid:**

1. *Cells* А. Об’єкт
2. *Cols* Б. Визначає кількість стовпців
3. *Rows* В. Перелік колонок
4. *ColCount* Г. Комірка таблиці
5. *RowCount* Д. Перелік рядків
   Е. Визначає кількість рядків

	А	Б	В	Г	Д	Е
1						
2						
3						
4						
5						

10. Установите відповідність назви і позначення основних властивостей компонента **StringGrid:**

1. *FixedCols* А. Колір фона фіксованих комірок
2. *FixedRows* Б. Індекси першого видимого стовпца
3. *FixedColor* В. Кількість стовпців, які не прокручуються
4. *Scrollbars* Г. Кількість рядків, які не прокручуються
5. *LeftCol* Д. Наявність смуг прокрутки
   Е. Індекси першого видимого рядка

	А	Б	В	Г	Д	Е
1						
2						
3						
4						
5						

11. Установите відповідність назви і позначення основних властивостей компонента **RadioGroup1**:

1. *Caption* А.Заголовок кнопки
2. *Items* Б. Рівень розміщення радіокнопок
3. *Itemindex* В. Напис в лівому верхньому кутку групи
4. *Columns* Г. Яку з кнопок обрав користувач

Д. Колір кнопки

	А	**Б**	**В**	**Г**	**Д**
1					
2					
3					
4					

Питання для самоконтролю

1. Який обчислювальний процес називають циклічним?
2. Назвіть відомі вам типи циклів.
3. Для організації яких циклічних процесів використовуються оператор **Repeat** ? Назвіть параметри циклу цього оператора.
4. Для організації яких циклічних процесів використовуються оператор **While**? Назвіть параметри циклу цього оператора.
5. Скільки разів виконується оператор циклу з передумовою?
6. Як здійснюється вихід з оператора циклу з передумовою?
7. Поясніть структуру оператора циклу з передумовою?
8. Поясніть порядок виконання циклу з передумовою?
9. Назвіть параметри циклу оператора **For.** Для яких циклів він використовується.
10. Що таке вкладені цикли?
11. Скільки разів виконується оператор циклу з параметром.
12. Що називається параметром циклу, початковим і кінцевим значеннями параметра циклу?
13. Поясніть порядок виконання циклу з параметром.
14. Поясніть призначення і застосування процедур Break і Continue.
15. Що є тілом циклу оператора циклу з постумовою?
16. Скільки разів виконується оператор циклу з постумовою?
17. Як здійснюється вихід з оператора циклу з постумовою?
18. Поясніть порядок виконання оператора циклу з постумовою.
19. Якого типу змінні використовуються для зберігання і обробки символів?
20. Що таке рядок?
21. Як оголошуються змінні для зберігання рядків?
22. Які типи рядків існують в Delphi і в чому їх відмінності?
23. Опишіть функції для роботи з рядками?

24. За допомогою яких функцій здійснюється переклад рядків з інші типи?

25. Як звернутися до окремого елементу рядка?

4 ГРАФІЧНЕ ПРЕДСТАВЛЕННЯ ІНФОРМАЦІЇ В СЕРЕДОВИЩІ DELPHI

Властивість Canvas

Delphi дозволяє розробляти програми, які можуть виводити на екран графіку: схеми, креслення, ілюстрації, створювати власні зображення або додавати в програму готові графічні файли.

Для створення власних графічних зображень використовується властивість **Canvas**. Такі об'єкти Delphi як форми, **Image**, **BitMap** мають цю властивість. Для виведення на екран графічних елементів (прямої лінії, кола, прямокутника і т.д.), необхідно застосувати до властивості **Canvas** цього об'єкту відповідний метод. Властивості дозволяють задавати графічним зображенням певні характеристики: колір, товщину і стиль ліній; колір і вид заповнення областей; характеристики шрифту при виведенні текстової інформації.

Методи виведення графічних примітивів розглядають властивість **Canvas** як деяке абстрактне полотно, на якому вони можуть малювати. Canvas перекладається з англійської мови як «полотно», «полотно для малювання». Для того, щоб намалювати що-небудь на формі або компоненті (Canvas підтримують багато компонентів), нам потрібно уміти задавати положення на екрані того, що малюємо. Для цього з канвою зв'язується система координат наступного виду (рис.4.1):

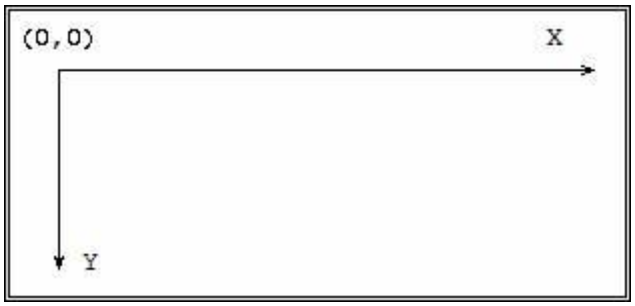


Рисунок 4.1 – Система координат методу Canvas

Полотно складається з окремих крапок — пікселів. Кожна крапка насправді є дуже маленьким прямокутником. І оскільки це не зовсім крапка, то використовуються термін — “піксел”.

Положення пікселя характеризується його горизонтальною (X) і вертикальною (Y) координатами. Крапка (0,0) знаходиться в лівому верхньому кутку. Звична для нас система координат “перевернута”: координати збільшуються зверху вниз і зліва

направо, тобто, якщо вісь X направлена зліва направо, як це прийнято при побудові математичних графіків, то вісь У - зверху вниз (У збільшується). Значення координат правої нижньої точки полотна залежать від розміру полотна. Наприклад, якщо малюють на формі, то це її ширина і висота.

Коли ми починаємо малювати, то беремо в руки олівець і наносимо на папір контур малюнка, а потім кистю або фломастером фарбуємо зображення. Аналогічний підхід реалізований при використанні властивості **Canvas**.

Олівець **Pen** застосовується для викреслювання ліній і контурів на полотні, а кисть **Brush** — для зафарбування областей, обмежених контурами. Якщо для них не задані властивості, то малюнок створюється чорним олівцем завтовшки в один піксель і заливка відбувається білим кольором.

Щоб задати колір олівця використовують його властивість кольору *Color*.

Pen.Color — визначає колір, яким відбувається креслення. Для зміни кольору використовують оператора

```
Canvas.Pen.Color := rgb(255,0,0) ;
```

де вибір кольорів здійснюється за технологією **RGB**, описаною в першому розділі. У нашому прикладі вибирається олівець червоного кольору.

При створенні зображення спочатку задають колір і, при необхідності, товщину олівця, а потім вже малюють.

Лінії можуть бути суцільними або такими, що складаються з крапок і пунктирів.

Pen.Style — визначає стиль, яким малюються лінії й має настіпні варіанти:

————	***psSolid***	Суцільна лінія
— — — — -	***psDash***	Лінія з серії розривів
.....	***psDot***	Лінія з крапок
– - – - – - – -	***psDashDot***	Лінія з розривів і крапок
–..–..–..–..	***psDashDotDot***	Лінія вигляду крапка, крапка, розрив
	psClear	Лінія не зображається

Наприклад,

Canvas.Pen.Style := psDashDotDot;

задає зображення фігури олівцем у вигляді двох крапок і розриву.

Олівець може бути товстим або тонким.

Pen.Width – задає товщину олівця, наприклад,

Canvas.Pen.Width:=5;

Якщо значення властивості **Canvas**.**Pen.Width** більше одиниці, то пунктирні лінії будуть виведені як суцільні.

Властивість **Brush** дозволяє заповнити кольором і штрихуванням області, включаючи внутрішню частину форм.

Canvas.brush.color:=rgb(0,255,0); задає зелений колір, що заповнює фігуру.

Brush.Style задає стиль штрихування.

Можливі варіанти заповнення областей показані на рис. 4.2.

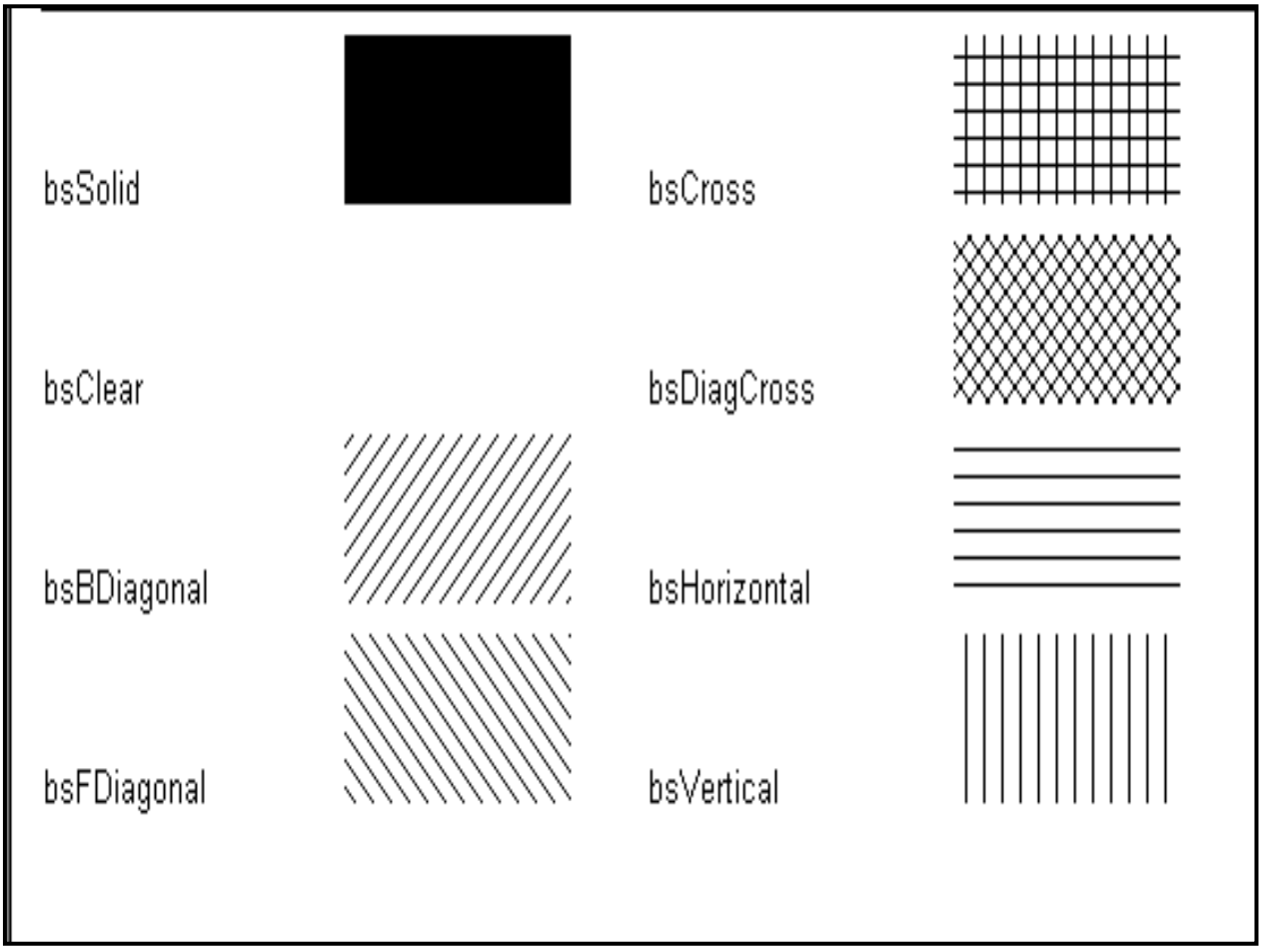


Рисунок 4.2 – Завдання стилів штрихування при заповненні областей

Оператор ***Canvas.brush.style := bsDiagCross;*** задає заповнення області в вигляді решітки з діагонально нахиленими лініями.

Оператор ***Canvas.brush.style:= bsVertical;*** заповнює область вертикальними лініями.

Один із способів створення зображень – це малювання пікселями. Кожна крапка (піксель) характеризується координатами і кольором, оператор ***Canvas.Pixels[X,Y]:= rgb(0,0,0);*** задає крапку з координатами X і У чорного кольору, де X - номер пікселя по горизонталі, а У – по вертикалі.

На рис. 4.3 показані різні варіанти заповнення областей компоненту **Image** за допомогою пікселів. Нижче приводиться текст програми .

```
procedure TForm1.Button1Click(Sender: TObject);
var i,j,k,f:integer;
begin
        {Малювання відбувається в об'єкті Image3 }
with Image3.Canvas  do begin
    f:=0; i:=0 ;  {Заповнення кольором області 1, рис.4.3}
while i<=100  do      begin
  {Змінна і визначає позицію пікселя по вертикалі (У)}
       j:=0;  k:=0;   f:=f+1;
             while j<=100   do
 {Змінна j визначає позицію пікселя по горизонталі (X)}
        begin
                    k:=k+1; if (k <= 9) and (f <=9) then
          {Задається величина чорних квадратів}
          Pixels[j,i]:=rgb(0,0,0) {Задається колір квадратів}
else
              if k>17 then k:=0;   j:=j+1;
              if f>17 then f:=0; end;
              i:=i+1;
        end;
while i<=100  do
          {Заповнення кольором області 2, рис.4.3}
 begin
   j:=100;
 while j<=200   do
  begin
      k:=k+1; if (k <= 9)  then
Pixels[ j,i]:=rgb(0,0,0) else if k<17 then k:=0;
 j:=j+1;
```

```
    end;
    i:=i+1;
    end;
          {Заповнення кольором області 3, рис.4.3}
k:=0; i:=0;     j:=100;
while i<=100  do
   begin
   j:=100;
 while j<=200   do
   begin
     Pixels[ i,j]:=rgb(0,0,0) ; j:=j+1;
 end;
   i:=i+10;
end;
while i<=200  do
          {Заповнення кольором області 4, рис.4.3}
        begin
j:=100;
while j<=200   do
      begin
    if i=j then
 Pixels[ j,i]:=rgb(255,255,255)
    else
Pixels[j,i]:=rgb(0,0,0);
  j:=j+7;
  i:=i+1;
                end;  end;  end;    end;
```

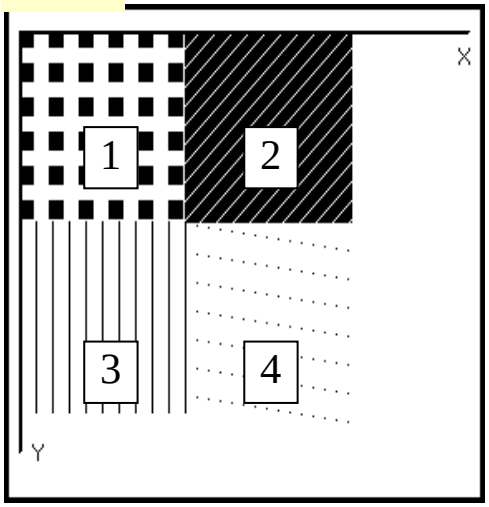


Рисунок 4.3 - Варіанти використання властивості Pixels

Як видно з рис. 4.3, використання пікселів дозволяє створювати різні варіанти точечної заливки областей або створювати точечні зображення.

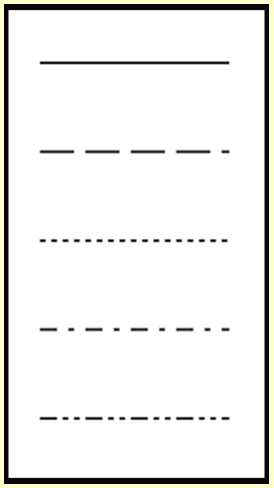
Canvas.MoveTo(x, y); - використовується для установки олівця в позицію, визначену координатами *x, y*, де *x, y* — цілі числа. Його можна використовувати в комбінації з **LineTo(x1, y1)**;

властивістю, яка малює лінію від позиції олівця **(x, y)** до позиції **(x1, y1)**.

Уявіть собі, що Ви збираєтеся створити малюнок за допомогою олівця, причому, не відриваючи його від листа паперу. Спочатку олівець ставиться в певну позицію, для цього використовується властивість **Canvas.MoveTo(x, y),** а потім кресляться лінії від однієї точки до іншої із застосуванням **LineTo**, причому початковою точкою лінії буде кінцева точка попередньої лінії.

Приклад побудови ліній показаний нижче. У програмі організований цикл, що дозволяє за допомогою оператора **Case** перебирати можливі варіанти стилю олівця. Малюється п'ять ліній одиничної товщини, зміщених по вертикалі на 40 пікселів один від одного.

```
procedure TForm1.N12Click(Sender: TObject);
var i:integer;
begin  for i:=0 to 4 do    begin
case i of
   0: Canvas.Pen.Style := psSolid;
   1: Canvas.Pen.Style := psDash;
   2: Canvas.Pen.Style := psDot;
   3: Canvas.Pen.Style := psDashDot;
   4:Canvas.Pen.Style:=sDashDotDot;
    Canvas.moveto(50,50+i*40 );
    Canvas.lineto( 150, 50+i*40);
 end;
end;
```



Для зображення дуг використовується властивість **Arc** з параметрами (X1, Y1, X2, Y2, X3, Y3, X4, Y4);

Arc малює дугу, що є частиною еліпса, обмеженого крапками (X1,Y1) і (X2,Y2). Дуга малюється проти годинникової стрілки від початкової точки, яка визначається перетином еліпса і крапки (X3,Y3) і кінцевої точки (X4, Y4).

Arc(x1,y1,x2,y2,x3,y3,x4,y4: Integer)

Метод **Ellipse** малює коло або еліпс за допомогою поточних параметрів пера **Pen**. Фігура заповнюється поточним значенням **Brush.** На рис. 4.4 показаний приклад побудови еліпса, який

вписаний в прямокутник з координатами верхньої лівої точки (X1, Y1) і правої нижньої крапки (X2, Y2).

Ellipse (x1,y1,x2,y2:Integer);

Прямокутник, в який вписується еліпс, зображений пунктиром. Якщо точки прямокутника формують квадрат, то зображається круг.

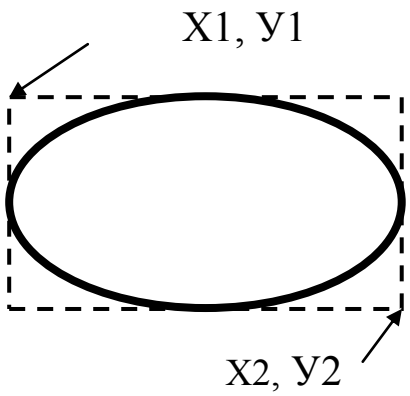


Рисунок 4.4 – Схема задання координат прямокутника

Для зображення прямокутників і квадратів використовують метод **Rectangle:**

Rectangle(X1, Y1, X2, Y2);,

причому координата верхньої лівої точки (X1, Y1) і правої нижньої крапки (X2, Y2) (рис. 4.3).

Щоб зобразити закруглений прямокутник, використовується метод **RoundRect**(X1, Y1, X2, Y2, R1, R2), використовуючи поточні установки пера Pen і заповнюючи площу фігури поточним гроном Brush.. Перші чотири параметри, передавані в **RoundRect**, є координатами прямокутника, наступні два параметри указують радіуси закруглення кутів.

RoundRect(x1,y1,x2,y2,x3,y3:Integer);

Якщо задати ширину еліпса R1=x2-x1, то верхня і нижня межі рамки виявляться такими, що цілком округляють (без прямолінійної частини). Якщо R2=y2-y1, те ж саме відбудеться з лівою і правою межами рамки. Якщо ж обидва вимірювання еліпса не менше розмірів рамки, то малюватиметься просто еліпс. Але, звичайно, для малювання еліпса краще використовувати метод **Ellipse**. Якщо один з розмірів еліпса задати нульовим, то малюватиметься прямокутна рамка, а для отримання такої рамки краще використовувати метод **Rectangle**.

Ломану замкнуту лінію можна зобразити за допомогою методу **Polygon:**

Polygon([Point(x1, y1),Point(x2, y2),Point(x3, y3)...Point(xn, yn)])

Point визначає координати крапок, що входять в лінію. Якщо зображення створюється за допомогою **Lineto**, то його внутрішню

частину не можна фарбувати, використовуючи **Brush.color**. Метод **Polygon** дозволяє задавати зафарбовування внутрішньої частини зображення.

Метод **Polyline** малює на канві поточним пером кусочно-лінійну криву по заданій безлічі крапок, заданій масивом Points:

Polyline(Points: array of TPoint);

Відмінність методу Polyline від методу Polygon полягає в тому, що метод Polygon замикає кінцеві крапки, а метод Polyline — ні. Малювання проводиться пером Pen. Метод не змінює поточної позиції PenPos пера Pen.

Те що робить метод Polyline, можна зробити і за допомогою методів MoveTo і LineTo, підвівши сначало перо до першої крапки, а потім послідовно виконуючи LineTo. Відмінність полягатиме в тому, що метод Polyline не змінить поточну пера, а методи MoveTo і LineTo змінять.

Метод **Polyline** ([Point(x1, y1), Point(x2, y2), Point(x3, y3) ... Point(xn, yn)]) має ті ж параметри, але внутрішню частину зображення не закрашують.

Метод **Pie** малює замкнуту фігуру — сектор кола або еліпса, за допомогою поточних параметрів пера Pen. Фігура заповнюється поточним значенням Brush.

Pie (X1,Y1,X2,Y2,X3,Y3,X4,Y4:Longint);

Точки (X1,Y1) і (X2,Y2) визначають прямокутник, що описує еліпс. Початкова точка дуги визначається перетином еліпса з прямою, що проходить через його центр і точку (X3,Y3). Кінцева точка дуги визначається перетином еліпса з прямою, що проходить через його центр і точпку (X4,Y4). Дуга малюється проти годинникової стрілки від початкової до кінцевої крапки. Малюються прямі, що обмежують сегмент і що проходять через центр еліпса і крапки (X3,Y3) і (X4,Y4).

Метод **Chord** (X1, Y1, X2, Y2, X3, Y3, X4, Y4) використовується для створення замкнутих фігур, обмежених еліпсом з координатами (X1, Y1, X2, Y2) і лінією (X3, Y3, X4, Y4).

Chord (x1,y1,x2,y2,x3,y3,x4,y4:Integer);

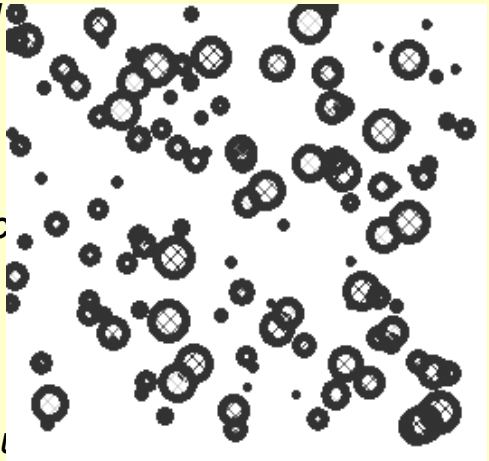
Розглянемо використання властивості **Canvas** на прикладах. Нами розроблена програма, яка виконує наступні функції:

✓ При включенні таймера кожні п'ять мілісекунд малюється круг з довільними координатами.

✓ Якщо кількість намальованих кругів більше 250, то збільшується інтервал роботи таймера і збільшується радіус намальованих кругів. Якщо шукати аналогію, то можна уявити, що спочатку йде швидко дрібний сніг. А потім крупні сніжинки падають все повільніше. Якщо кількість зображень перевищила 350, відбувається очищення екрану.

✓ Вибір стилю і кольору відбувається випадковим чином.

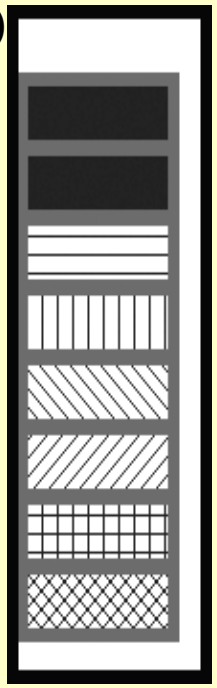
```
procedure TForm1.Timer1Timer(Sender: TObject);
Var   X, Y, S,R: Integer;
begin
Timer1.Interval := 5;
Randomize;
k:=k+1;  {Лічильник кількості кругів}
s:=10; { Величина для набуття
випадкового значення радіусу}
 if k>350 then
begin
k:=0; refresh ;  end
{Очищення екрану, якщо кількіс
об'єктів перевищує 350 }
 Else
 if k>250 then
 begin
{Якщо кількість об'єктів перев
інтервал таймера і розмір радіусу}
 timer1.Interval:=10; s:=20; end;
{Вироблення випадкових значень координат}
X:= Random(form1.Width — 200);
Y:= Random(form1.Height — 200);
Canvas.Pen.Color:= Rgb(0,0,0);
 Canvas .Pen.width:= 1;
 if k mod 3=0 then    Canvas.brush.Color:=rgb(255,0,0)
else
if k mod 3=1 then  Canvas.brush.Color:=rgb(0,255,0)
else        Canvas .brush.Color:=rgb(0,0,255) ;
{В залежності  від значення  k круги  будуть червоного ,
зеленого або синього кольору}
        case k of
{Випадковий стиль заливки}
          1: Canvas .brush.style := bsSolid   ;
          2: Canvas .brush.style := bsClear  ;
```



```
            3: Canvas .brush.style := bsHorizontal  ;
            4:  Canvas .brush.style := bsVertical   ;
            5:  Canvas .brush.style := bsFDiagonal   ;
            6:  Canvas .brush.style := bsBDiagonal   ;
            7:  Canvas .brush.style := bsCross   ;
            8: Canvas .brush.style := bsDiagCross   ;    end;
  r:=1+ Random(s); {Випадкова величина радіусу}
    Canvas .ellipse(X, Y, X + r,y+r);  {Малюється круг}
end;
```

Ще одна програма демонструє зображення декількох прямокутників з різним типом заливки. Кожен прямокутник обведений чорною лінією з товщиною, рівною шести.

```
procedure TForm1.N10Click(Sender: TObject);
var i:integer;
begin     timer1.Enabled:=false;
for i:=1 to 8 do    begin
case i of
  1: Canvas.brush.style := bsSolid;
  2: Canvas.brush.style := bsClear;
  3: Canvas.brush.style := bsHorizontal;
  4: Canvas.brush.style := bsVertical;
  5: Canvas.brush.style := bsFDiagonal;
  6:  Canvas.brush.style := bsBDiagonal;
  7:  Canvas.brush.style := bsCross;
  8:  Canvas.brush.style := bsDiagCross;    end;
      Canvas.brush.Color:=rgb(0,0,255);
      Canvas.pen.width:=6;
      Canvas.pen.color:=rgb(0,0,0);
      Canvas.rectangle(10,50+i*30,90,i*30+ 80);
end; end;
```



4.2 Виведення тексту в графічному режимі

Для виведення тексту на поверхню графічного об'єкту використовується метод **TextOut**. Оператор виклику методу **TextOut** виглядає таким чином:

Canvas.TextOut(X, Y, Текст)

де:

✓ *X, Y* — координати точки графічної поверхні, від якої виконується виведення тексту.

✓ *Текст* — змінна або константа символьного типу, значення якої визначає текст, який виводиться зазначеним методом.

Шрифт, який використовується для виведення тексту, визначається значенням властивості **Font** відповідного методу **Canvas**.

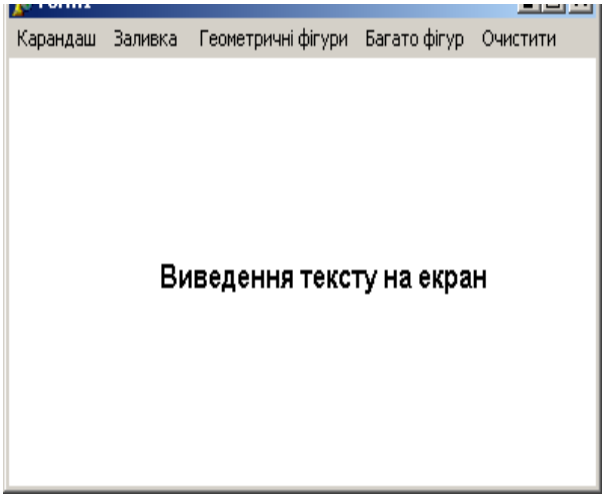
У табл.4.2 задані властивості *TFont*.

Таблиця 4.2 - Властивості TFont

Назва	Опис властивості
Name	Задається назва шрифту, наприклад, Arial**.**
Size	Розмір шрифту в пунктах (points). Пункт— це одиниця вимірювання розміру шрифту, використовувана в поліграфії. Один пункт рівний 1/72 дюйми.
Style	Стиль зображення символів: нормальним, напівжирним (fsBold), курсивним (fsltalic), підкресленим (fsUnderline), перекресленим (fsStrikeOut). Властивість Style дозволяє комбінувати необхідні стилі. Наприклад, оператор програми, який встановлює стиль «напівжирний курсив», виглядає так: **Canvas.Font. Style : = [fsBold, fs Italic];**
Color	Завдання кольору шрифту, наприклад, **Canvas. Font.Color:=RGB(255,0,0)**; – букви червоного кольору.

Нижче приводиться текст програми, яка виводить на екран надпис і результати її роботи.

```
procedure TForm1.N7Click(Sender: TObject);
begin   Canvas . font.name:='Arial';
 Canvas . font.size:=12;
    Canvas . font.style:=[fsBold];
    Canvas . font.color:=rgb(255,0,0)  ;
Canvas    .TextOut(100,100,'Виведення тексту на
екран');
```




```
End;
```

Текст виводиться шрифтом червоного кольору, розміром 12, жирного стилю Arial.

Для очищення екрану від зображень використовується команда **Refresh**.

4.3 Побудова графіків функцій

Досить часто результати розрахунків зручно представити у вигляді графіка. Для більшої інформативності і наочності графіки зображають на тлі координатних осей і оцифрованої сітки.

На рис.4.3 показана форма з об'єктами, необхідними для побудови графіка. Нами використаний компонент **MainMenu** для організації меню, що складається з трьох пунктів: «**Розрахунок даних»** для визначення координат точок графіка; «**Побудова графіка»** як основна частина програми і пункту «**Вихід»** для завершення роботи з програмою.

Для побудови графіка використовується раніше нам невідомий компонент **Image** панелі **ADDITIONAL**. Він використовується як полотно для малювання за допомогою методу **Canvas**, для завантаження графічних файлів, якщо вони вставляються в програму із зовнішнього джерела. Основною властивістю є властивість *Picture*, яка визначає який саме файл вставляється в додаток.

У нашому випадку не буде використано цю властивість, оскільки ми створюємо зображення, а не вбудовуємо готовий файл.

Для введення початкових, кінцевих значень X і кроку змінювань узято три компоненти **Edit** (рис. 4.5).

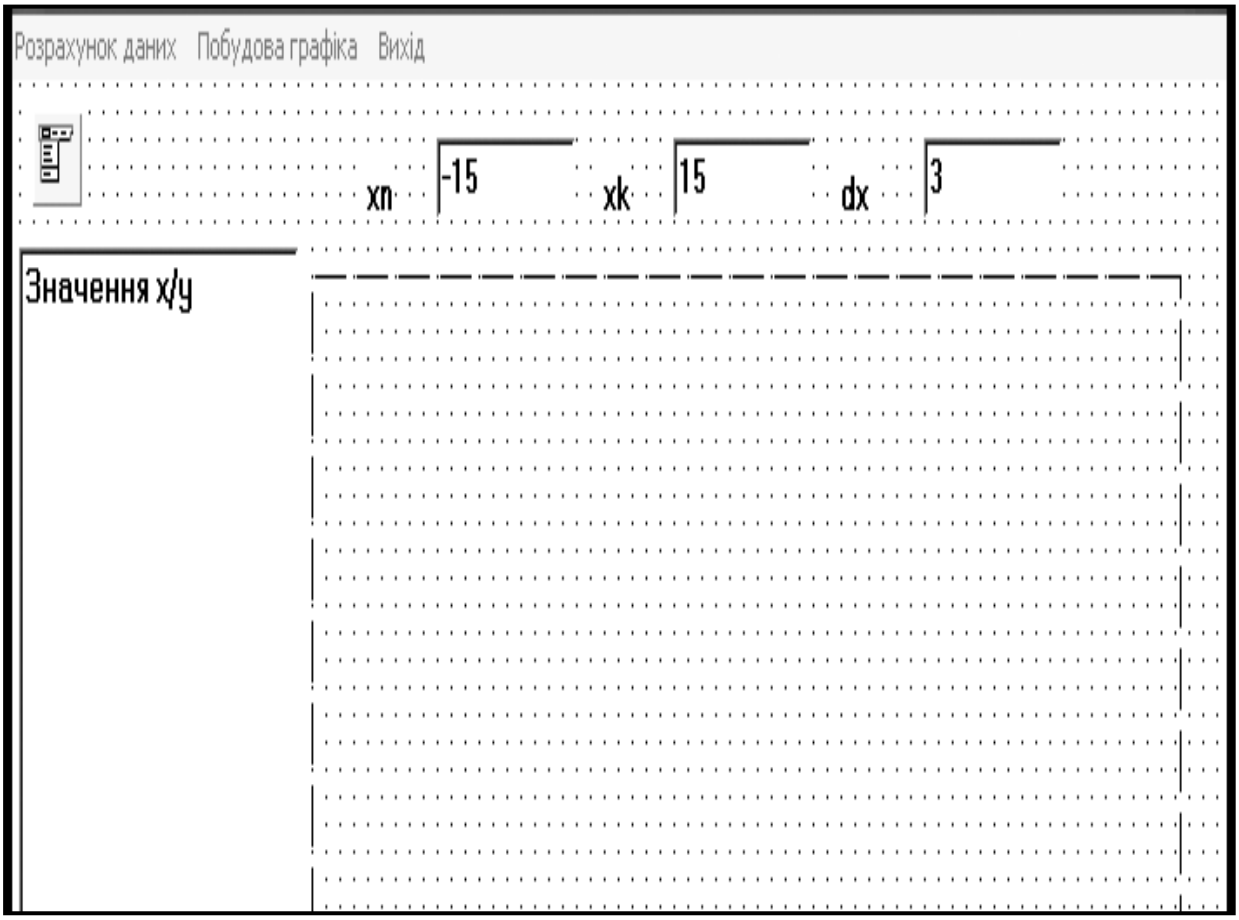


Рисунок 4.5 - Форма для побудови графіка

Якби будували графік ручним способом на міліметрівці, то спочатку б порахували значення в кожній точці, щоб вибрати масштаб, і подивитися, в яких межах змінюються дані. Для цієї мети в програму введений пункт меню «**Розрахунок даних**».

При натискуванні по цьому пункту меню виконуються наступні дії:

✓ Вводяться початкове, кінцеве значення X і крок змінювань, що визначають в якому інтервалі осі абсцис будується графік. Дані перетворюються в числові значення. Це організовано за допомогою операторів:

xn:=strtoint(edit1.Text); xk:=strtoint(edit2.Text);
dx:=strtoint(edit3.Text);

✓ Значення У, які відкладатимуться по осі ординат, записуються в масив. Для цього вводиться змінна *i*, що визначає порядковий номер елементу в масиві. При x:=xn номер елементу

масиву рівний одиниці, кожному наступному x відповідає черговий елемент масиву У.

✓ За допомогою оператора while x<=xk do організований цикл, що дозволяє перебрати всі значення X із заданим кроком. Оскільки в циклі виконуються чотири оператори, після оператора **do** введена операторна дужка *begin – end*.

✓ У тілі циклу відбувається розрахунок значень масиву У, вивід їх на екран, а також зміну порядкового номера елементу масиву i:=i+1;і зміна значення x на введений крок x:=x+dx;.

✓ Після завершення циклу визначається кількість точок графіка за формулою n:=i-1;

Текст програми процедури, компільований при виборі пункту меню «**Розрахунок даних»** приводиться нижче**.**

Нами будується графік функції

$$y = 3.2 \cdot (\frac{x}{2})^2 - 9 \cdot \frac{x}{8} + 20 \cdot \sin(x) - 50$$

```
procedure TForm1.N1Click(Sender: TObject);
  var i:integer;   xx:string;
begin
        {Введення початкових значень для завдання
                        інтервалу X}
xn:=strtoint(edit1.Text);  xk:=strtoint(edit2.Text);
dx:=strtoint(edit3.Text);
  x:=xn;    i:=1;
  while x<=xk do
begin
                {Розрахунок значень масиву У}
y[i]:= 3.2*sqr(x/2 )-9* (x/8)  +20*sin(x)-50;
  str( y[i]:1:1,xx);
            {Виведення елементів масиву на екран}
memo1.lines.add('x ='+ inttostr(x)+  '  y  ='+xx);
  i:=i+1;
  x:=x+dx;
       end;
      n:=i-1;
            {Розрахунок кількості точок на графіці}
 end;
```

На рис. 4.6 показані результати розрахунків точок для подальшої побудови графіка.

«**Побудова графіка».** При натискуванні по цьому пункту меню виконується ряд дій. Розглянемо їх поетапно.

Зображення графіка створюватиметься на компоненті **Image** за допомогою методу **Canvas.** Це приведе до того, що при замовленні будь-яких опцій, наприклад, кольору олівця, доведеться набирати **image1.Canvas.pen.Color:=rgb(0,0,0)**. Щоб уникнути записів, що повторюються, введемо нового оператора **With**. Це оператор приєднання, використовуваний для скорочення записів звернення.

Оператор **With** має формат:

With [об’єкт] **do begin** [оператори] **end**;

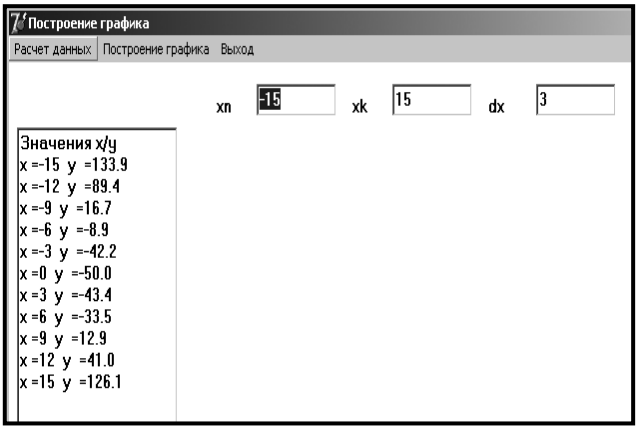


Рисунок 4.6 - Результати розрахунку точок

У операторах, які слідують за словом **do**, можна не указувати назву об'єкту. Ця назва автоматично приєднуватиметься, причому тільки до тих операторів, до яких це приєднання необхідне.

Для побудови вісей слід визначити центр системи координат.

Нам доведеться працювати в двох системах координат: системі координат комп'ютера і звичайною Декартовою. Графік повинен зовні бути звичайним графіком, побудованим в Декартовій системі координат, але будувати його ми будемо ґрунтуючись на системі координат комп'ютера.

Аналізуємо наявні дані. Значення змінних X і У можуть бути і від’ємними, і додатніми як по осі X, так і по осі У. В цьому випадку логічно розташувати центр Декартової системи координат в центрі компонента **Image**. Визначимо координати його центру. Для цього проводимо розрахунки.

xc:=image1.width div 2; yc:=image1. height div 2;

Нагадаємо, що операція **div** є операцією цілочисельного ділення. Отже, якщо ширина компоненту **Image** дорівнювала, припустимо, 301 пікселю, а висота 400, то центр розташовуватиметься в точці з координатами (150, 200) від лівого краю **Image**. Це означає, що центр координат розташовуватиметься в центрі компоненту **Image**. В центрі координат $x_c=0$ і $y_c=0$, але в пікселях це буде відстань, рівна половині ширини компоненту по ширині і по висоті відповідно.

Оператори *pen.Width:=5;* і *pen.Color:=rgb(0,0,0);* задають зображення координатних осей чорного кольору товщиною в п'ять пікселів.

Для зображення осей використані наступні оператори:

```
moveto(0,image1.height div 2);
lineto(image1.width,  image1.height div 2 );
moveto(image1.width div 2,0);
lineto(image1.width div 2,image1.height );
```

Спочатку олівець за допомогою

moveto(0,image1.height div 2);

ставиться у відповідну точку екрану, а потім за допомогою оператора **lineto(image1.width, image1.height div 2);** протягується по всій ширині **Image**. Аналогічно ставимо олівець у верхню крапку **moveto(image1.width div 2,0);** і протягуємо по всій висоті. На рис. 4.7, що приводиться нижче, показано чотири координати, що визначають початкові і кінцеві точки осей.

Для зображення масштабної сітки змінюється товщина олівця і стиль. Нами обирається олівець товщиною в один піксель, чорного кольору і пунктирного стилю.

pen.Width:=1; pen.Color:=rgb(0,0,0);
pen.Style:=psDot;

Визначимо, з якою частотою повинна виводитися на екран координатна сітка. Припустимо, ліворуч і праворуч від осі ординат будується однакова кількість вертикальних ліній. Тоді їх кількість визначають оператором m:=xc div 5;

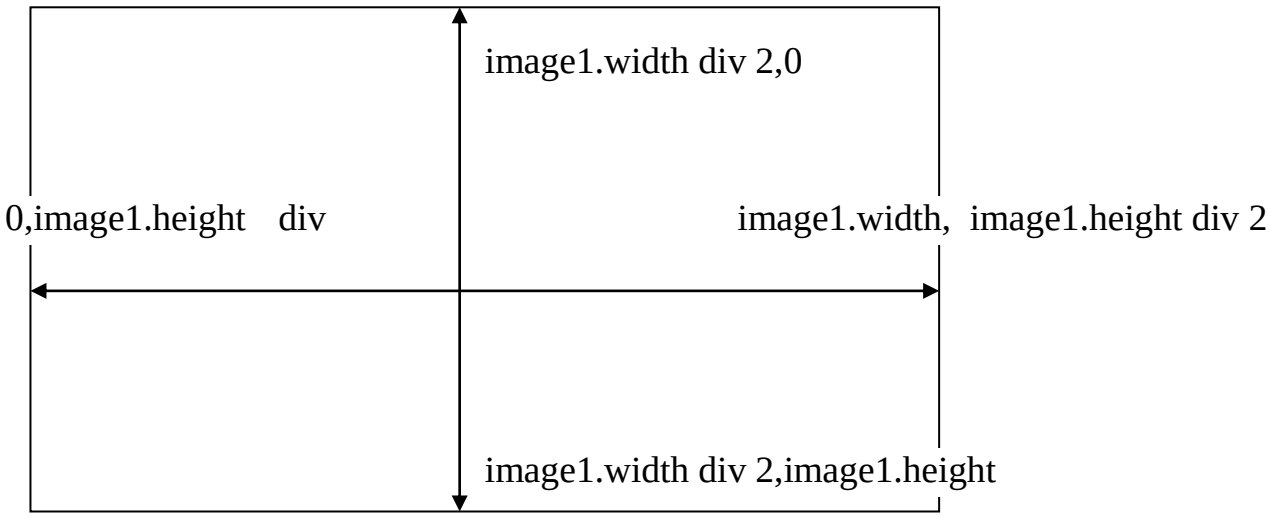


Рисунок 4.7 – Побудова координатних осей

Цикл створює зображення масштабної сітки. Результат показаний на рис. 4.8.

```
for i:=0 to 8 do begin
moveto(xc+i*m,  0);
lineto(xc+m*i  , height );
moveto(xc—i*m,  0);
lineto(xc—m*i  ,height );
moveto(0,yc—m*i);
lineto(2*xc,  yc—m*i );
 moveto(0,yc+m*i);
lineto(2*xc,  yc+m*i );
end;
```

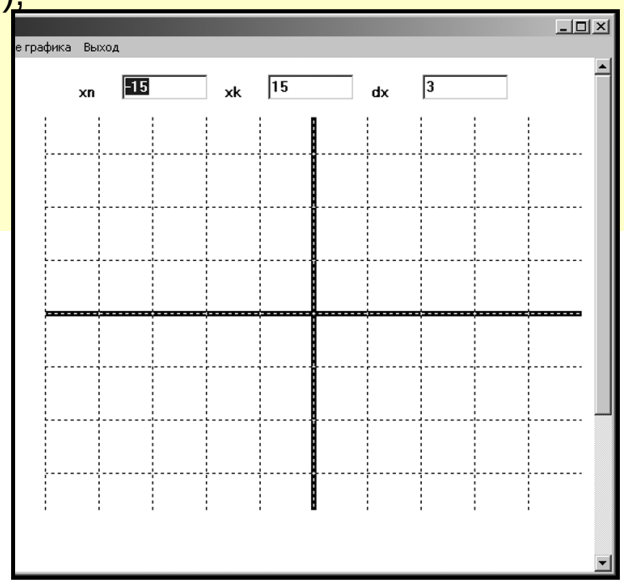


Рисунок 4.8 – Побудова координатної сітки

У лівій частині екрану в компоненті **Memo** вказані значення X і У, на підставі яких будуватиметься графік, в правій – побудована вісь координат і масштабна сітка.

Ідея побудови безпосередньо графіка полягає в наступному. Графік будують за допомогою набору ліній. Спочатку олівець встановлюється в першу точку за допомогою оператора **moveto**, а потім в циклі відбувається додавання по одній лінії, що сполучає сусідні точки графіка. Якщо був заданий невеликий крок, то зображення графіка виглядатиме як побудоване по крапках, якщо крок заданий великим, то вийде шматкова крива. Заздалегідь задається колір графіка і товщина його лінії.

Необхідним при побудові графіка є визначення масштабу. Уявіть собі, що будується графік синуса. Як відомо, значення У варіюються в межах від -1 до 1. Якщо залишити побудову графіка без масштабу, то ми не зможемо побачити графік, бо не зможемо побачити один піксель то вищий за вісь, то нижчий від неї. Отже, при необхідності, слід підбирати масштаб для побудови графіка. Нами визначався масштаб для осі Х за допомогою оператора*:*

mx:=trunc(2*xc/(xk—xn));

Зверніть увагу на використану функцію **trunc**. При побудові графічних зображень всі координати повинні бути цілими, функція **trunc** і проводить перетворення речових даних до цілого типу.

Координата точки графіка визначається таким чином: (x*mx+xc, yc- trunc(y[i])) – до центру координат додають значення Х, помножене на величину масштабу. Аналогічно з координатою по осі У, за винятком того, що значення віднімається. Це пов'язано з тим, що номери пікселів збільшуються зверху вниз, а не навпаки. Нульове значення знаходиться у верхньому лівому кутку екрану, а найбільше - в його нижньому правому кутку. Нижче приводиться фрагмент програми і результати її виконання (рис. 4.9).

```
        x:=xn;  i:=1;
        pen.color:=rgb(0,0,255);
        pen.width:=8;
        mx:=trunc(2*xc/(xk—xn));
        moveto(x*mx+xc,yc-
trunc(y[i]));
        while x<=xk do begin
        px:= x*mx +xc;
         py:=yc—trunc(y[i]);
         lineto (px,py);
        x:=x+dx;  i:=i+1;
        end;
```

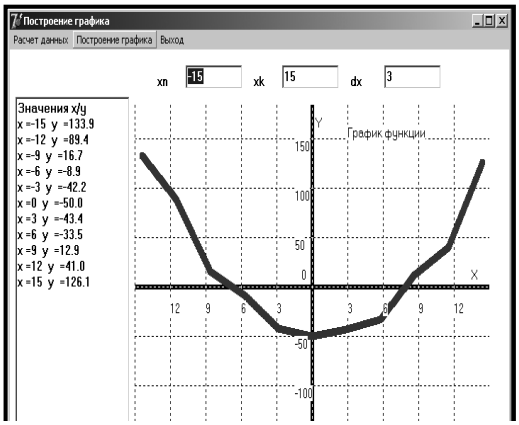


Рисунок 4.9 - Результати виконання програми

І на останок. Слід ввести підписи осей і назву графіка. Використовуємо метод **Textout.** Нами задана висота букв і їх колір. З урахуванням масштабу відкладалися написи на масштабній сітці.

```
font.Size:=10; font.color:=rgb(0,0,0);
textout(xc+5,10, 'Y');
textout(2*xc—25,yc—20, 'X');
textout( xc—15,yc—20, '0');
textout(xc+50,20,'Графік функції');
for i:=1 to 3 do
textout(xc—24,yc—m*i, inttostr(50*i));
for i:=1 to 4 do
textout(xc—24,yc+m*i, inttostr(—50*i));
for i:=1 to 4 do
textout(xc+m*i,yc+14, inttostr(3*i));
for i:=1 to 4 do
textout(xc—m*i,yc+14, inttostr(3*i));
```

Оскільки масив У і значення, що визначають інтервал Х, використовувалися в двох процедурах (розрахунок даних і побудова графіка), то вони описувалися в розділі

```
public   y:array[1..20] of real;   x,xn,dx,xk,n:integer;
```

Нижче приводиться повний лістинг програми [6].

```
public   y:array[1..20] of real;   x,xn,dx,xk,n:integer;
   { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.N1Click(Sender: TObject);
                        {Розрахунок даних}
   var i:integer;   xx:string;
begin
     xn:=strtoint(edit1.Text);  xk:=strtoint(edit2.Text);
     dx:=strtoint(edit3.Text);
     x:=xn;    i:=1;
       while x<=xk do
        begin
```

```
            y[i]:= 3.2*sqr(x/2 )-9*(x/8)  +20*sin(x)-50;
            str( y[i] :1:1,xx);
           memo1.lines.add('x ='+ inttostr(x) +  '  y  ='+xx);
           i:=i+1;
           x:=x+dx;
       end;
    n:=i—1;
 end;
procedure TForm1.N2Click(Sender: TObject);
                    {Побудова графіка}
var xc,yc,i,m,px,py,mx:integer;
begin
    with image1.Canvas  do
begin
        xc:=image1.width div 2;
        yc:=image1.height div 2;
        pen.Width:=5;pen.Color:=rgb(0,0,0);
        moveto(0,image1.height div 2);
        lineto(image1.width,  image1.height div 2 );
        moveto(image1.width div 2,0);
        lineto(image1.width div 2,image1.height );
        pen.Width:=1;pen.Color:=rgb(0,0,0);
        pen.Style:=psDot;
        m:=xc div 5;
        for i:=0 to 8 do begin
            moveto(xc+i*m,  0);lineto(xc+m*i  , height );
            moveto(xc—i*m,  0);lineto(xc—m*i  ,height );
            moveto(0,yc—m*i);
            lineto(2*xc,  yc—m*i );
            moveto(0,yc+m*i);
            lineto(2*xc,  yc+m*i );
        end;
      x:=xn;  i:=1;
      pen.color:=rgb(0,0,255);  pen.width:=8;
      mx:=trunc(2*xc/(xk—xn));
      moveto( (x)*mx+xc,yc— trunc(y[i]));
      while x<=xk do
        begin
            px:= x*mx +xc;
            py:=yc—trunc(y[i]);
            image1.Canvas .lineto (px,py);
            x:=x+dx;  i:=i+1;
        end;
     font.Size:=10;
    textout(xc+5,10, 'Y');
```

```
        textout(2*xc—25,yc—20, 'X');
        textout( xc—15,yc—20, '0');
        textout( xc+50,20, 'Графік функції');
        for i:=1 to 3 do
                textout(xc—24,yc—m*i, inttostr(50*i));
        for i:=1 to 3 do
                textout(xc—24,yc+m*i, inttostr(—50*i));
        for i:=1 to 4 do
                textout(xc+m*i,yc+14, inttostr(3*i));
        for i:=1 to 4 do
                textout(xc—m*i,yc+14, inttostr(3*i));
    end;
end;
procedure TForm1.N3Click(Sender: TObject);
                {Припинення роботи програми}
    begin
    close;
    end;
end.
```

4.4 Компонент Image і його властивості

Відкрийте панель **ADDITIONAL** і натисніть по кнопці  , на екран буде поміщений компонент **Image**. Властивість **Picture** визначає файл для завантаження. На рис. 4.10 показано вікно вибору графічного файлу. Спочатку натискують по кнопці із зображенням багатокрапки в рядку **Picture**, а потім у вікні редактора, що відкрилося, за допомогою кнопки **Load** вибирають файл.

Розмірами малюнка управляють дві властивості **Stretch** і **AutoSize**. Якщо властивість *AutoSize*=true, то розмір **Image** автоматично підбирається під розмір файлу; інакше розмір компоненту може бути недостатній для представлення всього малюнка або навпаки, виявиться дуже великим. Властивість **Stretch** масштабує малюнок під розмір компоненту.

Властивість *Center*=true розташовує малюнок по центру компоненту.

Рисунок 4.10 – Загрузка файлу в компонент **Image**

Властивість **Transparent** (прозорість) дозволяє зробити малюнок прозорим, що можна використовувати при накладенні одного файлу на інший [1]. На рис.4.9 показаний різний вид накладених файлів залежно від значення цієї властивості.

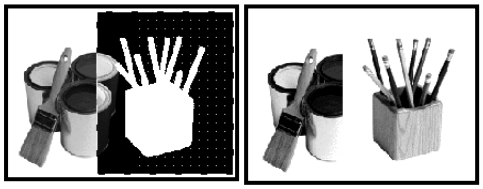


Рисунок 4.9 – Використання властивості Transparent
а - *Transparent =true* ; б - *Transparent =false*

Використовуючи компонент **Image** можна зробити переглядача малюнків. Для цього відкрийте сторінку Win3.1 і додайте на форму компонент **FilelistBox**. Встановити його властивість **Mask** =*.jpg. Це дозволить переглядати файли з відповідним розширенням. Для елементу **Image** задайте властивість **Stretch** =True. Введіть оператора

Image1.picture.loadfromfile(Filelistbox1.filename);

у процедуру обробки події **OnClick** компоненту **FilelistBox**. Цей оператор дозволить завантажувати в компонент **Image1** файл, вибраний в діалоговому вікні. Натисніть двічі по формі та задайте теку для пошуку графічних файлів.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
k:=0;      Filelistbox1.directory:='H:\Картинки';
end;
```

У нашому випадку передбачається відкривати диск H: і в ньому теку «Картинки».

На рис. 4.11 приведений результат роботи процедури.

Рисунок 4.11 – Проглядання малюнків

4.4 Компонент Chart, побудова графіків

Для швидкої побудови діаграм використовується компонент **Chart**, розташований на панелі **ADDITIONAL**. Розглянемо використання цього компоненту на прикладі.

Припустимо, що потрібно побудувати два графіки функцій: параболи $y=3 \cdot x^2 - 6 \cdot x - 100$ і гіперболи $y=1.3 \cdot x^3 - 6 \cdot x^2 + 2 \cdot x - 5$.

На форму переносяться два компоненти **RadioButton**, які організовуватимуть вибір виду графіка, та компонент **Chart**, на якому будуватиметься графік.

На рис. 4.12 показана форма з нанесеними на неї об'єктами.

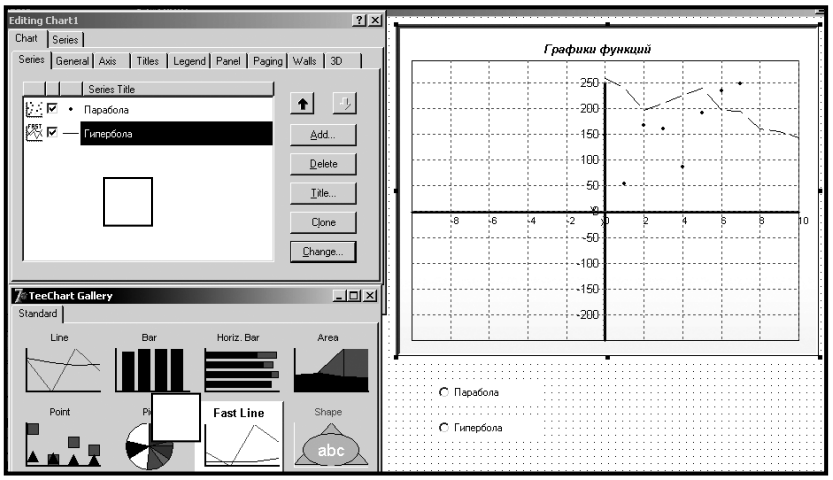


Рисунок 4.12 – Форма з нанесеними на неї об'єктами

На формі можуть бути побудовані одночасно два графіки або один з вибраних. Для вибору типу діаграми слід натиснути правою кнопкою миші усередині компоненту **Chart**. Вибрати підміню «Edit Chart». Для додавання серій (графіка або діаграми) вибирається пункт «*Add*» (рис. 4.11, вікно 1), кнопка «*Delete*» використовується для видалення непотрібних серіїв, «*Title*» - для введення заголовка, «*Clone*» - для створення копії графіка і кнопка «*Change*» - для зміни зовнішнього вигляду діаграми (рис. 4.11, праворуч).

Програмно можна задати колір, розмір шрифту, стиль оформлення і текст заголовка, а також ввести рівняння для побудови графіка, як це зроблено в процедурі, що приводиться.

```
        procedure TForm1.RadioButton1Click(Sender:
TObject);
        var x:integer; color:tcolor;
        begin
        if     RadioButton1.Checked=true then
        {Якщо обрана парабола}
        begin
        color:=rgb(128,128,128);
        {Розмір шрифту заголовка}
        Chart1.Title. font.size:=12;
        {Колір шрифту заголовка}
        Chart1.Title. font.color:=rgb(255,255,255);
        Chart1.Title. brush.color:=rgb(0,255,0);
         {Оформлення заголовка}
        Chart1.Title. brush. style:=bscross;
        Chart1.Title.Text.add('Парабола');  {Текст заголовку}
         for x := -10 to 10 do
                        {Цикл для розрахунку значень У}
          Series1.AddXY( x, 3*x*x-6*x-100,  clBlack);
        end;     end;
```

Розглянемо докладніше використання функції ADDXY (x, y, колір). В ролі першого параметру задається змінна, яка відкладатиметься по осі X, в ролі другого – залежність, що описує У, потім указується колір графіка.

Процедура, що описує побудову гіперболи пишеться аналогічно, ми приводимо тільки фрагмент розрахунку значень У.

```
for x := -10 to 10 do
Series2.AddXY( x, 1.3*sqr(x)*x -6*x*x+2*x-5,   clblue);
```

На рис. 4.13 показаний результат побудови обох графіків. Як видно, парабола побудована по крапках, для гіперболи вибраний лінійний графік.

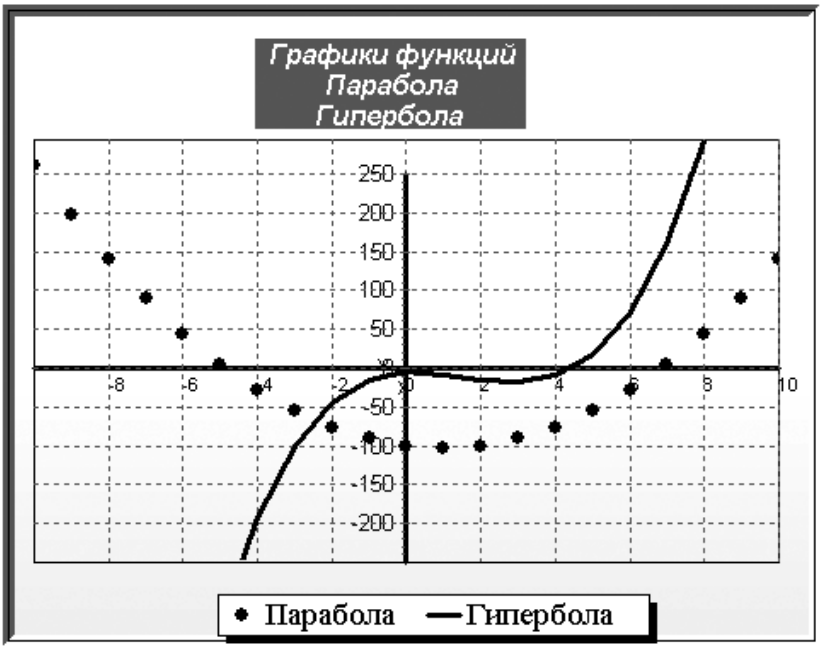


Рисунок 4.13 – Графіки функцій

На рис. 4.14 показано вікно редагування діаграми (Editing Chart). Нами була розглянута перша сторінка «*Series*». Сторінка «*General*» використовується для завдання зовнішнього вигляду прямокутній області, в якій будується графік: наявність смуг прокрутки, розмірів і полів по краях графіка.

Сторінка «*Axis*» задає розташування осей, написи на них, крок ділень (рис. 4.14, а), що відкладаються, мінімальне і максимальне значення.

Сторінка «*Titles*» використовується для задання заголовка і варіантів його оформлення. Сторінка «*Legend*» дозволяє задавати легенду графіка, «*Panel*» - змінювати зовнішнє оформлення прямокутника, в якому будується графік, подальші три сторінки також визначають зовнішнє оформлення графіка.

Окрім вкладки «*Chart*», яка задає стиль діаграми в цілому, при редагуванні можна змінювати зовнішній вигляд кожної серії окремо (вкладка «*Series*»), рис. 4.14 б.

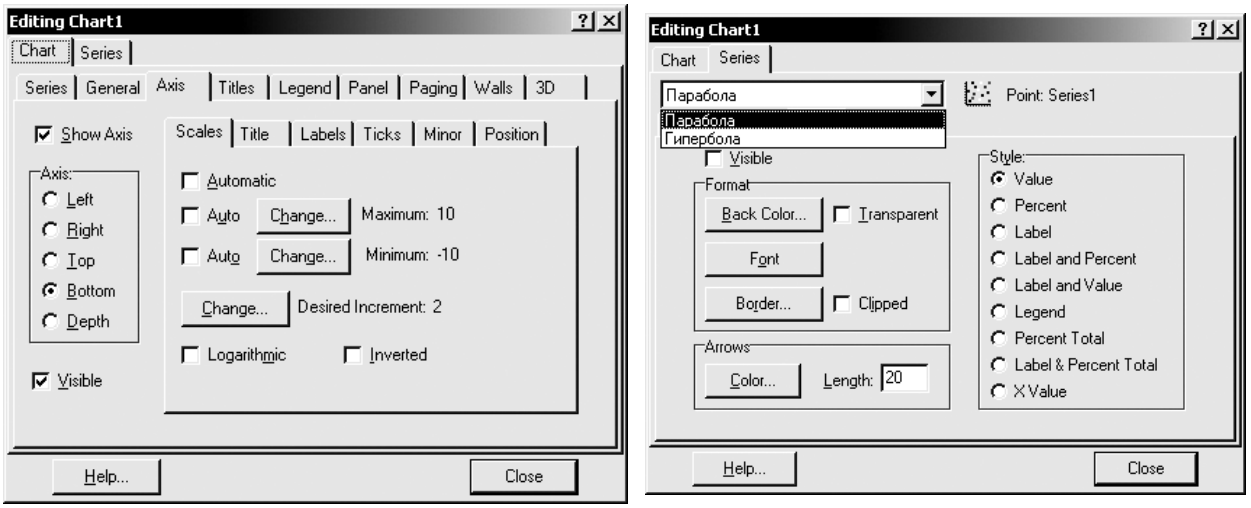


Рисунок 4.14 – Модификація діаграм
а) зміна осей; б) зміна параметрів окремих серій

Окрім графіків функцій, які задаються рівняннями, можна будувати графіки і діаграми, використовуючи дані таблиць, як це показано на рис. 4.15.

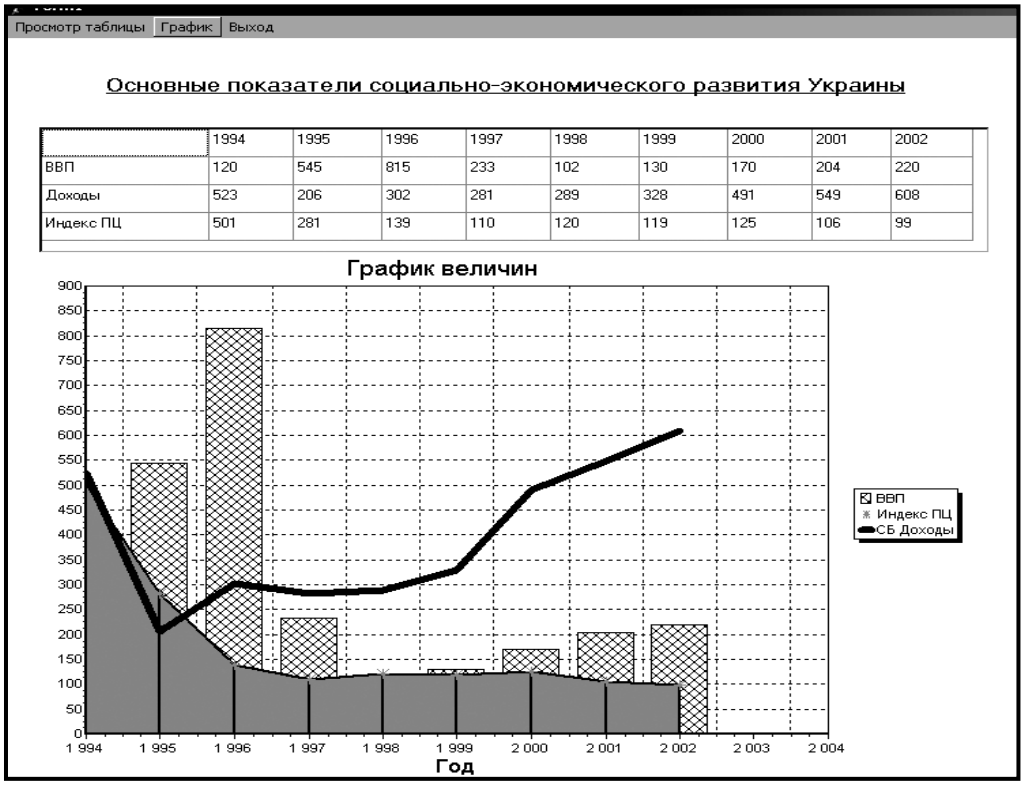


Рисунок 4.15– Побудова серій на підставі інформації таблиць

Після введення даних в таблицю, задається побудова серій за допомогою процедури.

```
procedure TForm1.N3Click(Sender: TObject);
var i,n:integer;
begin
n:=9;    {Кількість даних в кожній серії}
i:=1;
for i:= 1 to n do
begin
                {Задання значень по вісям X і Y}
form1.Series1.AddXY(strtofloat(StringGrid1.Cells[i,0]),
strtofloat(StringGrid1.Cells[i,1]));
form1.Series2.AddXY(strtofloat(StringGrid1.Cells[i,0]),
strtofloat(StringGrid1.Cells[i,2]));
form1.Series3.AddXY(strtofloat(StringGrid1.Cells[i,0]),
strtofloat(StringGrid1.Cells[i,3]));
end;
end;
```

Як видно з тексту програми, що приводиться, дані для осі X прочитуються з нульового рядка таблиці. Це роки від 1994 по 2002. Перша серія будується на підставі даних першого рядка таблиці (ВВП), друга - відповідно з другого рядка (Доходи) і аналогічно третя серія (Індекси ПЦ).

Основні положення теми

✓ Для створення графічних зображень в середовищі Delphi можна використовувати властивість **Canvas.**

✓ Методи **Canvas** дозволяють варіювати при зображенні кольором, товщиною і стилем олівця (пера); вибирати заливку об'єктів різним кольором і видом штрихування.

✓ При створенні зображень необхідно враховувати особливості системи координат, центр якої розташовується в лівому верхньому кутку екрану, таким чином вісь X направлена, як завжди, зліва направо, а вісь У- зверху вниз.

✓ При завданні координат об'єктів використовуються тільки цілі достатні (>0) числа.

✓ Оператор **Refresh** застосовується для очищення екрану.

✓ Для побудови графіків в середовищі Delphi використовують властивість **Canvas** та компонент **Image.**

✓ Для скорочення запису при зверненні до властивостей **Canvas** відбувається за допомогою оператора **With.**

✓ При побудові графіків необхідно враховувати особливості системи координат і при необхідності вводити масштабування.

✓ Підрисуночний надпис, який пояснює рисунок, зручно вводити за допомогою методу **Textout.**

✓ Компонент **Image** використовується для вбудовування графічних файлів в програму.

✓ Основною властивістю є властивість **Picture**, що визначає який файл повинен бути завантажений.

✓ Масштабуванням файлів управляють дві властивості **Stretch** і **AutoSize.**

✓ Властивість **Transparent** дозволяє робити рисунки прозорими і накладати їх один на одного.

- ✓ Компонент **Chart** використовується для швидкого і зручного відображення інформації у вигляді графіків і діаграм.
- ✓ Основним методом додавання даних в діаграму є метод **ADDXY**.
- ✓ Підменю **EditChart** дозволяє модифікувати зовнішній вигляд діаграм, видаляти їх, міняти вид графічного представлення інформації.

Тести для самоперевірки знань до теми 4

Завдання 1–5 мають по чотири варіанти відповіді, серед яких лише ОДИН ПРАВИЛЬНИЙ. Виберіть правильний варіант відповіді.

1. Укажіть призначення **Pen** у властивості **Canvas**:
а) для зафарбування областей;
б) для викреслювання ліній;
в) для завдання кольору ліній;
г) для завдання стиля ліній;

2. Укажіть призначення **Brush** у властивості **Canvas**:
а) для зафарбування областей;
б) для викреслювання ліній;
в) для завдання кольору ліній;
г) для завдання стиля ліній;

3. Укажіть призначення **MoveTo(X,Y)** у властивості **Canvas**:
а) переміщення олівця з позиції X у позицію Y;
б) установка олівця в позицію з координатами X, Y;
в) креслення лінії від X до Y;
г) креслення лінії від Y до X.

4. Укажіть правильний оператор для виводу тексту на екран:
а) Canvas.TextOut(50;50;'Текст');
б) Canvas.TextOut('Текст',50,50);
в) Canvas.TextOut(50,50,'Текст');
г) Canvas.TextOut(50, 'Текст',50).

5. Для введення даних у діаграму використовується метод:
а) EditChart; в) AddXY;
б) Picture; г) Stretch.

Завдання 6–7 **мають два варіанти відповіді. Треба обрати ПРАВИЛЬНИЙ, щоб приведений вислів мав сенс.**

6. Оператор **Refresh** використовується для заповнення екрану заданим кольором.

Так | Ні

7. Для побудови графіків використовують властивість **Canvas** та компонент **Image**.

Так | Ні

Завдання 8–11 мають на меті встановлення відповідності. До кожного рядка, позначеного ЦИФРОЮ, доберіть відповідник, позначений БУКВОЮ.

8. Установите відповідність назви і позначення основних властивостей компонента **Pen.Style:**

1. *psSolid*
2. *psDash*
3. *psDot*
4. *psDashDot*
5. *psDashDotDot*

А. Лінія з серії розривів
Б. Лінія з розривів і крапок
В. Лінія не зображується
Г. Суцільна лінія
Д. Лінія з крапок
Е. Лінія вигляду

	А	Б	В	Г	Д	Е
1						
2						
3						
4						
5						

9. Установите відповідність назви основних властивостей компонента **Brush.Style**, які задають стиль штрихування і їх зовнішній вид**:**

1. *bsSolid*
2. *bsCross*
3. *bsHorizontal*
4. *bsVertical*
5. *bsFDiagonal*
6. *bsBDiagonal*
7. *bsDiagCross*

А.
Б.
В.
Г.
Д.
Е.
Ж.
З.

	А	Б	В	Г	Д	Е	Ж	З
1								
2								
3								
4								
5								
6								
7								

10. Установите відповідність між назвою і позначенням основних властивостей компонента **Pen**:

1. *Color*
2. *Style*
3. *Width*

А. Задає товщину олівця
Б. Визначає колір лінії
В. Визначає стиль лінії
Г. Завдає довжину олівця

	А	Б	В
1			
2			
3			

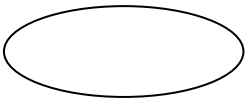
11. Установите відповідність назви і позначення основних методів компонента **Canvas** для зображення фігур**:**

1. *Ellipse*
2. *Arc*
3. *Rectangle*
4. *Poligon*
5. *RoundRect*

А.

Б.

В.



Г.

Д.

Е.

	А	**Б**	**В**	**Г**	**Д**	**Е**
1						
2						
3						
4						
5						

Питання для самоконтролю

1. Які об'єкти Delphi мають властивість Canvas ?
2. Охарактеризуйте систему координат методу Canvas.
3. Який метод застосовується для викреслювання ліній і контурів на полотні?
4. Який метод застосовується для зафарбування областей, обмежених контурами?
5. За допомогою якого оператора визначається стиль, яким малюються лінії?
6. Яким чином можна задати товщину олівця?
7. Який метод дозволяє малювати дугу, що є частиною еліпса, обмеженого крапками?
8. За допомогою якого метода визначається побудова еліпса, який вписаний в прямокутник з координатами верхньої лівої і нижньої правої точок?
9. Який метод використовують для зображення прямокутників і квадратів?
10. В чому полягає відмінність методу Polyline від методу Polygon?
11. За допомогою якого метода малюється замкнутий сектор кола або еліпса?
12. Який метод використовується для створення замкнутих фігур, обмежених еліпсом з координатами і лінією?
13. Який метод дозволяє вивести текст на поверхню графічного об'єкту?
14. Назвіть властивості, що характеризують виведений на поверхню графічного об'єкту текст?
15. Охарактеризуйте основні властивості компоненти Image?
16. Який компонент панелі **ADDITIONAL** використовується для швидкої побудови діаграм?
17. Як відбувається вибір типу діаграми в компоненті Chart?
18. Яким чином можна додавати або видаляти серії (графіка або діаграми), вводити заголовки, створювати копії графіка та змінювати зовнішній вигляд діаграми?
19. Назвіть основний метод додавання даних в діаграму.

5 ФАЙЛОВА СИСТЕМА DELPHI

5.1. Процедури для роботи з файлами

Файлом є об'єкт, що складається з послідовності компонент. Довжина файлу визначається як кількість цих компонент. Створення текстового файлу відбувається, наприклад, в Блокноті; при збереженні даних в Excel створюється файл Excel. Файл може бути джерелом інформації, коли з нього відбувається зчитування, і приймачем, якщо проводять запис в нього інформації .

Розрізняють файли трьох типів: текстові (визначаються типом TextFile), такі, що типізуються (задаються пропозицією File of ...) і не типізуються (визначаються типом File).

Файловий тип можна задати одним з трьох способів, наведених у табл.5.1.

Таблиця 5.1 – Різновиди файлових типів і способи задання

Загальний спосіб задання	Приклад	Файловий тип
<ім'я> : File of <тип>;	f1:file of real	файл, в який записані дійсні числа
<ім'я> : TextFile;	f1:textfile;	текстовий файл
<ім'я> : File;	f1: file	опис файлу, що не типізується

Вид файлу, взагалі кажучи, визначає спосіб зберігання в нім інформації. Проте в Delphi немає засобів контролю виду раніше створених файлів. При оголошенні вже існуючих файлів програміст винен сам стежити за відповідністю виду оголошення характеру що зберігаються у файлі даних.

Операції з файловими змінними можна розділити на чотири групи (табл.5.2):

-установчі та завершуючі операції;

-операції введення-виведення;

-переміщення по файлу;

-спеціальні операції.

Для зберігання інформації, отриманої в результаті роботи програм, використовують різні зовнішні пристрої, що запам'ятовуються: гнучкі магнітні диски, оптичні диски, вінчестера і інші носії. Файлова система складається з двох рівнів: логічних і фізичних файлів. У загальному випадку зберігання інформації організоване у вигляді іменованих областей зовнішньої пам'яті, званих файлами. Файл в такому розумінні називають *фізичним* файлом. Для організації роботи з фізичними файлами в мовах програмування передбачені спеціальні структури даних тип даних файл. Файл в такому розумінні називають *логічним* файлом, оскільки в цьому випадку він є лише логічною моделлю зберігання інформації не залежну від організації конкретного фізичного файлу.

Логічний файл описується як змінна одного з файлових типів, фізичний файл – це реально існуючий файл, що знаходиться в одній з тек Windows. Всі основні процедури і функції, що забезпечують введення-виведення даних, працюють тільки з логічними файлами. Фізичний файл повинен бути пов'язаний з логічним до виконання процедур відкриття файлів.

Для простоти і наочності файлову змінну можна зобразити як послідовність однотипних компонент що зберігається на магнітній стрічці. На самому початку записується ім'я файлу, після нього компоненти файлу, а в самому кінці ознака кінця файлу. Прочитування або запис інформації з файлу здійснюється за допомогою уявної магнітної головки, яку ми надалі називатимемо поточним покажчиком.

Файл може знаходитися в різних станах:

1. Файл закритий.
2. Файл відкритий для читання.
3. Файл відкритий для запису.

Файли стають доступні програмі тільки після виконання особливої процедури відкриття файлу. Ця процедура полягає в пов'язанні раніше оголошеною файловою змінною з ім'ям що існує або новостворюваного файлу, а також у вказівці напряму обміну інформацією: читання з файлу або запис в нього.

Таблиця 5.2 - Розподіл операцій з файловими змінними

Група	Процедура	Призначення
1 група	AssignFile(f, 'FileName')	пов'язує файлову змінну **f** з фізичним файлом, повне ім'я якого задане в рядку FileName
	Reset(f)	відкриває логічний файл **f** для подальшого читання даних або, як то кажуть, відкриває вхідний файл. Після успішного виконання процедури Reset файл готовий до читання з нього першого елементу.
	Rewrite(f)	відкриває логічний файл **f** для подальшого запису даних (відкриває вихідний файл). Після успішного виконання цієї процедури файл готовий до запису в нього першого елементу.
	CloseFile(f)	закриває відкритий до цього логічний файл. Виклик процедури **CloseFile** необхідний при завершенні роботи з файлом.
2 група	Write(F,Data);	запис у файл. Тип змінної для запису повинен бути сумісний з базовим типом файлової змінної.
	Read(F,Data);	зчитування з файлу Тип змінної для запису повинен бути сумісний з базовим типом файлової змінної.
3 група	EOF(f)	повертає значення TRUE, коли при читанні досягнутий кінець файлу. Це означає, що вже прочитаний останній елемент у файлі або файл після відкриття опинився порожній.
	Seek(f, N)	дозволяє явно змінити значення поточного показника, встановивши його на елемент із заданим номером.
	N:=FileSize(f)	Визначається розмір файлу
	M:=FilePos(f)	Визначається поточна позиція у файлі
4 група	Rename(f, NewName)	дозволяє перейменувати фізичний файл на диску, пов'язаний з логічним файлом **f**. Перейменування можливе після закриття файлу.

	Erase(f)	*знищує фізичний файл на диску, який був пов'язаний з файловою змінною* ***f****. Файл до моменту виклику процедури Erase повинен бути закритий.*
	ChDir('ім'я')	*Вибір поточного підкаталогу*
	MkDir('ім'я')	*Створення підкаталогу*
	RmDir('ім'я')	*Видалення порожнього підкаталогу*

Файлова змінна для скріплення логічного і фізичного файлів зв'язується з ім'ям файлу в результаті звернення до стандартної процедури **Assignfile**, яка має один з двох форматів:

Assignfile (f1,'A:text.txt'); — зв'язує файлову змінну *f1* з файлом text.txt, який знаходиться на диску A:

або

Assignfile (f1,Opendialog1.filename); — зв'язує файлову змінну *f1* з файлом, який буде вказаний при появі діалогового вікна Opendialog1.

Для запису даних у файл використовується процедура **Rewrite(f1)**. При використанні цієї процедури відбувається створення нового файлу.

Для закриття файлу використовується процедура **Closefile**(f1).

Процедура **Reset**(f1) дозволяє відкрити існуючий файл для зчитування з нього інформації. Після виконання операцій читання/запису файли повинні бути закриті. Закриття файлу дозволяє зберегти інформацію, що зберігається в них, від ушкоджень при збої в роботі програми. Окрім цього тільки закриті файли можна видаляти або перейменовувати.

У процедурі, що приводиться нижче, відбувається запис двох рядків в текстовий файл і двадцяти цілих випадкових чисел у файл, що типізується, містить цілі числа.

Структура текстових файлів відрізняється тим, що вміст текстового файлу розглядається як послідовність символьних рядків змінної довжини.

Логічну структуру текстового файлу можна зобразити у вигляді наступної схеми (рис.5.1):

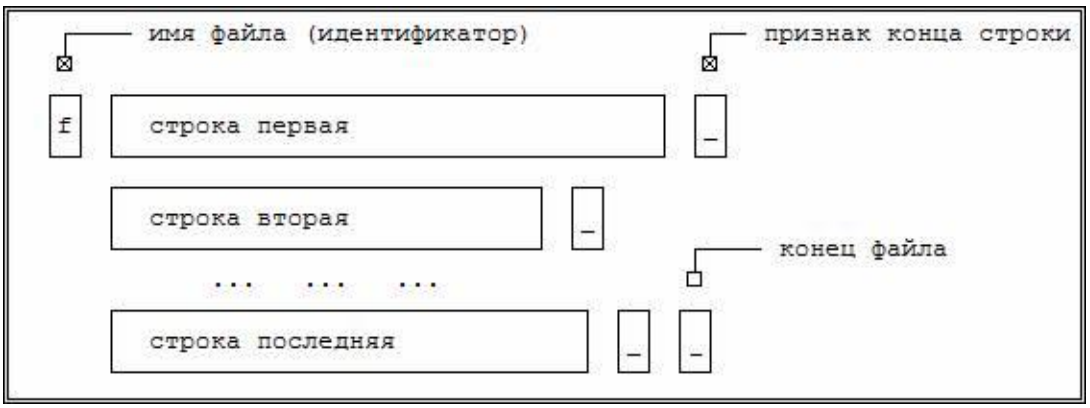


Рисунок 5.1 – Логічна структура текстового файлу

Представником текстового файлу в програмі є змінна файлового типу, яка повинна бути описана з вказівкою стандартного типу textfile:

Var fp1,fp2:textfile;

Текстовий файл трактується як сукупність рядків змінної довжини. Доступ до кожного рядка можливий лише послідовно, починаючи з першої. Аналогічно, при читанні будь-яка частина рядку сприймається як символьне преставлення значення, тип якого відповідає змінній з процедури Read. Readln WriteLn - після свого виконання здійснюють переход до наступного рядка.

Робота з текстовим файлом відбувається рядок за рядком, причому характер читання і запису є строго послідовним.

Початкові операції **AssignFile, Reset, Rewrite, CloseFile** до них додана процедура **Append(fp1)** - відкриває файл на запис, але попередній вміст не віддаляється. Нові дані дописуються в кінець файлу.

Операції введення-виводу **Read Write** працюють аналогічно.

Інструкція **Write** предназначена для виведення на екран монітору повідомлень і значенні змінних. Після слова write в дужках задається перелік змінних, значення яких повинні бути виведені. Крім найменувань змінних в перелік можна додати повідомлення — текст, розташований в одинарні кавички.

Наприклад:

```
write(Summa);
write('Результат розрахунків');
```

Після імені змінної через двокрапку можна помістити опис (формат) поля виведення значення змінної.

Для змінній типу Integer формат — це ціле число, яке задає ширину поля виводу (кількість позицій на екрані).

Наприклад, інструкція

```
write(d:5);
```

показує, що для виведення значення змінної d використовується 5 позицій.

Якщо значення змінної таке, що його зображення займає менше позицій, чим вказано у форматі, то перед першою цифрою числа будуть виведені пропуски так, щоб загальна кількість виведених символів була рівна вказаному у форматі.

Для змінних типу Real формат уявляє собою два цілих числа, разділених двокрапкою. Перше число визначає ширину поля виведення, друге — кількість цифр дробової частини числа. Якщо задати тільки ширину поля, то на екрані з'явиться число, представлене в форматі з плаваючею точкою.

Наприклад, нехай змінна x1 типу real має значення 13.25 , тоді в результаті виконання інструкції

```
write('x1=',x1:5:2);
```

на екрані буде виведено: *x1=13.25*

Інструкція **Writeln** відрізняється від інструкції **Write** тільки тим, що після виведення повідомлення або значень змінних курсор переводиться на початок наступного рядка.

Текстовий файл є набором символів, тому при записі в нього даних іншого типу здійснюватиметься перетворення в символьний. Аналогічно, при зчитуванні чергова частина рядка сприймається як символьне кінцеве значення, тип якого відповідає змінній з процедури **Read**. **Readln WriteLn** - після свого виконання здійснюють перехід до наступного рядка.

EoLn(f) - перевіряє, чи досягнутий кінець поточного рядка.

```
procedure TForm1.Button1Click(Sender: TObject);
var xx:string;
  m,i:integer;
f2:textfile;
f3:file of integer;          {Опис файлових змінних}
begin
assignfile(f2,'c:q.txt');   {На диску C: буде створений текстовий файл q.txt }
 rewrite(f2);     {Файл відкривається для запису в нього інформації}
xx:=' У файл записується інформація';
writeln(F2,xx);
```

```
writeln(f2', Це текстовий файл' );
closefile (f2);                    {Закриття файлу}
        assignfile(f3,'c:q1.dat');
        {На диску С: буде створений текстовий файл q1.dat }
        rewrite(f3);
        {Файл відкривається для запису в нього інформації}
For i:=1 to 20 do
begin
        m:=random(50);
            { Отримання п'ятдесяти випадкових чисел}
 write(f3,m); { Запис їх у файл}
end;
closefile (f3);                        { Закриття файлу}
          reset(f3);
        {Файл відкривається для прочитування інформації}
For i:=1 to 20 do
begin
read(f3,m);
{Прочитування інформації з файлу і вивід в компонент Memo2}
memo2.Lines.Add(inttostr(m)) ;
end;
  closefile (f3);
end;
```

На рис. 5.2 показані відкриті файли, для їх відкриття використаний Блокнот. Інформація у файлах, що типізуються, зберігається в машинному уявленні, компоненти файлу нічим не відділяються один від одного, кожний компонент займає об'єм, рівний об'єму свого типу.

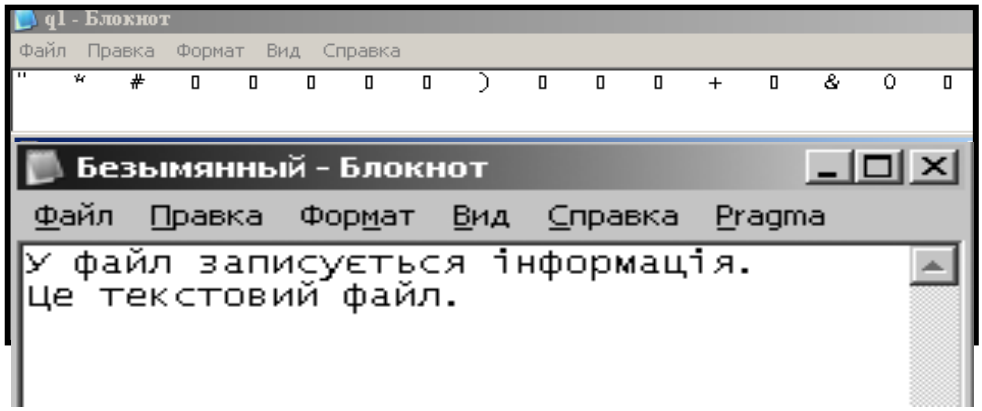


Рисунок 5.2 - Файли q1.dat (верхнє вікно) і q.txt (нижнє вікно), відкриті в Блокноті

При записуванні/прочитуванні інформації з файлу текстового типу допустимі оператори:

writeln **–** після виконання цього оператора здійснюється перехід на новий рядок;

write **–** запис інформації в один рядок;

readln **–** прочитування відрядкового тексту;

read **–** прочитування в один рядок.

Для файлів, що типізуються, допустимі тільки дві операції ***write*** і ***read***, оскільки при записуванні в них даних не вводиться поняття рядка.

Процедура **Erase** знищує файл на диску. Наприклад, **Erase**(f2); знищує створений у попередній процедурі файл q.txt, при цьому опис файлу повинний бути розташований у розділі ***Public*****.**

Логічна функція **EOF**(f) використовується для аналізу стану файлу. Вона повертає значення *true*, якщо досягнутий кінець файлу. Це дозволяє здійснювати прочитування інформації з файлу, не знаючи заздалегідь скільки даних в нього записано.

Нижче нами наводиться приклад прочитування чисел з файлу q1.dat, створеного в попередній процедурі.

```
procedure TForm1.Button2Click(Sender: TObject);
 var s,k,a:integer;
begin    s:=0;
reset(f3);  {Файл відкривається для зчитування інформації}
while not Eof(f3) do
begin
   {Считування інформації з файлу, підрахунок кількості чисел у
                                        файлі і їх суми}
read(f3,a);
k:=k+1;s:=s+a;end;
                                    closefile (f3);
      {Виведення отриманих результатів в компонент Memo}
memo1.lines.add  ('s='+inttostr(s));
memo1.lines.add  ('k='+inttostr(k));
end;
```

Функція **EOln**(f) повертає значення *true*, якщо досягнутий кінець рядка або файлу і використовується при роботі з текстовими файлами **[**2**].**

Розглянемо процедуру, що дозволяє визначити кількість символів і рядків в текстовому файлі. У процедурі введено три лічильники:

S – визначає кількість рядків;

S1 – зберігає кількість знаків з пропусками;

S2 – без пропусків.

У програмі організовано два цикли. Перший працює доки не буде досягнутий кінець файлу, другий - цикл організовується у межах рядку.

procedure TForm1.Button3Click(Sender: TObject);

```
        var s,k,s1,s2:integer;   col:char;
begin
 s1:=0; s2:=0; s:=0; {Занулення лічильників}
       memo2.lines. clear;  {Очищення компоненту  Memo}
       assignfile(f2,'c:q.txt');

  {Зв'язує файлову змінну f2 з файлом q.txt, який знаходиться на
                                    диску C:}

{Файл відкривається для зчитування інформації}
reset(f2);
       while not eof(f2) do {Доки не досягнутий  кінець файла}
           begin
           k:=0;
           while not  eoln(f2) do
  {Доки не досягнутий  конець рядка}
                begin
                read (f2,col); {Посимвольне зчитування інформації}
                if col<>' ' then s2:=s2+1; {Підрахунок всіх символів в
рядку, за виключенням  пропусків}
                k:=k+1; {Підрахунок всіх символів в рядку}
                end;
           readln(f2 );
           memo2.lines.add
            (inttostr(s+1)+'рядок містить'+ inttostr(k)+ '  симв. ' );
                  {Виведення кількості символів в рядку}
            s:=s+1;      s1:=s1+k;
            end;

memo2.lines.add  ('Кількість рядків'+inttostr(s));
memo2.lines.add  ('Знаків з пропусками '+inttostr(s1));
memo2.lines.add  ('Знаків без пропусків'+inttostr(s2));
closefile (f2);  {Закриття файла}
```

```
end;
```

На рис. 5.3 показаний текстовий файл, який аналізується, і результати роботи програми. Для контролю правильності роботи програми файл був проаналізований в Microsoft Word.

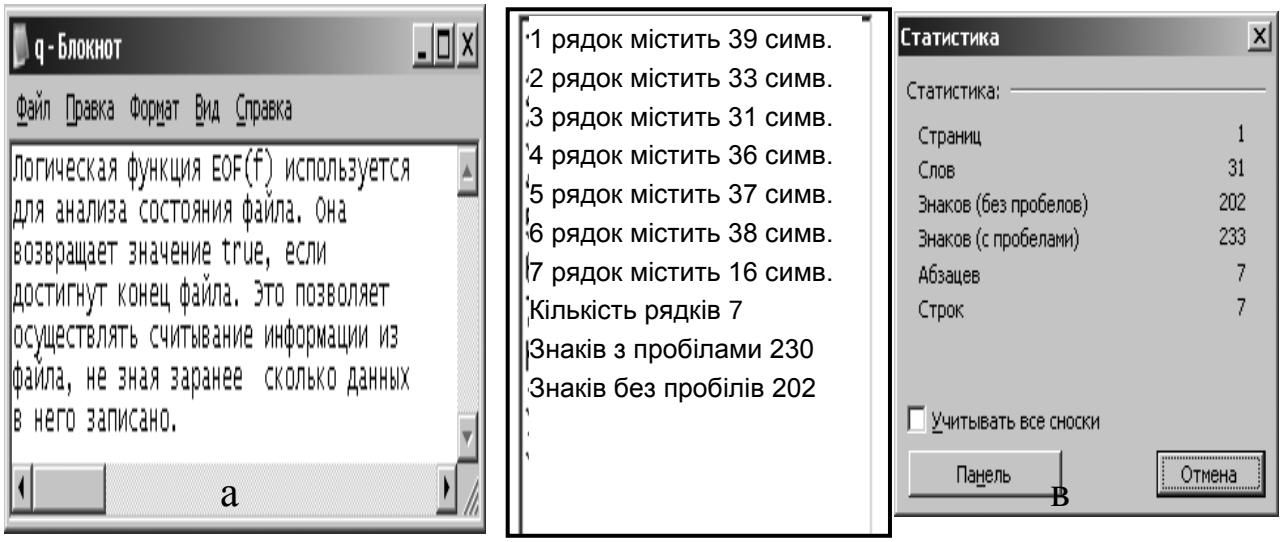


Рисунок 5.3 – Визначення кількості рідків і символів у файлі
а- анализуємий текстовий файл; б- результати роботи програми;
в – вікно «Статистика» Microsoft Word

5.2. Приклад застосування файлової системи

Файли зручно використовувати при роботі з інформацією великого об'єму. При запуску програми немає необхідності вводити кожного разу інформацію, а можна считувати її з файлу і при необхідності змінювати.

Розглянемо задачу, у якій використовується масив з достатньо великою кількістю даних. На рис.5.4 показана таблиця з введеними початковими даними. У розділі 3 ми розглядали один з варіантів введення даних в таблицю, коли дані вводилися при її появі екрані. Другим можливим варіантом є введення початкових даних програмно. Цей варіант використовують у тому випадку, коли дані відносно постійні.

При завантаженні програми з'являється таблиця з вже введеними даними, які при необхідності можна змінити.

Наприклад, при створенні таблиці, що містить розклад руху поїздів, дані міняються не так і часто. Окремі виправлення вносяться вже при роботі з програмою.

Дата отримання інф	Параметр	Прибуток	Фонди основні, г	Фонди обігові, грн.
Січень 2004	1	-2428	319	1633
Березень 2004	2	-2319	3135	3134
Червень 2004	3	-953	3102	6660
Жовтень 2004	4	-663	3072	6938
Січень 2005	5	5899	961	13188
Березень 2005	6	11673	13305	9456
Червень 2005	7	10623	13983	42732
Жовтень 2005	8	17747	20097	54896
Січень 2006	9	12873	23066	50181
Березень 2006	10	11060	26104	59935

Рисунок 5.4 – Табличні початкові дані

На рис. 5.5 показано створене нами меню для роботи з програмою. Меню «**Таблиця**» використовується для завантаження шапки таблиці, «**Введення даних»** дозволяє вибрати, дані з якого стовпчика повинні вводитися програмно, і вивести ці дані на екран. «**Перехід на іншу форму»** відкриває нову форму, на якій будується графік, для записування/зчитування інформації введено наступні два пункти меню: пункт «**Про програму»** виводить на екран опис функцій, що виконуються програмою; робота припиняється при виборі пункту «Вихід».

Текст наведеної нижче процедури формує шапку таблиці і визначає її розміри.

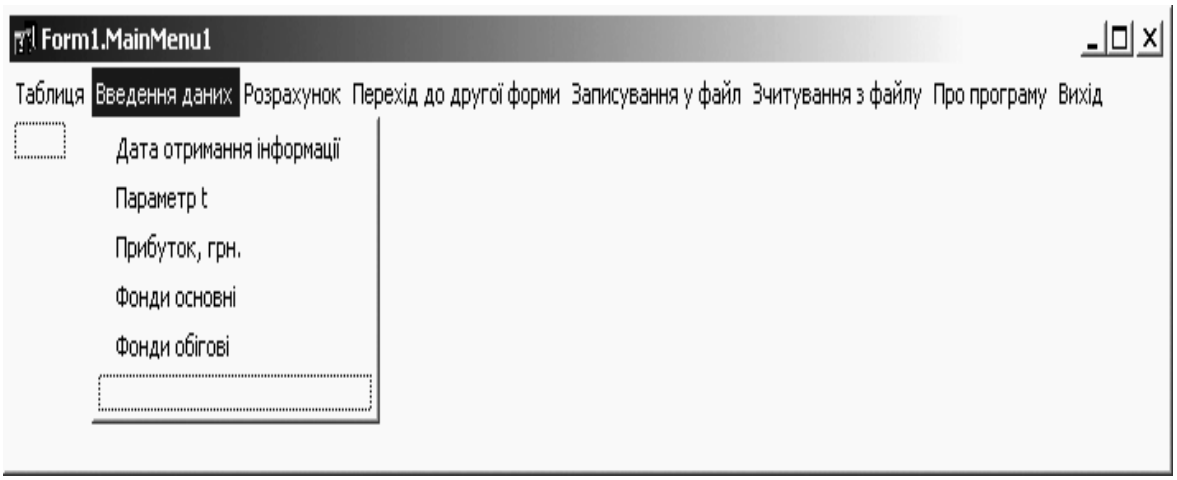


Рисунок 5.5 – Програмне меню

```
procedure TForm1.N1Click(Sender: TObject);
begin
      Memo1.Visible:=false;
{Компонент Memo, який використовується для введення тексту
                опису функцій програми, невидимий}
     StringGrid1.show; {Таблиця виводиться на екран}
     with stringgrid1 do
{Оператор приєднання }
          begin
 {Визначається ширина  кожного стовпця окремо}
             colwidths[0]:=150; colwidths[1]:=75;
              {Сумуються  розміри стовпців і визначається
                        загальна  ширина  таблиці}
colwidths[2]:=75; colwidths[3]:=125; colwidths[4]:=145;
  width:= colwidths[0] + colwidths[1] +   colwidths[2] +
             colwidths[3]   +  colwidths[4]+10 ;
                        {Вводиться «шапка» таблиці}
               cells [0,0]:= 'Дата отримання інформації';
        cells [1,0]:= 'Параметр t';
        cells [2,0]:= 'Прибуток, грн.';
        cells [3,0]:= 'Фонди основні, грн.';
        cells [4,0]:= 'Фонди обігові, грн.';
           end;
end;
```

Для введення даних в перший стовпчик таблиці «Дата отримання інформації» використана процедура:

```
procedure TForm1.N3Click(Sender: TObject);
begin
       with stringgrid1 do
   begin
    cells [0,1]:= 'Січень 2004'; cells [0,2]:= 'Березень 2004';
    cells [0,3]:= 'Червень 2004';cells [0,4]:= 'Листопад 2004';
    cells [0,5]:= 'Січень 2005';cells [0,6]:= 'Березень 2005';
    cells [0,7]:= 'Червень 2005';cells [0,8]:= 'Листопад 2005';
    cells [0,9]:= 'Січень 2006';cells [0,10]:= 'Березень 2006';
   end;
end;
```

В кожну комірку заноситься інформація, яка з’явиться на екрані при виборі цього пункту меню. Ми не наводимо тексти програм, які використовуються для заповнення інших стовпців

таблиці, бо вони відрізняються тільки номером стовпців і інформацією, що заноситься в комірки.

За умовою задачі вимагається розрахувати три статистичні характеристики: математичне очікування, дисперсію і середнє квадратичне відхилення. Математичне очікування розраховують за формулою $M_x = \frac{\sum_{i=1}^{N} x_i}{N}$, де M_x – математичне очікування , x_i – можливі значення величини, N – кількість даних. Дисперсію визначають як $D_x = \frac{\sum_{i=1}^{N} (x_i - M_x)^2}{N}$, а середнє квадратичне відхилення як корінь із дисперсії $\sigma_x = \sqrt{D_x}$.

Щоб з даними таблиці можна було б проводити різні дії, їх необхідно зчитати в масиви. При цьому масив даних типа Дата залишається текстовим типом, бо він потрібний тільки для побудови графіка, а інші масиви перетворюються в дані числових типів.

Для визначення математичних очікувань табличних даних реалізована процедура:

```
procedure TForm1.Mx1Click(Sender: TObject);
                              {Опис локальних змінних}
var i:integer; xx:string;
                 begin
               with stringgrid1 do
                  begin
                    for i:=1 to 10 do
                      begin
                         N[i]:=cells[0,i]; {Дані}
     Fob[i]:=strtoint(cells[2,i]);        {кожного   стовпця}
                         Pr[i]:=strtoint(cells[2,i]);
       FOSN[i]:=strtoint(cells[2,i]); {заносяться до масиву}
                      end;
                         M1:=0;
cells[0,11]:='Mx'; {Встановлюється початкове значення Mx рівним
нулю і вводиться в нульовий стовпець назва обчислювальної
величини}
     for i:=1 to 10 do M1:=M1+pr[i]; {Накоплюється сума масиву
значень прибутку}
     M1:=1/10*M1;
  str(M1:1:2,xx);cells[2,11]:= xx;
     {Розрахунок середнього  арифметичного і виведення  його
                                  значення на екран}
               {Аналогичні обчислення для двох інших масивів}
               M2:=0;
               for i:=1 to 10 do M2:=M2+Fosn[i];
               M2:=1/10*M2;
               str(M2:1:2,xx);
               cells[3,11]:= xx;
               M3:=0;
               for i:=1 to 10 do M3:=M3+Fosn[i];
               M3:=1/10*M3;
               str(M3:1:2,xx);
               cells[4,11]:= xx;
                 stringgrid1.RowCount:=stringgrid1.RowCount+1;
    {Збільшення кількості рядків таблиці на один для виведення
                         значень математичних очікувань}
                end;
end;
```

Процедура, яка використовується для розрахунку значень дисперсій, організована аналогічно.

```
procedure TForm1.Dx1Click(Sender: TObject);
var i: integer;   xx:string;
begin
     stringgrid1.RowCount:=stringgrid1.RowCount+1;
     d1:=0;
     stringgrid1.cells[0,12]:='Dx';
     for i:=1 to 10 do d1:=d1+sqr(pr[i]—M1);
     d1:=d1/10;
     str(d1:1:2,xx);
     stringgrid1.cells[2,12]:= xx;
     d2:=0;
     for i:=1 to 10 do d2:=d2+sqr(fosn[i]—M2);
     d2:=d2/10;
     str(d2:1:2,xx);
     stringgrid1.cells[3,12]:= xx;
     d3:=0;
      for i:=1 to 10 do d3:=d3+sqr(fob[i]—M2);
     d3:=d3/10;
     str(d3:1:2,xx);
     stringgrid1.cells[4,12]:= xx;
end;
```

Розрахунок значень середнього квадратичного відхилення реализовано за приведеним алгоритмом.

```
procedure TForm1.x1Click(Sender: TObject);
var xx: string;
    begin
    stringgrid1.RowCount:=stringgrid1.RowCount+1;
    stringgrid1.cells[0,13]:='Ср.кв.откл';
     q1:= sqrt(d1);
     str(q1:1:2,xx);
     stringgrid1.cells[2,13]:= xx;
     q2:= sqrt(d2);
     str(q2:1:2,xx);
     stringgrid1.cells[3,13]:= xx;
     q3:= sqrt(d3);
     str(q3:1:2,xx);
     stringgrid1.cells[4,13]:= xx;
 end;
```

Позначення помилки починається з латинської букви E (табл.5.2) :

Таблиця 5.2

Опис помилки	Позначення
EzeroDevide	ділення на нуль
EconvertError	помилка перетворення типу
EInOutError	помилка введення-виведння
Eoverflow	помилка переповнювання для цілочисельних операцій
EMathError	помилка в математичних операціях
EaccessViolation	помилка виникає при неправильному використанні пам'яті

При створенні додатків часто приходиться виконувати стандартні операції у виді операцій відкриття файлів, записуванні в них інформації, вибору шрифтів, кольорової палітри, друку. Розробники Delphi передбачили автоматизоване виконання цих операцій, включивши їх в бібліотеку сторінки DIALOGS (рис. 5.6).

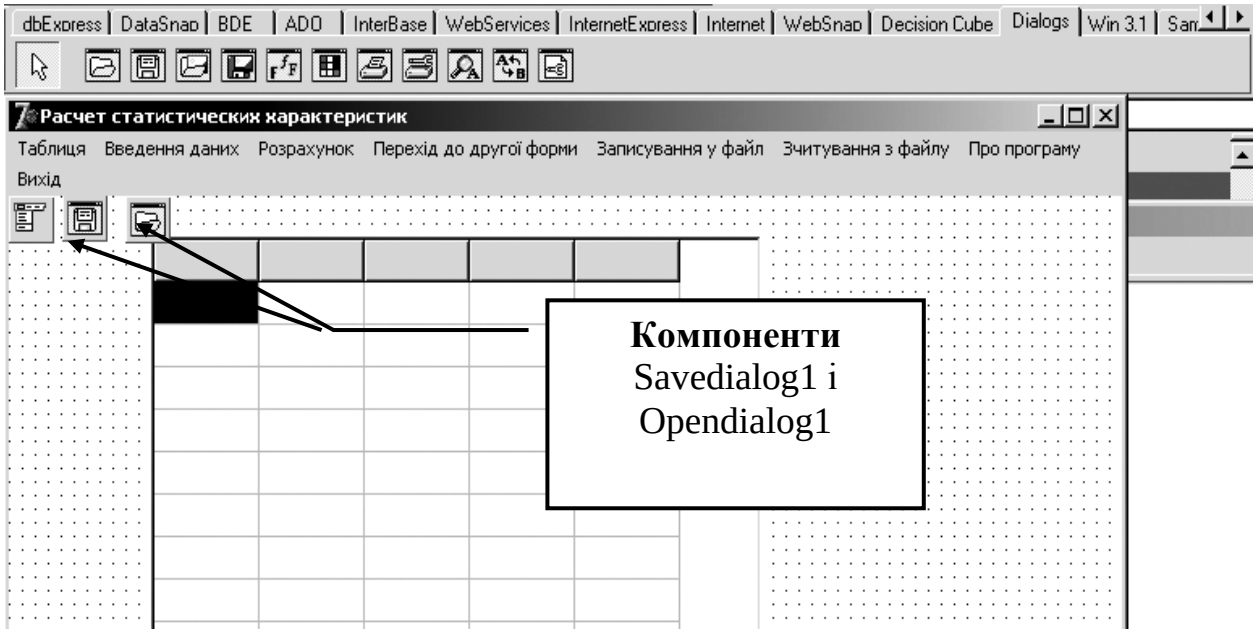


Рисунок 5.6 - Розташування компонентів **Savedialog1** і **Opendialog1** на формі

Властивості компонентів **Savedialog** і **Opendialog** практично однакові:

✓ *FileName* – строкова змінна, яка зберігає ім’я обраного користувачем файла;

✓ *Filter* – обмежує типи файлів, які з'являються в діалоговому вікні роботи з ними. Наприклад, шаблони *.txt| *.doc дозволять продивлятися тільки файли двох указаних типів. Зверніть увагу, замість звичної коми розмежувачем шаблонів виступає риска «|».

✓ Властивість *InitialDir* визначає диск і теку, які будуть відкриті при використанні діалогових вікон, в нашому прикладі - це диск H:, а в ньому - тека ABC.

✓ Властивість *Title* задає заголовок вікна. Якщо властивість залишити незаповненою, з'явиться стандартне повідомлення Windows. Як видно з рисунків, нами заданий текст «Ви зберігаєте таблицю», який і з'являється замість стандартного.

✓ Основним методом опису компоненту є **EXEcute**. Якщо при відкритті діалогового вікна користувач обрав файл або ввів його ім'я, то функція **EXEcute** повертає *true*. Якщо вибір не зроблений, натиснута кнопка «Відміна», якщо клавіша «**Esc**», то повертається значення *false*.

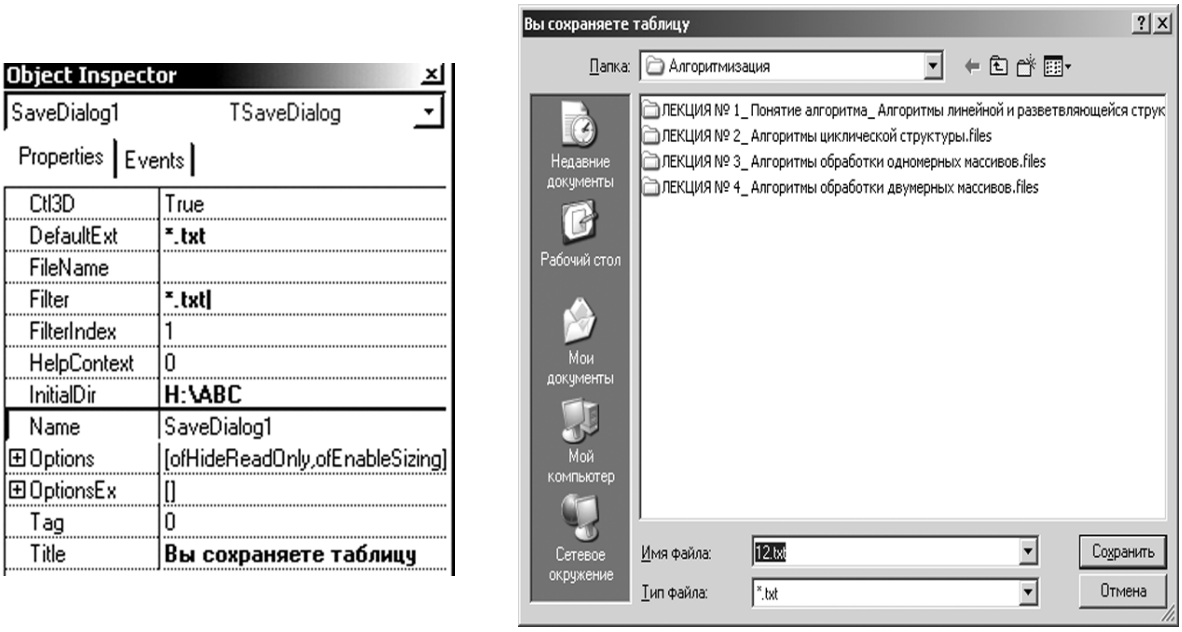


Рисунок 5.7 - Властивості компонента **Savedialog1**

Стандартне звернення до діалогового компоненту має вид:

```
if savedialog1.eXEcute then
 begin
       assignfile(f1,savedialog1.filename);
       rewrite(f1);
{Далі слідують оператори запису до файлу}
 end;
```

Якщо введено ім'я файлу, в якому зберігатиметься інформація, то файлова змінна зв'язується з ім'ям цього файлу і файл відкривається для запису в нього інформації. Запис введених даних у файл відбувається у декілька етапів. Спочатку поміщають на форму компоненти Savedialog1 і Opendialog1 панелі компонентів DIALOGS. Всі Delphi-діалоги, що знаходяться на цій вкладці, у тому числі і Delphi-діалоги вибору файлу, невізуальні, тобто при перенесенні їх на Форму в працюючій програмі їх не видно. Вони видимі тільки на етапі конструювання. Компонент **OpenDialog** дозволяє відкрити в програмі стандартне Windows-окно відкриття файлу, компонент **SaveDialog** - вікно збереження.

Delphi-діалоги вибору файлу самі по собі нічого не роблять, а тільки надають настройки, зроблені користувачем при виборі файлу. Найважливіший метод Delphi-діалогів - **Execute**. Він спацьовує у момент натиснення кнопки "**відкрити"** або "**зберегти"** у вікні вибору файлу.

```
procedure TForm1.N10Click(Sender: TObject);
  var xx:string; i:integer;
  begin
 showmessage
   ('Введіть назву файлу, в який записуватиметься інформація');
 if savedialog1.eXEcute then          {Якщо вказано ім'я файлу}
         begin
            assignfile(f1,savedialog1.filename);
             {Зв'язування файлової змінної  f1  з файлом )
               rewrite(f1);
           { Файл відкритий для запису в нього інформації}
             xx:='Дата отримання інформації';  writeln(f1,xx);
                    {Записується у файл заголовок}
          for i:=1 to 10 do begin
               xx:= stringgrid1.cells[0,i];
               writeln(f1,xx);
         { У файл записується нульовий стовбець  таблиці}
           end;
          xx:='Параметр t';  writeln(f1,xx);
          { У файл записується перший стовбець  таблиці}
          for i:=1 to 10 do begin
             xx:= stringgrid1.cells[1,i]; writeln(f1,xx); end;
             xx:='Прибуток, грн.';writeln(f1,xx);
           { У файл записується другий стовбець  таблиці}
          for i:=1 to 10 do begin
           xx:= stringgrid1.cells[2,i]; writeln(f1,xx);end;
```

```
        xx:='Фонди основні, грн.'; writeln(F1,xx);
       { У файл записується третій  стовбець  таблиці}
        for i:=1 to 10 do begin
        xx:= stringgrid1.cells[3,i]; writeln(f1,xx);end;
        xx:='Фонди оборотні, грн.'; writeln(f1,xx);
     { У файл записується четвертий стовбець  таблиці}
        for i:=1 to 10 do begin
        xx:= stringgrid1.cells[4,i]; writeln(f1,xx); end;
       end;
    closefile(f1); {Закриття файлу}
   end;
```

Якщо за допомогою Блокнота відкрити створений файл, то він виглядатиме аналогічно зображеному на рис. 5.8.

Процедура прочитування інформації з файлу реалізована аналогічним чином.

```
    procedure TForm1.N11Click(Sender: TObject);
    var i:integer; xx:string;
    begin
    showmessage ('Введіть назву файлу, з якого прочитується інформація');
    if opendialog1.execute then {Якщо вказано ім'я файлу}
    begin assignfile(f1,opendialog1.FileName); {пов'язуємо файлову змінну f1  з файлом )
        OpenDialog1.Filter := 'Текстовий файл(*.txt)|*.TXT';
          { Організовуємо фільтр для перегляду тільки файлів з розширенням txt}
        reset(f1); {Файл відкритий для прочитування інформації}
        readln(f1,xx); { Дані з файлу прочитуються в таблицю в тому ж порядку, в якому були в неї записані}
        with stringgrid1 do begin
            cells [0,0]:=xx;
        for i:=1 to 10 do begin
        readln(f1,xx);
        cells [0,i]:=xx;
          end;
        readln(f1,xx);cells [1,0]:=xx;
        for i:=1 to 10 do begin
        readln(f1,xx);
        cells [1,i]:=xx;
```

```
Дата отримання інформації
Січень 2004
Березень 2004
Червень 2004
Жовтень 2004
Січень 2005
Березень 2005
Червень 2005
Жовтень 2005
Січень 2006
Березень 2006
Параметр t
1
2
3
4
5
6
7
8
9
10
Прибуток, грн.
-2428
-2319
-953
-663
5899
11673
10623
17747
12873
11060
Фонди основні, грн.
319
3135
3102
3072
961
13305
13983
20097
23066
26104
Фонди обігові, грн.
1633
3134
```

Рисунок 5.8 - Збережена інформація у файлі

```
end;
readln(f1,xx);cells [2,0]:=xx;
for i:=1 to 10 do begin
readln(f1,xx);
cells [2,i]:=xx;
end; readln(F1,xx);cells [3,0]:=xx;
for i:=1 to 10 do begin  readln(f1,xx);
cells [3,i]:=xx;
end;
readln(f1,xx);cells [4,0]:=xx;
for i:=1 to 10 do begin  readln(f1,xx);
cells [4,i]:=xx;
end;
end;  end;     end;
```

На рис. 5.9 показані результати розрахунків, проведених після зчитування даних з файлу. За отриманими результатами необхідно побудувати графік, який буде розміщений на окремій формі. Для створення нової форми у вже існуючому додатку необхідно виконати команду меню File|New|Form, після чого відкриється нова форма. Таким чином проект Delphi міститиме дві форми, одну для розрахунків і записування/зчитування інформації у файли, а другу - для графічної інтерпретації інформації.

Дата получения инф	Параметр	Прибыль,	Фонды основные	Фонды оборотные,
Янв 2004	1	-2428	319	1633
Март 2004	2	-2319	3135	3134
Июль 2004	3	-953	3102	6660
Окт 2004	4	-663	3072	6938
Янв 2005	5	5899	961	13188
Март 2005	6	11673	13305	9456
Июль 2005	7	10623	13983	42732
Окт 2005	8	17747	20097	54896
Янв 2006	9	12873	23066	50181
Март 2006	10	11060	26104	59935
Mx		6351.20	6351.20	6351.20
Dx		49614230.5	49614230.56	49614230.56
Ср.кв.откл		7043.74	7043.74	7043.74

Рисунок 5.9 - Результати роботи програми

Щоб можна було переходити з однієї форми на іншу, використані оператори:

```
procedure TForm1.N9Click(Sender: TObject);
begin
Form2.show;                    {Показати форму 2}
Form1.visible:=false;          {Приховати форму 1}
end;
```

Оскільки побудова графіка детально описувалася раніше, приводиться лістинг програми і результати її роботи. Використовувані в процедурах масиви N, Pr описані в першому модулі, тому звернення до них має вигляд form1.N[i], form1.P[i].

```
public
  xc, yc: integer;{ Описуємо змінні, що визначають координати центру як глобальні}
  end;
var
  Form2: TForm2;
implementation
uses file1; { У модулі file2, так нами названа друга форма, використовується модуль першої форми file1}

procedure TForm2.N2Click(Sender: TObject);
var i:integer;
begin
with image1.Canvas do
begin
xc:=30; yc:=300;       {Координати цетра}
pen.Width:=3;
                              {Малюємо осі}
moveto (30,30);
lineto(30,yc+300);moveto (30,yc);
lineto (600,yc);textout(32,15,'Прибыль');
                    {Розміщення тексту на осях}
textout (605,295,'Дата'); pen.Width:=1;
for i:=1 to 11 do begin
moveto(xc+i*50,30);lineto(xc+i*50,400);
                  {Створення координатної сітки}
moveto (30,30+i*50);lineto  (600,30+i*50);
 end;  for i:=1 to 11 do begin
                     {Текстові написи на сітці}
textout(25+i*50,352,form1.N[i]);
  end;  end; end;

procedure TForm2.N3Click(Sender: TObject);
var i:integer;
begin
with image1.Canvas do
                           {Побудова графіка }
begin  pen.Color:= rgb(0,0,0); pen.Width:=4;
moveto (80,yc-trunc(form1.pr[1]) div 65);
for i:=2 to 11  do
```

```
        lineto (50*i+xc,yc-trunc(form1.pr[i]) div 65);
        end;end;

        procedure TForm2.N5Click(Sender: TObject);
        begin
        close;                          { Закриття другої форми}
        end;

        procedure TForm2.N4Click(Sender: TObject);
        begin
        Form1.show;                  { Повернення в першу форму}
        Form2.visible:=false;
        end; end.
```

На рис. 5.10 показаний побудований графік.

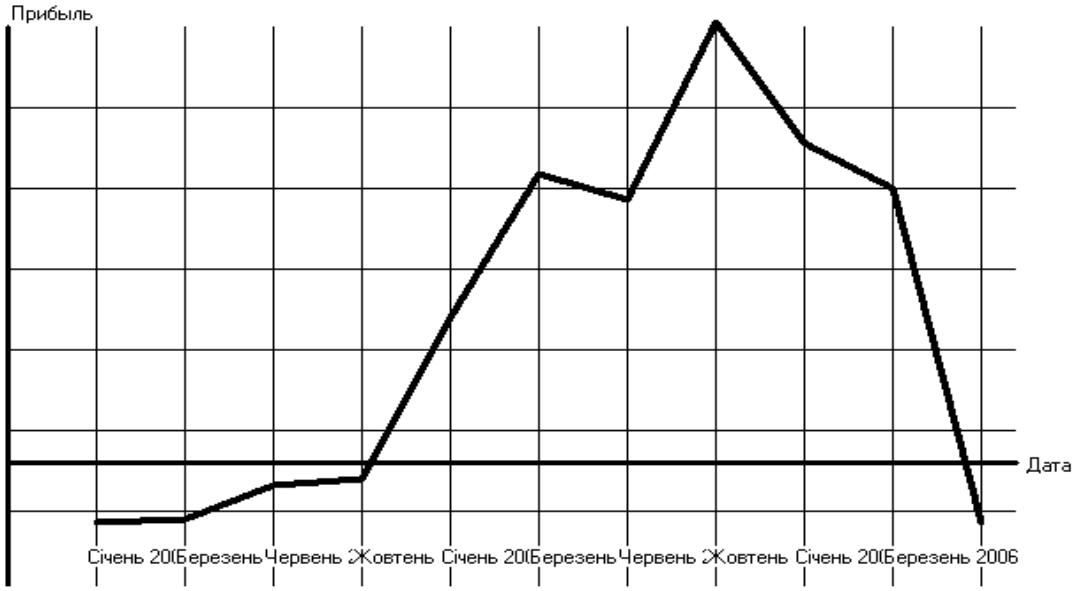


Рисунок 5.10 - Графічна інтерпретація результатів роботи програми

Розглянемо докладніше оператора:

lineto (50*i+xc,yc-trunc(form1.pr[i]) div 65);

Він використаний при побудові графіка. Використання звернення Form1.pr[i] означає, що елементи масиву **PR** можна використовувати і в другій формі, хоча вони розраховувалися або вводилися в першій. Достатньо тільки вказати ім'я форми-джерела даних.

Основні положення теми

✓ Файлом є об'єкт, що складається з послідовності компонент. Довжина файлу визначається як кількість цих компонент.

✓ Файли бувають текстові, такі, що типізуються і не типізуються.

✓ Робота з текстовим файлом відбувається рядок за рядком, причому характер читання і запису є строго послідовним.

✓ Описують файли залежно від їх типу, наприклад:

f1:textfile; — текстовий файл;

f1:file of real;— файл, в який записані дійсні числа;

f1: file ; — опис файлу, що не типізується.

✓ Для скріплення логічного і фізичного файлів використовується процедура **Assignfile**.

✓ Файлові змінні текстових файлів повинні бути описані як глобальні

✓ Для відкриття файлу щодо записвуння інформації використовують оператора **Rewrite.**

✓ Для відкриття файлу на зчитування інформації використовують оператора **Reset.**

✓ При записуванні в текстовий файл будь-яких значень розділювач між ними не ставиться. Але при зчитуванні в ролі розділювача виступають пробіли, символи табуляції та переводуа рядка.

✓ В текстові файли можна записувати тільки прості типи та рядки.

Для закриття файлу застосовують оператора **Closefile(F1).**

✓ Логічна функція **EOF(f)** використовується для аналізу стану файлу. Якщо досягається кінець файлу, її значення стає ***true.***

✓ Функція **EOln** повертає значення ***true***, якщо досягається кінець рядка або файлу і використовується при роботі з текстовими файлами.

✓ При використанні декількох форм допускається звернення до даних, що зберігаються в різних формах, при цьому слід посилатися на імена цих форм.

Тести для самоперевірки знань до теми 5

Завдання 1–3 мають по чотири варіанти відповіді, серед яких лише ОДИН ПРАВИЛЬНИЙ. Виберіть правильний варіант відповіді.

1. Укажіть правильний опис текстового файлу:

а) f1:file; в) f1: textfile;
б) f1:file of real; г) f1: filetext;

2. Укажіть правильний опис файлу, в який записані дійсні числа:

а) f1:file; в) f1: textfile;
б) f1:file of real; г) f1: file of integer;

3. Укажіть правильний опис файлу, що не типізується:

а) f1:file; в) f1: textfile;
б) f1:file of real; г) f1: file of integer;

Завдання 4–7 мають два варіанти відповіді. Треба обрати ПРАВИЛЬНИЙ, щоб приведений вислів мав сенс.

4. Для файлів, що типізуються, допустимі операції ***writeln*** і ***readln***.

Так | Ні

5. Логічна функція **EOF(f)** повертає значення *false*, якщо досягнутий кінець файлу.

Так | Ні

6. Функція **EOln**(f) використовується при роботі з текстовими файлами і повертає значення *true*, якщо досягнутий кінець рядка.

Так | Ні

7. Якщо при відкритті діалогового вікна **Savedialog** обраний файл або введене його ім'я, то функція **Execute** повертає *false*.

Так | Ні

Завдання 8–10 мають на меті встановлення відповідності. До кожного рядка, позначеного ЦИФРОЮ, доберіть відповідник, позначений БУКВОЮ.

8. Установите відповідність між назвою процедури і її призначенням**:**

1. *Assignfile*
2. *Rewrite*
3. *Reset*
4. *Closefile*
5. *Erase*

А. Запис даних у файл
Б. Закриття файлу
В. Знищування файлу на диску
Г. Скріплення логічного і фізичного файлів
Д. Відкриття існуючого файлу для зчитування
Е. Переміщення файлу

	А	Б	В	Г	Д	Е
1						
2						
3						
4						
5						

9. Установите відповідність між назвою оператору та його дією при записуванні/прочитуванні інформації з файлу текстового типу:

1. *Writeln*
2. *Write*
3. *Readln*
4. *Read*

А. Запис інформації на новому рядку
Б. Прочитування в один рядок
В. Прочитування відрядкового тексту
Г. Знищення інформації
Д. Після запису здійснюється перехід на новий рядок

	А	Б	В	Г	Д
1					
2					
3					
4					

10. Установите відповідність між назвою і позначенням основних властивостей компонента **Savedialog**:

1. *FileName*
2. *Filter*
3. *InitialDir*
4. *Title*

А. Обмежує типи файлів
Б. Задає заголовок вікна
В. Відкриває обраний файл
Г. Зберігає ім’я обраного файла
Д. Визначає диск і теку, які будуть відкриті

	А	Б	В	Г
1				
2				
3				
4				

Питання для самоконтролю

1. Який об'єкт є Файлом і яким чином визначається його довжина?

2.Назвіть різновиди файлів в залежності від їх типу.

3Назвіть основні відмінності текстових файлів від типізованих . Назовите основные отличия текстовых файлов от типизированных файлов.

4. Назвіть основні процедури і функції, які призначені для роботи з текстовими файлами.

5. На які чотири групи роздфляють операції з файловими змінними?

6. З яких рівнів складається файлова система?

7. Як описується логічний файл?

8. Що таке фізичний файл?

9. Назвіть умову, яка перед'являється до фізичного файлу до виконання процедур відкриття файлів.

10. Назвіть основні процедури та функції,передбачені для роботи – з текстовими файлами.

11. Яка логічна функція повертає значення *false*, якщо досягнутий кінець файлу?

12. Яка процедура використовується для скріплення логічного і фізичного файлів процедура Assignfile?

13. Як повинні бути описані файлові змінні текстових файлів?

14. Які символи виступають в ролі розділювача при зчитуванні?

15. Який оператор застосовують для закриття файлу ?

16. Яка логічна функція використовується для аналізу стану файлу?

17.Яка логічна функція повертає значення *true*, якщо досягається кінець рядка ?

6 Процедури і функції

6.1. Загальні поняття

При програмуванні завдань іноді виникає необхідність багатократного виконання однієї і тієї ж послідовності дій. Наприклад, необхідно обчислити багато разів величину факторіалу для різних початкових даних. Звичайно, можна кілька разів написати один і той же текст програми, а можна оформити його у вигляді підпрограми і при необхідності викликати стільки раз, скільки обчислень потрібно провести.

Така автономна частина програми, що реалізовує певний алгоритм і що допускає звернення до неї з різних частин програми, називається підпрограмою.

Використання підпрограм дозволяє реалізувати один з найпрогресивніших методів програмування - структурне програмування.

Підпрограми діляться на два види: функції і процедури. Перші використовуються, коли потрібно отримати єдиний результат, а другі – коли результатів повинно бути багато.

Залежно від того, звідки відбуватиметься звернення до підпрограм, розрізняється ступінь їх доступності для різних процедур і модулів.

Наприклад, якщо помістити опис підпрограми (у прикладі це функція) в розділ **public**, вона стає доступною для будь-яких класів, об'єктів і модулів програми [2].

```
public
function  fact(r:byte):longint;
```

Якщо опис процедури поміщений в розділ **implementation**

```
implementation
function  fact(r:byte):longint;
```

то вона стає доступною тільки в межах даного модуля.
Опис процедури, поміщений в одну з процедур програми

```
procedure TForm1.Button1Click(Sender: TObject);
function fact(r:byte):longint;
```

робить її доступною тільки в цій процедурі.

Перш ніж почати описувати підпрограми пригадаємо, що є програма в Delphi. програма зберігається на диску з назвою заголовка програми і розширенням DPR. Ось її архітектура:

```
program Project1;                    //Заголовок програми
uses                                 //Список модулів, що
Forms,                               підключаються
Unit1 in 'Unit1.pas' {Form1};
{$R *.RES}                           //Файли ресурсів, що
                                     підключаються
begin                                //Текст програми
Application.Initialize;
Application.CreateForm(TForm1,
Form1);
Application.Run;
end.
```

Файл програми складається із заголовка і блоку програми. Блок програми включає оголошення модулів (частин програми), що підключаються, ресурсні файли (ікони, покажчики миші, малюнки), що підключаються, текст програми (створення вікон, запуск додатку на виконання).

Викликати файл програми на редагування можна з меню "Project" пунктом "View Source". Щоб уникнути появи помилок, некоректної роботи програми настійно не рекомендується самостійно уручну редагувати цей файл. Всі необхідні зміни проводяться в Delphi автоматично.

Заголовок або назва програми завжди повинні співпадати з назвою проекту. Якщо вам необхідно змінити назву проекту, то слідує пересохранить проект під іншим ім'ям.

Для того, щоб розбити програму на декілька окремих завдань, які можна редагувати окремо і застосовуються модулі. У Delphi кожне окреме проектоване вікно має свій модуль. Але їх в програмі може бути значно більше.

Структура модуля:

```
unit Unit1;                          //Заголовок модуля
interface                            //Заголовки доступних властивостей
uses                                 //Модулі, що підключаються
Windows,SysUtils, Forms;
type                                 //Опис типу
TForm1 = class(TForm)
private                              //Прихований розділ властивостей
```

```
 Private declarations }
        public                          //Відкритий розділ властивостей
 Public declarations }
        end;
        var                             //Опис змінною
Form1: TForm1;
        implementation                  //Опис властивостей
        {$R *.DFM}                      //Файли ресурсів, що підключаються
        end.
```

Вся ваша ручна робота зводиться в написання програмної коди в автоматично створюваних процедурах усередині розділу **implementation**. Весь текст модуля автоматично змінюється при відповідній зміні стану проекту. Наприклад, якщо ви додали або видалили з проектованої форми кнопку або перейменовуєте властивість NAME для будь-якого компоненту.

Як правило, програміст не виходить за області **implementation** розділу, а тут самостійно не створює процедури. Програмна оболонка робить це автоматично в міру необхідності і ліквідовує порожні зайві заголовки процедур при збереженні файлу модуля.

Отже, в Delphi передбачено декілька засобів для ділення програми на частини. На верхньому рівні ділення це модулі, на нижньому рівні (рівні елементарних підзадач) це процедури і функції.

За допомогою процедур і функцій (про це вже мовилося вище) можна скомпонувати групу операторів для виконання деякої одиничної дії. Процедуру або функцію можна викликати з різних місць програми.

Процедури і функції складаються з операторів локальних даних і внутрішніх процедур і функцій. Структура опису процедури має вигляд:

```
Procedure <Ім'я> (Перелік_формальних_параметрів);
Label   //Опис  міток;
Const   //Опис констант;
Type   //Опис типів;
Var  //Опис змінних;
Procedure  //Опис внутрішніх процедур;
Function   //Опис внутрішніх функцій;
Begin
  //Оператори
End;
```

Структура опису функції має вигляд:

```
Function <Ім'я> (Перелік_формальних_параметрів): Тип_результата;
Label  //Опис міток;
Const  //Опис констант;
Type  //Опис типів;
Var  //Опис змінних;
Procedure  //Опис внутрішніх процедур;
Function  //Опис внутрішніх функцій;
Begin
//Оператори, серед яких є хоча б один, який привласнює імені функції значення результату
End;
```

У заголовку *підпрограми* (у її оголошенні) указується список **формальних** *параметрів* змінних, які приймають значення, передавані в *підпрограму* ззовні під час її виклику. Скорочено ми далі опускатимемо слово "формальний".

Оскільки усередині *підпрограми параметри* розглядаються як змінні з початковим значенням, то імена *локальних змінних*, що описуються в розділі var (внутрішньому для *підпрограми*), не можуть співпадати з іменами *параметрів* цієї ж *підпрограми*. Список *параметрів* може і зовсім бути відсутнім:

```
procedure proc1;
function func1: boolean;
```

В цьому випадку *підпрограма* не отримує ніяких змінних "ззовні". Проте відсутність *параметрів* і, як наслідок, передаваних ззовні значень зовсім не означає, що при кожному *виклику підпрограма* виконуватиме абсолютно однакові дії. Оскільки *глобальні змінні* видно зсередини будь-якої *підпрограми*, їх значення можуть неявно змінювати внутрішній стан *підпрограм*. Цьому дуже небажаному ефекту буде присвячений пункт "*Побічний ефект*".

Якщо ж *параметри* є, то кожен з них описується за наступним шаблоном:

```
[<способ_подстановки>]<имя_параметра>:<тип>;
```

Про можливі способи підстановки значень в *параметри* (<порожній>, var, const) ми розповімо в розділі "Способи підстановки *аргументів*".

Якщо спосіб підстановки і тип декількох *параметрів* співпадають, опис цих *параметрів* можна об'єднати:

```
[<способ_подстановки>]<имя1>,...,<имяN>: <тип>;
```

Приклад опису всіх трьох способів підстановки:

```
function func2(a,b:byte; var x,y,z:real; const c:char);
```

У заголовку *підпрограми* можна указувати тільки прості (не складені) типи даних. Отже, спроба записати

```
procedure proc2(a: array[1..100]of char);
```

викличе помилку вже на етапі компіляції. Для того, щоб обійти це обмеження, складений тип даних потрібно описати в розділі type, а при оголошенні *підпрограми* скористатися ім'ям цього типу:

```
type arr = array[1..100] of char;
procedure proc2(a: arr);
function func2(var x: string): arr;
```

Основна відмінність між *функціями* і *процедурами* полягає в кількості *повертаних ними значень*.

Будь-яка *функція*, завершивши свою роботу, повинна повернути основній програмі (або іншій *підпрограмі, що* викликала її) рівне одне значення, причому його тип потрібно явним чином вказати вже при оголошенні *функції*.

Для повернення результату застосовується спеціальна "змінна", що має ім'я, співпадаюче з ім'ям самої *функції*. Оператор привласнення значення цій "змінній" обов'язково повинен зустрічатися в тілі *функції* хоч би один раз.

Наприклад:

```
function min(a,b: integer): integer;
  begin if a>b
  then min:= b
  else min:= a
  end;
```

На відміну від *функцій*, *процедури* взагалі не повертають (явним чином) ніяких значень.

Будь-яка підпрограма може бути викликана не тільки з основного тіла програми, але і з будь-якої іншої підпрограми, оголошеної пізніше за неї. _

При виклику в підпрограму передаються фактичні параметри або аргументи (у круглих дужках після імені підпрограми, розділені комами):

```
<имя_подпрограммы>(<список_аргументов>);
```

Аргументами можуть бути змінні, константи і вирази, що включають виклики функцій.

Кількість і типи передаваних в підпрограму аргументів повинні відповідати кількості і типам її параметрів. Крім того, тип кожного аргументу повинен обов'язково враховувати спосіб підстановки, вказаний для відповідного параметра (докладніше про це буде розказано в розділі "Способи підстановки аргументів"). Якщо у підпрограми взагалі немає оголошених параметрів, то при виклику список передаваних аргументів буде відсутній разом з дужками, що обрамляють його.

Виклик функції не може бути самостійним оператором, тому що повертане значення потрібно кудись записувати. Зате воно може стати рівноправним учасником арифметичного виразу. Наприклад:

```
c:= min(a,a*2);
if min(z, min(x,y))= 0 then...;
```

Процедура ж нічого не повертає явним чином, тому її виклик є окремим оператором в програмі. Наприклад:

```
err(res,'Привет!');
```

Зауваження: Після того, як викликана підпрограма завершить свою роботу, управління передається операторові, наступному за оператором, що викликав цю підпрограму.

Способи підстановки аргументів

Як вже згадувалося вище, при виклику підпрограми підстановка значень аргументів в параметри проводиться відповідно до правил, вказаних в атрибуті <спосіб_подстановки>. Розглянемо три різні значення цього атрибуту:

- <порожній>;
- var;
- const.

Параметр-значення

Опис

У списку параметрів підпрограми перед параметром-значенням службове слово відсутнє. Наприклад, функція func3 має три параметри-значення:

```
function func3(x:real; k:integer; flag:boolean):real;
```

При виклику підпрограми параметру-значенню може відповідати аргумент, що є виразом, змінною або константою, наприклад:

```
dlina:= func3(shirina/2, min(a shl 1,ord('y')), true) +0.5;
```

Для типів даних тут не обов'язковий строгий збіг (еквівалентність), достатньо і сумісності по привласненню.

Механізм передачі значення

В області пам'яті, що виділяється для роботи підпрограми, що викликається, створюється змінна з ім'ям <имя_подпрограммы>.<имя_параметра>, і в цю змінну записується значення переданого у відповідний параметр аргументу. Подальші дії, вироблювані підпрограмою, виконуються саме над цією новою змінною. Значення ж вхідного аргументу не зачіпається. Отже, після закінчення роботи підпрограми, коли весь її тимчасовий контекст буде знищений, значення аргументу залишиться таким самим, яким воно було на момент виклику підпрограми.

Як приклад розглянемо послідовність дій, що виконуються при передачі аргументів 1+a/2, a і true в описану вище функцію func3. Хай a - змінна, що має тип byte, тоді значення виразу 1+a/2 матиме тип real, a true і зовсім є константою (неіменованою).

Отже, при виклику func3(1+a/2,a,true) будуть виконані наступні дії:

1. створити тимчасові змінні func3.x, func3.k, func3.flag;
2. обчислити значення виразу 1+a/2 і записати його змінну func3.x;
3. записати в змінну func3.k значення змінної a;
4. записати в змінну func3.flag значення константи true;
5. провести дії, описані в тілі функції;
6. знищити всі тимчасові змінні, зокрема func3.x, func3.k, func3.flag.

Вже видно, що значення аргументів не зміняться.

Зауваження: При використанні параметрів-значень в контексті підпрограми створюються хоча і тимчасові, але цілком повноцінні копії вхідних аргументів. Тому небажано передавати в параметри-значення "великі" аргументи (наприклад, масиви): вони займатимуть багато зайвої пам'яті.

Параметр-змінна

Опис

У списку параметрів підпрограми перед параметром-змінною ставиться службове слово var. Наприклад, процедура proc3 має три параметра-переменные і один параметр-значення:

```
procedure proc3(var x,y:real; var k:integer; flag:boolean);
```

При виклику підпрограми параметру-змінній може відповідати тільки аргумент-змінна; константи і вирази заборонені. Крім того, тип аргументу і тип параметра-змінної повинні бути еквівалентними.

Механізм передачі значення

На відміну від параметра-значення, для параметра-змінної не створюється копія при виклику підпрограми. Замість цього в роботі підпрограми бере участь та сама змінна, яка послужила аргументом. Зрозуміло, що якщо її значення зміниться в процесі роботи підпрограми, то ця зміна збережеться і після того, як буде знищений контекст підпрограми. Зрозуміло знову ж таки і обмеження на аргументи, які повинні відповідати параметрам-змінним: ні константа, ні вираз не зможуть зберегти зміни, внесені в процесі роботи підпрограми.

Отже, параметри-змінні і служать тими посередниками, які дозволяють отримувати результати роботи процедур, а також збільшувати кількість результатів, повертаних функціями.

Зауваження: Для економії пам'яті в параметр-змінну можна передавати і таку змінну, змінювати значення якої не потрібно. Скажімо, якщо потрібно передати як аргумент масив, то краще не створювати його копію, як це буде зроблено при використанні параметра-значення, а використовувати параметр-змінну.

Параметр-константа

Опис

У списку параметрів підпрограми перед параметром-константою ставиться службове слово const. Наприклад, процедура proc4 має один параметр-змінну і один параметр-константу:

```
procedure proc4(var k:integer; const flag:boolean);
```

При виклику підпрограми параметру-константі може відповідати аргумент, що є виразом, змінною або константою. Під час виконання підпрограми відповідна змінна вважається звичайною константою. Обмеженням є те, що при виклику іншої підпрограми з тіла поточного параметру константі не може бути підставлений як аргумент в параметр-змінну.

Для типів даних тут не обов'язковий строгий збіг (еквівалентність), достатньо й сумісності по привласненню.

Механізм передачі значення

У деяких джерелах можна зустріти твердження про те, що для параметра-константи, як і для параметра-змінної, не створюється копія у момент виклику підпрограми.

```
var a: byte;
Implementation
procedure prob(const c:byte);
  begin
        Label1.Caption:=IntToStr(longint(addr(c)));        {физ.адрес
параметра з}
  end;
Procedure TForm1.Button1Click(Sender:TObject);
begin
  a:=0;
  Label2.Caption:=IntToStr(longint(addr(a)));  {физ.адрес змінної а}
  prob(a);
end;
end.
```

Проте виконання простої перевіркидоводить зворотне: фізичні адреси змінної а і параметра з розрізняються. Отже, в пам'яті ці змінні займають різні позиції, а не одну, як було б у разі параметра-змінної. Ви можете переконатися в цьому самостійно, запустивши дану програму в трьох різних варіантах (для параметра-значення, параметра-змінної і параметра-константи), а потім порівнявши результати.

Області дії імен

Розмежування контекстів

Глобальні об'єкти - це типи даних, константи і змінні, оголошені на початку програми до оголошення будь-яких підпрограм. Ці об'єкти будуть видні у всій програмі, у тому числі і у всіх її підпрограмах. Глобальні об'єкти існують протягом всього часу роботи програми.

Локальні об'єкти оголошуються усередині якої-небудь підпрограми і "видні" тільки цій підпрограмі і тим підпрограмам, які були оголошені як внутрішні для неї. Локальні об'єкти не існують, поки не викликана підпрограма, в якій вони оголошені, а також після завершення її роботи.

Побічний ефект

Оскільки глобальні змінні видно в контекстах всіх блоків, то їх значення може бути змінене зсередини будь-якої підпрограми. Цей ефект називається побічним, а його використання дуже небажано, тому що може стати джерелом незрозумілих помилок в програмі.

Щоб уникнути побічного ефекту, необхідно строго стежити за тим, щоб підпрограми змінювали тільки свої локальні змінні (у тому числі і параметри-змінні).

Збіг імен

Взагалі кажучи, збіги глобальних і локальних імен допустимі, оскільки до кожного локального імені неявно приписано ім'я тієї підпрограми, в якій воно оголошене. Таким чином, в приведеному вище прикладі фігурують змінні a, g, pr1.p, pr1.b, pr1.f.pp, pr1.f.c, pr2.d.

Якщо є глобальна і локальна змінні з однаковим ім'ям, то зсередини підпрограми до глобальної змінної можна звернутися, приписавши до неї спереду ім'я програми:

<имя_программы>.<имя_глобальной змінній>

Наприклад (локальній змінній тут привласнюється значення глобальної):

```
a:= prog.a;
```

Зауваження: Не дивлячись на те що збіги імен локальних і глобальних змінних не викликають ніяких колізій на рівні компілятора, варто все-таки утримуватися від них, тому що вони також можуть стати причиною непередбаченого побічного ефекту, аналогічного описаному в попередньому пункті.

Параметри, що не типізуються

У оголошенні підпрограми можна не указувати тип параметра-змінної:

```
procedure proc5(var x);
```

Такий параметр називатиметься таким, що не типізується. У цей параметр можна передати аргумент, що відноситься до будь-якого типу даних.

Для того, щоб усередині самої підпрограми коректно обробляти значення, що поступили через параметр, що не типізувався, існує два різні способи.

Явне перетворення типу

За допомогою операції явного перетворення типу даних можна перетворити значення, що не типізується, відноситься до потрібного типу даних. Наприклад, в процедурі proc5 значення одного і того ж параметра x інтерпретується трьома різними способами: як ціле число, як дійсне число і як масив:

```
procedure proc5(var x);
type arr = array[1..10] of byte;
var x: integer;
  z: real;
  m: arr;
begin
  ...
  y:= integer(x);
  z:= real(x);
  m:= arr(x);
  ...
end;
```

Поєднання в пам'яті

Другий спосіб: описати усередині підпрограми локальну змінну, яка фізично співпадатиме із змінною, передаваною через параметр, що не типізується:

```
<локальна_змінна>: <тип> absolute <нетипизов_параметр>;
```

В цьому випадку будуть суміщені значення, фізично записані в цих змінних, в точності так само, як це відбувається при підстановці аргументу в параметр-змінну, проте без контролю за збігом типів даних. Тому цілком можлива, наприклад, ситуація, коли перші чотири байти рядка (аргументу, переданого в параметр, що не типізується) сприйматимуться як *longint*-число:

```
function func5(var x):real;
var xxx: longint absolute x;
begin
  { тут з початком будь-якої змінної
  що поступила в параметр x
  ... можна звертатися як з longint-числом:
  за допомогою локальної змінної xxx}
end;
```

6.2. Використання функцій

Розглянемо використання функцій на прикладі розрахунку виразу

$$C = \frac{5!}{3!} - 6! + 4!^2$$

де знак оклику означає обчислення факторіалу, тобто 5! - це добуток чисел 1•2•3•4•5.

Як видно з умови завдання, факторіал потрібно обчислити чотири рази. Отже, доцільно помістити обчислення факторіалу в підпрограму, оскільки результатом є одне число – значення добутку, то можна використовувати функцію, а не процедуру. На рис.6.1 приводиться форма з результатами розрахунків.

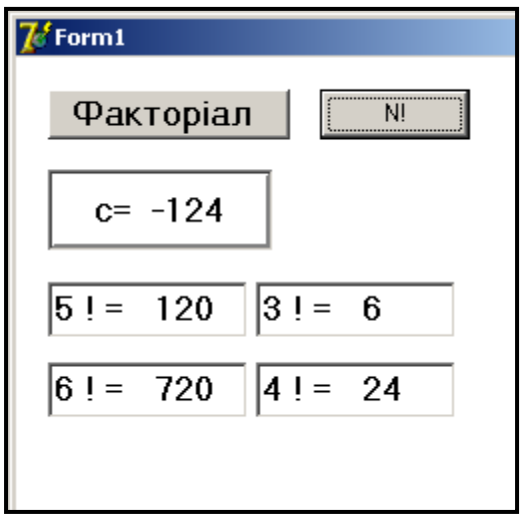


Рисунок 6.1 – Розрахунки з використанням функцій

На формі - дві кнопки, перша служить для розрахунку значення C за формулою, а друга - для виводу на екран складових формули. Оскільки функція викликатиметься з двох процедур, то її опис поміщений в розділ **implementation** (доступність в будь-якій процедурі конкретного модуля).

Опис функції починається із заголовка, що має формат

Function <ім'я функції> (параметри): тип результату;

Ім'я функції вибирається довільно за тими ж принципами, за якими створюються імена звичайних змінних. Після імені функції в круглих дужках слідує перелік формальних параметрів, початкових даних, які необхідно передати для роботи функції, він аналогічний переліку значень, що вводяться, в звичайних процедурах. Але під час описування функції ви нібито описуєте ідею, створюєте макет програми, яка запрацює тільки при заміні формальних параметрів, вказаних у заголовку функції, на фактичні параметри, які будуть визначені при зверненні до неї.

За круглими дужками через двокрапку указується тип результату роботи функції. У нашому прикладі заголовок має вигляд

function fact(r:byte):longint;

де ***fact*** – ім'я функції; ***r*** – параметр, що визначає значення числа, для якого обчислюється факторіал, він описаний як дане типу byte, результат добутку чисел визначений як longint.

Обчислення факторіалу проводиться шляхом організації циклу, що накопичує добуток чисел від двох до r. Для роботи циклу введені дві змінні, які описуються у функції і є її локальними змінними. Добуток позначений як P.

У тілі функції повинен бути обов'язково оператор, що віддає імені функції результат. У нас це *fact:=p;*

Виклик функції проводили з двох процедур. При виклику указують ім'я функції і в круглих дужках приводиться список фактичних параметрів, тобто чисельних даних, необхідних для роботи функції. Оскільки результатом функції є одне значення, наприклад, число, то функцію можна викликати з будь-якого оператора, що працює з даними цього типу. Це може бути оператор привласнення, умовний, циклу і т.д.

Наприклад:

if fact(5)>fact(n) then c:=2 else c:=3;

або

```
for i:=1 to fact(8) do s:=s+fact(i);
```

```
implementation
function fact(r:byte):longint;
     var i:byte; p:longint;
begin      { Функція для розрахунку факторіалу}
p:=1;
for i:=2 to r do p:=p*i;
   fact:=p;
end;

        procedure TForm1.Button1Click(Sender: TObject);
var c:real;
begin
C:=fact(5) /fact(3) -fact(6)+sqr(fact(4) );       {Виклик функції для розрахунку C}
panel1.caption:='c=  '+floattostr(c);
end;

        procedure TForm1.Button2Click(Sender: TObject);
begin
    {Багатократний виклик функції для розрахунку складових
                          виразу для визначення C}
edit1.Text:='5 ! =   '+inttostr(fact(5));
edit2.Text:='3 ! =   '+inttostr(fact(3));
edit3.Text:='6 ! =   '+inttostr(fact(6));
edit4.Text:='4 ! =   '+inttostr(fact(4));
end;
```

При роботі з підпрограмами оперують поняттями «формальні» і «фактичні» параметри. Під «формальними» параметрами розуміють перелік початкових даних, які потрібні для роботи підпрограми, і імена результатів її роботи [4].

При виклику підпрограми формальні параметри замінюються на фактичні, тобто дані, які необхідні для обчислень і імена результатів. Порядок проходження, тип і кількість параметрів повинні співпадати.

Наприклад, якщо описана функція із заголовком виду

function Name(a:byte; b,c : real; d:string):integer;

то при зверненні до неї з основної програми виклик повинен виглядати таким чином

N:=Name(23,3.4,5.6,’one’);

Як видно з прикладу, кількість початкових даних дорівнює чотирьом, перший формальний параметр повинен бути цілого типу, два інших – речового і останній – символьного. Порушувати порядок і відповідність типів параметрів, а також їх кількість не можна. При порушенні описаних типів з'являються повідомлення про помилки:

[Error] Unit1.pas(48): Too many actual parameters – кількість фактичних параметрів перевищує кількість формальних;

[Error] Unit1.pas(48): Not enough actual parameters – кількість фактичних параметрів менше кількість формальних;

[Error] Unit1.pas(48): Incompatible types: 'Byte' and 'Extended'	невідповідність типів параметрів
[Error] Unit1.pas(48): Incompatible types: 'Byte' and 'String'	

6.3. Процедури

Якщо з алгоритму виходить необхідність отримання більш ніж одного результату, наприклад, при роботі з масивами, використання функції стає неприпустимим. В цьому випадку використовують такий вид підпрограм, як процедури. Принцип дії процедур, їх розміщення і алгоритм заміни формальних параметрів на фактичні аналогічний описаному в попередньому параграфі.

Проте, підпрограми-процедури мають відмінності від підпрограм-функцій.

1. Оскільки результатом дії процедури є не одне значення, а декілька, то її не можна викликати з виразу. Для виклику процедури існує спеціальний оператор виклику.

2. У заголовку процедури відсутній опис типу результату, він поміщається в списку формальних параметрів і відрізняється від початкових даних наявністю слова **var.**

3. Формальні параметри діляться на два види: параметри-значення і параметри-змінні. Параметри-значення аналогічні формальним параметрам функції і служать для передачі в підпрограму початкових даних для її роботи, а параметри-змінні використовуються як для передачі початкових даних, так і/або для повернення в точку виклику процедури результатів її роботи.

Опис процедури можна розміщувати або в розділі **implementation** або в локальній процедурі.

В обох видах підпрограм можливе використання змінних, описаних в розділі **implementation**, вони визначаються як глобальні змінні або в розділі опису тієї процедури, з якої походить виклик підпрограми. Такі змінні називаються локальними.

Глобальні змінні діють на всьому тілі програми, включаючи підпрограми і звичайні процедури. Область дії локальних процедур обмежується тілом тієї процедури, з якої походить виклик.

Заголовок процедури виглядає таким чином

```
procedure vas(n,m:byte;c:real; var  d:byte; var e:real);
```

де ***n,m,c*** - параметри-значення, а ***d,e*** - параметри – змінні.

Виклик процедури відбувається по оператору `vas(1,2,3.4,dd,ee),` де перші три значення – початкові дані для її роботи, а подальші два – результати виконання процедури.

Розглянемо алгоритм взаємодії параметрів процедури на прикладі.

У процедурі **TForm1.Button1Click** вводяться початкові значення для чотирьох змінних *a:=0; b:=0;c:=0;d:=0.* У процедурі `vas(n:byte; var  d:byte);` їх значення змініються на одиницю. До і після звернення до процедури значення виводяться на екран в компонент Memo **[**2**].**

Розглянемо текст програми.

```
implementation    var b,d,c,a:byte;

procedure vas(n:byte; var  d:byte);
     begin
a:=1; b:=1;  c:=1;d:=1; {Змінювання значень змінних
                                    в процедурі}
     end;

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var a,c:byte;
begin
    memo1.Lines.add('Значення параметрів до моменту звернення до
                                    процедури');
 a:=0;   b:=0; c:=0; d:=0;              {Початкові значення змінних}
memo1.Lines.add('a=' + inttostr(a) + '  b=' + inttostr(b) +
                                  '  c=' + inttostr(c) + '  d=' + inttostr(d) );
vas(0,d);   {Виклик процедури}
memo1.Lines.add(' Значення параметрів після звернення
                                    до процедури ' );
memo1.Lines.add('a=' + inttostr(a) +  '  b=' + inttostr(b) +
                                '  c='+inttostr(c) + '  d='+inttostr(d) );
end;
```

Розберемо кожен параметр окремо. Значення ***a*** до виклику процедури було рівним нулю, це значення є параметром-значенням. Отже, його зміни в підпрограмі не передаються в програму, яка здійснює виклик. Параметр ***a*** описаний як локальний в програмі, це означає, що область його дії обмежується місцем виклику і не розповсюджується на підпрограму. Локальні описи перекривають глобальні в області їх опису, тому, не дивлячись на те, що в процедурі його значення змінюється на одиницю, при поверненні в основну програму воно залишається рівним нулю [2].
На рис.6.2 показаний результат виконання процедури.

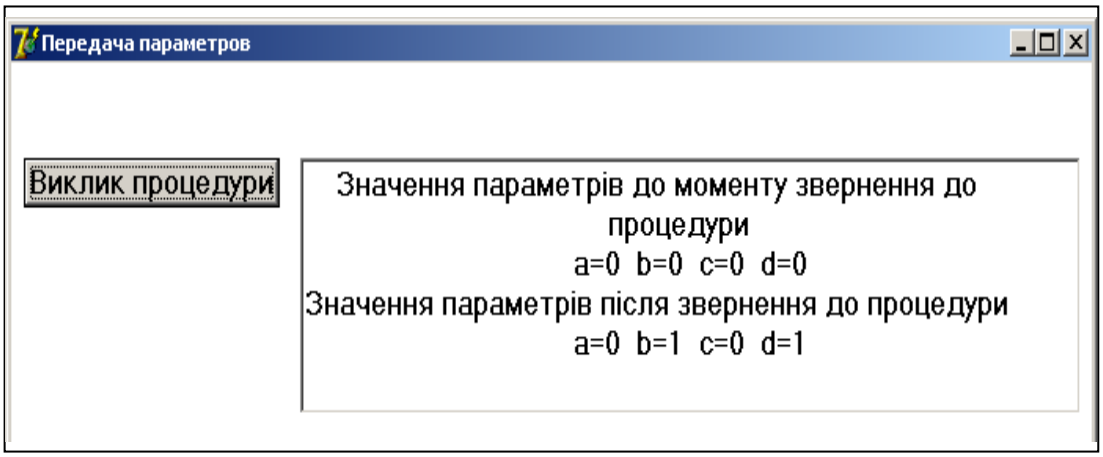


Рисунок 6.2 - Результати виконання процедури

Параметр ***b*** описаний як глобальний, що дозволяє йому діяти однаково у всьому тілі програми і його значення, що змінилося на одиницю, і передається на екран з основної програми.

Параметр ***c*** також описаний як локальний, і механізм зміни його значень аналогічний механізму зміни значень параметра ***a***.

Параметр ***d*** є параметром-змінною. Отже, його зміни передаються в основну програму як результат її виконання і значення одиниця, яким параметр ***d*** стає рівним в процедурі, буде передано в основну програму.

Розглянемо алгоритм роботи процедури на прикладі обробки масивів. Необхідно знайти сумарне максимальне значення елементів трьох масивів, що формуються випадковим чином, а також місцерозташування максимального елементу. На рис.6.3 показаний результат виконання програми. Для виведення значень елементів масивів використаний компонент **StringGrid1**, для виведення сумарного значення максимуму і його номера використано два компоненти **LabeledEdit**, які відрізняються від раніше вивчених компонентів **Edit** і **Label**, тим, що суміщають обидві функції – вивід на екран результату і пояснюючого тексту. Пояснюючий текст задається в підвластивості **Caption** властивості **EditLabel**.

Розглянемо лістинг програми. Оскільки процедура **find** буде використана в двох процедурах програми, то її опис поміщений в розділ **implementation.**

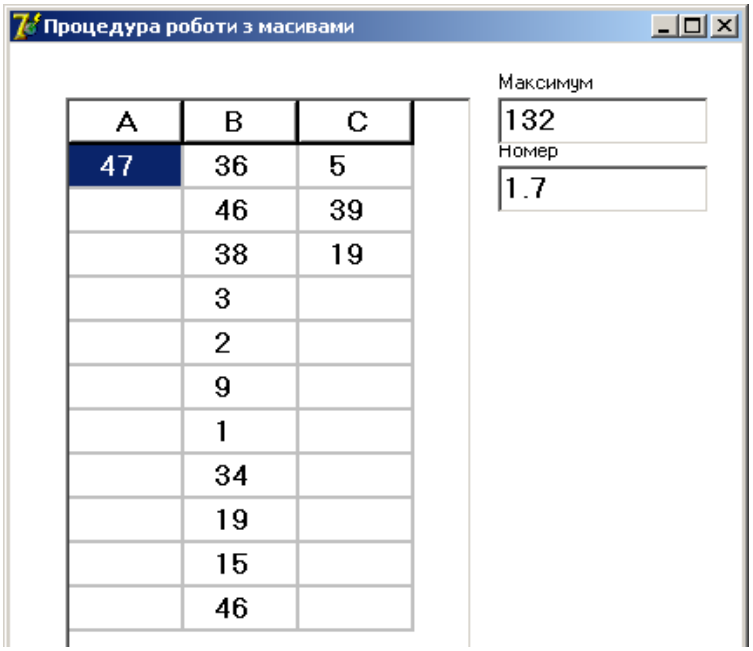


Рисунок 6.3 - Обробка масивів за допомогою процедури

Для роботи процедури необхідно передати масив і його розмірність, результатом виконання буде значення максимального елементу масиву і його номер. Заголовок процедури має вигляд

procedure find(XX:mas; n:integer; var max,k:integer);

де ***XX*** – масив, що передається; ***n***- його розмірність (параметри- константи); ***max, k*** – значення максимального елементу масиву і його позиції відповідно (параметри-змінні).

У першій процедурі **TForm1.FormCreate** формувалися масиви, визначалася кількість рядків в таблиці і вона заповнювалася елементами масивів.

Розмірність масивів формувалася випадковим чином, так розмірність першого масиву визначалася за формулою nn[1]:=random(5)+1; де ***nn[1]*** – розмірність масиву А. Оскільки при виробленні випадкових чисел виходять значення від нуля до вказаної межі, то до кожної з розмірностей додана одиниця для запобігання появі масиву з нульовою кількістю елементів.

Розмірність таблиці Stringgrid1, в яку виводяться значення елементів масиву, залежить від величини максимальної розмірності масивів. Кількість рядків в ній визначалася за допомогою тієї ж процедури Find:

find(nn,3,maxn,k);
stringgrid1. rowcount:=maxn;

Таблиця заповнювалася випадковими значеннями в межах зазначеної раніше розмірності кожного масиву.

У другій процедурі обчислювалися значення максимальних елементів масивів, їх суми і середнє місцерозташування максимального елементу.

```
implementation
  type mas=array[1..20] of integer;
   var  A,B,C,nn : mas;
    procedure find(XX:mas; n:integer; var max,k:integer);
   var i:integer; { Процедура пошуку максимального елементу
масиву і його номера}
   begin
   max:=xx[1]; k:=1;
For i:=2 to n do
   if max<xx[i] then
 begin
   max := xx[i];k:=i;
end;
end;
{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
var i,maxn,k:integer;
begin
  randomize;
            {Визначення розмірності масивів}
nn[1]:=random(5)+1;
nn[2]:=random(12)+1;
nn[3]:=random(8)+1;
    {Звернення до процедури Find для визначення максимальної
                        розмірності трьох масивів}
find(nn,3,maxn,k);
                  {Визначення кількості рядків в таблиці}
     with   stringgrid1  do
begin
          rowcount:=maxn;
                        {Вивід на екран заголовку}
          Cells[0,0]:='  A';
```

```
          Cells[1,0]:='  B';
         Cells[2,0]:='  C';
                              {Формування масивів}
          for i:=1 to nn[1] do
               begin
               A[i]:=random(50);
               Cells[0,i]:='   ' +inttostr(A[i]);
               end;
          for i:=1 to nn[2] do
                begin
               B[i]:=random(50);
               Cells[1,i]:='   ' +inttostr(B[i]);
                end;
          for i:=1 to nn[3] do
                 begin    C[i]:=random(50);
    Cells[2,i]:='   ' +inttostr(C[i]);  end;
    end;
end;

 procedure TForm1.LabeledEdit1SubLabelClick(Sender: TObject);
max1,k1, max2,k2,  max3,k3:integer;
ss:real;
xx:string;
begin
{Виклик процедури для визначення максимальних значень
кожного з трьох масивів}
  find(A,nn[1],max1,k1);
  find(B,nn[2],max2,k2);
  find(C,nn[3],max3,k3);
{Виведення на екран суми максимальних елементів і середнього
номера}
  LabeledEdit1.Text:=inttostr(max1+max2+max3);
  ss:=  (k1+k2+k3) /3 ;
  str(ss:1:1,xx);
  LabeledEdit2.Text:=xx;
end;
end.
```

Основні підсумки розділу

✓ Для багатократного виконання однієї і тієї ж послідовності дій з різними початковими даними використовуються підпрограми

✓ Підпрограми діляться на два види: функції і процедури. Функції використовуються, якщо результатом роботи підпрограми є одне значення.

✓ Процедури використовуються при необхідності отримання декількох результатів.

✓ Для опису функції використовують заголовок вигляду:

Function <ім'я функції> (параметри): тип результату;

✓ Параметри підпрограм поділяються на два види: формальні та фактичні.

✓ Під «формальними» параметрами розуміють перелік початкових даних, які потрібні для роботи підпрограми, і імена результатів її роботи.

✓ Фактичні параметри – це дані, які необхідні для обчислень і імена результатів. Порядок проходження, тип і кількість параметрів повинні співпадати.

✓ Для виклику підпрограми приводять її ім'я і перелік фактичних параметрів в круглих дужках. Виклик функції виконується у виразах.

✓ Залежно від того, де розміщений опис підпрограми, буде визначаться область її дії.

✓ Підпрограма, описана в розділі **public,** стає доступною для будь-яких класів, об'єктів і модулів програми.

✓ Підпрограма, описана в розділі **implementation,** стає доступною тільки в межах даного модуля.

✓ Опис підпрограми, поміщений в одну з процедур, дозволяє звертатися до неї тільки з цієї процедури.

✓ Якщо при роботі підпрограми планується отримання більш, ніж одного результату, використовують підпрограми-процедури.

✓ Заголовок підпрограми-процедури має вигляд

procedure <ім'я процедури>

(параметри-значення; var параметри-результати);

✓ Для виклику процедури існує спеціальний оператор виклику.

<ім'я процедури>(фактичні параметри);

✓ У заголовку процедури відсутній опис типу результату. Він розміщається в переліку формальних параметрів і відрізняється від початкових даних наявністю слова **var.**

✓ Формальні параметри діляться на два види: параметри-значення і параметри-змінні. Параметри-значення служать для передачі в підпрограму початкових даних для її роботи. Параметри-змінні використовуються для передачі початкових даних і повернення результатів роботи процедури.

✓ Опис процедури можна розміщувати або в розділі **implementation** або в локальній процедурі.

✓ При описуванні змінних, задіяних у програмі, їх розділяють на глобальні та локальні. Глобальні змінні діють у всьому тілі програми, включаючи підпрограми і звичайні процедури. Область дії локальних змінних обмежується тілом тієї процедури, у якої вони описані. Це приводить до того, що глобальні змінні зберігають свої значення у всій програмі. Проте, якщо у будь-якій процедурі ідентифікатори з однаковими іменами описані як локальні, то в межах цієї процедури вони діють як локальні.

Тести для самоперевірки знань до теми 6

Завдання 1–5 мають по чотири варіанти відповіді, серед яких лише ОДИН ПРАВИЛЬНИЙ. Виберіть правильний варіант відповіді.

1. Скільки видив підпрограм використовуються:

а) 3; в) 2;
б) 4; г) 5.

2. Функції використовуються, якщо результатом роботи підпрограми є:

а) одне значення; в) три значення;
б) два значення; г) багато значень.

3. Укажіть правильно оформлений опис функції, яка доступна тільки в межах даного модуля:

а) public function fact(r:byte);
б) implementation function fact(r:byte):longint;
в) public function fact(r:byte):longint;
б) implementation function fact(r:byte);

4. Укажіть правильно оформлений опис процедури, яка повертає в точку виклику результати своєї роботи:

а) procedure vas(n:byte;c:real; var e:real);
б) procedure vas(n,m:byte; c,d,e:real);
в) procedure vas(n,m:byte:c:real: var e:real);
г) procedure vas(n,m;byte;c:real; var d;byte);

5. Укажіть правильно оформлений виклик процедури, яка має такий опис **procedure vas(n:byte;c:real; var d:byte; var e:real)**:

а) D:=vas(1,3.4,dd,ee); в) vas(113.4,dd,ee);
б) vas(1.3,4,dd,ee); г) vas(1,3.4,dd,ee);

Завдання 6–7 мають два варіанти відповіді. Треба обрати ПРАВИЛЬНИЙ, щоб приведений вислів мав сенс.

6. У тілі функції обов'язково повинен бути оператор, що віддає імені функції результат.

Так | Ні

7. У тілі процедури обов'язково повинен бути оператор, що віддає імені процедури результат.

Так | Ні

Завдання 8–9 мають на меті встановлення відповідності. До кожного рядка, позначеного ЦИФРОЮ, доберіть відповідник, позначений БУКВОЮ.

8. Установіть відповідність назви параметрів для опису та виклику підпрограм**:**

1. *Формальні*
2. *Фактичні*

А. Виклик підпрограм
Б. Компіляція підпрограм
В. Опис підпрограм

	А	**Б**	**В**
1			
2			

9. Установіть відповідність назви параметрів при описуванні змінних, задіяних у програмі**:**

1. *Глобальні*
2. *Локальні*

А. Змінні, які діють у всьому тілі програми
Б. Змінні, які діють у тілі тієї процедури, у якої вони описані
В. Змінні, які діють у тілі процедур описаних нижче

	А	**Б**	**В**
1			
2			

Питання для самоконтролю

1. Що використовуються для багатократного виконання однієї і тієї ж послідовності дій з різними початковими даними?
2. На яких два види розділяють підпрограми?
3. Назвіть які види параметрів підпрограм вам відомі.
4. Чим відрязняються підпрограми, описані в розділі **public** і розділі **implementation?**

ЗАКЛЮЧЕННЯ

Пройдений курс є початковим при вивченні будь-якої алгоритмічної мови програмування. В нього увійшли тільки базові розділи. В цілому, розглянуті структури застосовуються практично у всіх сучасних мовах програмування, як універсальних, так і спеціалізованих. За відсутністю вичерпного довідкового матеріалу значну частину корисної інформації можна отримати, проводячи експерименти. Відправною крапкою до поглибленого вивчення програмування може стати список літератури, що рекомендується.

За відсутностю вичерпного довідкового матеріалу значну частину корисної інформації можна отримати, проводячи експерименти. Відправною точкою до поглибленого вивчення програмування може стати список літератури, що рекомендується

БІБЛІОГРАФІЧНИЙ ПЕРЕЛІК

1. Архангельский А.Я. Delphi 2006. Справочное пособие: Язык Delphi, классы, функции Win32 и NET. – М.: Бином, 2009. – 1152 с.

2. Культин Н.Б. Delphi в задачах и примерах. – СПб.: BHV, 2008. – 288 с.

3. Фаронов В.В. Delphi 2005. – СПб.: Питер, 2006. – 544 с.

4. Культин Н. Основы программирования в Delphi (+ CD-ROM). Самоучитель. – СПб.: БХВ - Петербург, 2009. – 640 с.

5. Бобровский С. Delphi 7. Учебный курс. – СПб.: Питер, 2008. – 736 с.

6. Чеснокова О. Delphi 2007. Алгоритмы и программы / Самоучитель. – М.: НТ Пресс, 2008. – 368 с.

7. Поган А.М., Царенко Ю.А. Программирование в Delphi. Просто как дважды два. – М.: Эксмо, 2007. – 320 с.

8. Белов В.В., Чистякова В.И. Программирование в Delphi: процедурное, объектно-ориентированное, визуальное / Учебное пособие для вузов. – М.: Горячая Линия-Телеком, 2009. – 240 с.

9. Методическое пособие по курсу «Информатика и компьютерная техника» (раздел «Программирование в среде Delphi) для студентов физико-металлургического факультета / Сост.: Павлыш В.Н., Анохина И.Ю., Кононенко И.Н. – Донецк: ДонНТУ, 2008. – 114 с.

Правильні відповіді до тестів

№ розділу	№ питання														
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	Б	В	А	Г	Г	В	А	Так	Так	1В,2Б, 3Г,4D	1В,2Д, 3Б 4Г	1А,2Г, 3Б,4В	1В,2Д, 3Б,4Г	1А,2Г 3Б,4В	
2	В	Б	А	В	А	В	Б	Б	Так	Так	Ні	1Г, 2А, 3Б	1Б, 2А, 3Г	1В,2А, 3Г	1Д, 2Г, 3Е, 4Ж, 5Б, 6В
3	Г	А	Б	Б	Ні	Так	Так	Так	1Г, 2В, 3Д, 4Б, 5Е	1В, 2Г, 3А, 4Д, 5Б	1В, 2А, 3Г, 4Б				
4	Б	А	Б	В	В	Ні	Так	1Г, 2А, 3Д, 4Б; 5Е	13, 2В, 3А, 4Д, 5Ж; 6Г, 7Б	1Б, 2В, 3А	1В, 2Г, 3Б, 4Е, 5А				
5	В	Б	А	Ні	Ні	Так	Ні	1Г, 2А, 3Д, 4Б; 5Е	1Д, 2А, 3В, 4Б	1Г, 2А, 3Д, 4Б					
6	В	А	Б	А	Г	Так	Ні	1Б, 2В	1А, 2Б						

Додаток А
Повідомлення про помилки компілятора Delphi

A

Array type required	Потрібний масив.
Assignment to FOR-Loop variable '<Ім’я>'	Змінна - параметр циклу FOR не може змінюватися усередині циклу.
Assignment to FOR-Loop variable '<Ім’я>'	Змінна - параметра циклу FOR може мати невизначене значення після виконання цього циклу.

C

Case label outside of range of case expression	Значення вказані в операторові CASE такі, що селектор не може їх приймати.
Compile terminated by user	Компіляція перервана пользовате-лем клавішами Ctrl і Break.
Constant expression expected	Очікувався вираз що складається з одних констант.
Constant expression violates subrange bounds	Константний вираз виходить за межі заданого інтервалу.

D

Declaration of <Ім'я> differs from previous declaration	Поточне декларування відрізняється від попереднього.
Division by zero	Виконується ділення на нуль.
Duplicate case label	Значення в операторові CASE повторюються.

E

EXCEPT or FINALLY expected	Очікується секція виключень оператора TRY.
Expression has no value	Вираз не повертає після себе ніякого значення.

F

File not found: '<Ім’я_файла>.dcu'	Файл зовнішнього модуля не був виявлений в поточному каталозі.
File not found: <Ім’я_файла>	Не був знайдений вказаний файл.

File type not allowed here	Файловий тип в даному місці не дозволений. Файлова змінна може передаватися в підпрограму тільки як параметр - змінна.
FOR-Loop variable '<Ім’я>' cannot be passed as var parameter	Змінна - параметр циклу не може передаватися в підпрограму як параметр. Вона повинна бути описана локально.
For loop control variable must be simple local variable	Змінна - параметр циклу FOR повинна бути описана локально.
For loop control variable must have ordinal type	Змінна параметра циклу FOR може бути цілого, символьного або такого, що перераховує типів.
FOR or WHILE loop executes zero times - deleted	Цикл FOR або WHILE не виконуватиметься жодного разу і тому видалений.
Function needs result type	Була визначена функція в якій не визначено результуюче значення.

I

Identifier redeclared: '<Ім’я>'	Повторно був описаний вказаний ідентифікатор. Всі імена в програмі повинні бути унікальні.
Illegal character in input file: '<Символ>' ($)	У програмі були виявлені неприпустимі символи.
Illegal type in Read/Readln statement	Несумісність типів при виклику оператора введення.
Illegal type in Write/Writeln statement	Несумісність типів при виклику оператора виводу.
Inaccessible value	Значення даної змінної не доступне або не визначене.
Incompatible types	Несумісність типів.
Incompatible types: '<Тип1>' and '<Тип2>'	Має місце несумісність вказаних типів.
Internal error: <Код_помилки>	Внутрішня помилка.
Invalid function result type	Була визначена функція, в якій вихідне значення не сумісне за типом з вказаним в заголовку.

L

Label already defined: '<Ім'я_метки>'	Вказана мітка вже була визначена.
Left side cannot be assigned to	Має місце спроба змінити значення константи
Line too long (more than 255 characters)	Рядок дуже довгий.
Low bound exceeds high bound	Нижня межа діапазону визначена більшою чим верхня межа.

M

Method identifier expected	Потрібно вказати назву методу.
Missing operator or semicolon	Пропущений оператор або крапка з комою.
Missing parameter type	Формальні параметри підпрограми дані без вказівки типів.

N

Not enough actual parameters	При виклику який - або процедури або функції було вказано недостатня кількість фактичних параметрів.

O

Object type required	Потрібний об'єкт.
Operator not applicable to this operand type	Оператор не призначений для роботи з операндами даного типу.
Ordinal type required	У даному місці потрібний скалярний тип.

P

Pointer type required	Потрібний покажчик.
Procedure cannot have a result type	Процедура не може мати результирующего значення. У таких случаях потрібно використовувати функцію.
PROCEDURE or FUNCTION expected	У даному місці очікується процедура або функція.

R

Record, object or class type required	Потрібний запис, об'єкт або клас.
Return value of function '<Ім'я_функції>' might be undefined	Вихідне значення вказаної функції може бути не визначене.

S

Statement expected, but expression of type '<Тип>' found	Очікувався оператор, але було виявлено вираз вказаного типу.
String constant truncated to fit STRING[<Номер>]	Строкова константа скорочена.
Syntax error in real number	Помилка при написанні дійсного числа.

T

Text after final 'END.' - ignored by compiler	Текст, написаний в програмі після останнього END був игнорирован.
Too many actual parameters	При виклику який - або процедури або функції було вказано дуже велика кількість фактичних параметрів.
Type of expression must be BOOLEAN	Тип виразу повинен бути логічним. Має місце несумісність типів.
Type of expression must be INTEGER	Тип виразу повинен бути цілим. Має місце несумісність типів.
Types of actual and formal var parameters must be identical	Типи формальних і фактичних параметрів повинні співпадати.

U

Undeclared identifier: '<Ім'я>'	Невідомий ідентифікатор, або оператор написаний з помилкою.
Unexpected end of file in comment started on line <Номер>	Коментар початий у вказаному рядку не закритий.
User break - compilation aborted	Компіляція перервана за бажанням користувача.
Unknown directive:	Невідома директива або написана з

'<Директива>'	помилкою.
Unsatisfied forward or external declaration: '<Ім’я_процедури>'	Вказане декларування не має реалізації.

V

Value assigned to '<Ім’я>' never used	Значення визначене для вказаної змінної не було жодного разу використано.
Variable '<Ім’я>' might not have been initialized	У вказаної змінної не було визначено початкове значення.

***

<Директива1> expected but <Директива2> found	Була виявлена директива 2, хоча очікувалася директива1.
'<Ім’я>' is not a type identifier	Вказане ім'я не є назвою типу.
'';' not allowed before 'ELSE'	Крапка з комою не ставиться перед ELSE.

Додаток Б

Функції виділення цілої або дробової частини

Frac(X)	Повертає дробову частину аргументу X.
Int(X)	Повертає цілу частину дійсного числа X. Результат належить речовому типу.
Round(X)	Округляє дійсне число X до цілого.
Trunc(X)	Повертає цілу частину дійсного числа X. Результат належить цілому типу.

Приклади:

Вираз	***Результат***
Frac(2.5)	0.5
Int(2.5)	2.0
Round(2.5)	3
Trunc(2.5)	2

Додаток В

Функції генерації випадкових чисел

Random	Повертає випадкове дійсне число в діапазоні 0 ? X < 1.
Random(I)	Повертає випадкове ціле число в діапазоні 0 ? X < I.
Randomize	Наново ініціалізував вбудований генератор випадкових чисел новим значенням, отриманим від

системного таймера.

Додаток Г

Підпрограми для роботи з порядковими величинами

Chr(X)	Повертає символ, порядковий номер якого рівний X.
Dec(X [N])	Зменшує цілу змінну X на 1 або на задане число N.
Inc(X [N])	Збільшує цілу змінну X на 1 або на задане число N.
Odd(X)	Повертає True, якщо аргумент X є непарним числом.
Ord(X)	Повертає порядковий номер аргументу X в своєму діапазоні значень.
Pred(X)	Повертає значення, передування значенню аргументу X в своєму діапазоні.
Succ(X)	Повертає значення, наступне за значенням аргументу X в своєму діапазоні.

Приклади:

Вираз	***Результат***
Chr(65)	'A'
Odd(3)	True
Ord('A')	65
Pred('B')	'A'
Succ('A')	'B'

Додаток Д

Підпрограми для роботи з датою і часом

Date	Повертає поточну дату у форматі TDateTime.
Time	Повертає поточний час у форматі TDateTime.
Now	Повертає поточну дату і час у форматі TDateTime.
DayOfWeek(D)	Повертає день тижня по даті у форматі TDateTime.
DecodeDate(...)	Розбиває значення дати на рік, місяць і день.
DecodeTime(...)	Розбиває значення часу на годину, хвилини, секунди і милисекунды.
EncodeDate(...)	Формує значення дати по року, місяцю і дню.
EncodeTime(...)	Формує значення часу по годині, хвилинам, секундам і милисекундам.
DateTimeToStr(...)	Перетворює дату і час **DateTime** в рядок.
DateToStr	Перетворює дату **Date** в рядок
TimeToStr(...)	Перетворює час **Time** в рядок.

Додаток Ж

Процедури передачі управління

Break	Перериває виконання циклу.
Continue	Починає нове повторення циклу.
Exit	Перериває виконання поточного блоку.
Halt	Зупиняє виконання програми і повертає управління операційній системі.
RunError	Зупиняє виконання програми, генеруючи помилку часу виконання.

Додаток К

Різні процедури і функції

FillChar(...)	Заповнює безперервну область символьним або байтовим значенням.
Hi(X)	Повертає старший байт аргументу X.
High(X)	Повертає саме старше значення в діапазоні аргументу X.
Lo(X)	Повертає молодший байт аргументу X.
Low(X)	Повертає саме молодше значення в діапазоні аргументу X.
Move(...)	Копіює задану кількість байт з однієї змінної в іншу.
ParamCount	Повертає кількість параметрів, переданих програмі в командному рядку.
ParamStr(X)	Повертає параметр командного рядка по його номеру.
SizeOf(X)	Повертає кількість байт, займане аргументом X в пам'яті. Функція SizeOf особливо потрібна для визначення розмірів змінних обощенных типів даних, оскільки представлення обощенных типів даних в пам'яті може змінюватися від однієї версії середовища Delphi до іншої. Рекомендуємо завжди використовувати цю функцію для визначення розміру змінних будь-яких типів даних; це вважається хорошим стилем програмування.
Swap(X)	Міняє місцями значення старшого і молодшого байтів аргументу.
UpCase(C)	Повертає символ C, перетворений до верхнього регістра.

Анохіна Інна Юріївна
Кононенко Ірина Миколаївна

КОНСПЕКТ ЛЕКЦІЙ ПО КУРСУ «ІНФОРМАТИКА ТА ОБЧИСЛЮВАЛЬНА ТЕХНІКА» ДЛЯ СТУДЕНТІВ ФІЗИКО-МЕТАЛУРГІЙНОГО ФАКУЛЬТЕТУ

Комп'ютерна верстка
Відповідальний за випуск
Технічний редактор

Підп. до друку . .2010 р. Формат 60х84 $^{1}/_{16}$. Папір офсетний.
Різографічний друк. Ум.-др. арк. . Ум.фарб.-відб. .
Облік.-вид. арк. . Тираж прим. Замовл. № .

Друк з оригінал-макету МПП "ВІК"
Свідоцтво про реєстрацію ДК №382 від 26.03.2001 р.
83059, м. Донецьк, вул. Разенкова, 12/17, тел. (062) 381-70-87