МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Т.М. Усатенко

ПРОГРАМУВАННЯ В СЕРЕДОВИЩІ DELPHI

Навчальний посібник

з дисципліни "Програмування"
для студентів механіко-математичного факультету
всіх форм навчання

Суми
Вид-во СумДУ

2004

ББК 32.973-01я73
У74
УДК 004.42(075.8)

Рекомендовано до друку вченою радою Сумського державного університету

Рецензенти:
д-р техн. наук, проф. Косторний С.Д. (Сумський національний аграрний університет),
д-р фіз. – мат. наук, проф. Малютін К.Г. (Українська академія банківської справи, м.Суми)

Усатенко Т.М.
У74 Програмування: Навчальний посібник. – Суми: Вид-во СумДУ, 2004. - 84 с.

ISBN 966-657-031-9

Посібник містить необхідний мінімум теоретичних відомостей про середовище програмування Delphi, приклади програмування в Delphi з докладними коментарями до них, а також завдання для самостійної роботи.

Посібник може бути корисним як для студентів і аспірантів, так і для широкого кола фахівців, що займаються складанням програм під Windows.

ББК 32.973-01я73

ISBN 966-657-031-9

Зміст

Передмова

Середовище програмування Delphi було розроблене компанією Borland у 1993 році. Назва програмного продукту походить від назви давньогрецького міста Дельфи, що у Фокіді. Delphi є нащадком програмного продукту Borland Pascal, у якому в повному обсязі використовується мова програмування Object Pascal.

Середовище Delphi – це комбінація декількох найважливіших технологій, основними характеристиками якого є:

- потужний компілятор у машинний код;
- об'єктно-орієнтована модель компонент;
- візуальна та досить швидка побудова додатків із програмних прототипів;
- засоби, які масштабуються, для побудови баз даних.

Компілятор, який вмонтований в середовище Delphi, забезпечує високу швидкість, необхідну для побудови додатків в архітектурі "клієнт-сервер", пропонує легкість розроблення та короткий час для перевірки готового програмного блока, забезпечує відповідну якість програмного коду.

Основний акцент об'єктно-орієнтованої моделі програмних компонент у середовищі Delphi робиться на максимальному повторному використанні коду. Це дає змогу програмістам будувати додатки досить швидко із раніше розроблених об'єктів, а також можливість створювати свої власні об'єкти для середовища Delphi. Ніяких обмежень щодо типів об'єктів, які можуть бути створені розробниками, не існує.

Процес розроблення програмного додатка в Delphi максимально спрощено. У першу чергу це стосується створення інтерфейсу, на який витрачається 80% усього

часу розроблення програми. Досить розмістити необхідні компоненти на поверхні Windows-вікна, змінити їх властивості та зв'язати події цих компонент (натискання на кнопку, вибір елемента зі списку та ін.) з кодом обробки відповідних подій – і простий проект готовий до роботи.

Таким чином, середовище візуального програмування Delphi може бути використано для створення як найпростіших додатків, так і складних корпоративних проектів.

У посібнику стисло викладені структура та основні складові частини середовища DELPHI; розглянуті такі поняття, як палітра компонентів та складові частини стандартної палітри компонентів, форма та редактор коду додатка, інспектор об'єктів, подія та процедура обробки події. Наведені приклади створення додатків для операційної системи WINDOWS з докладними поясненнями, порадами та зауваженнями. Посібник також містить завдання для самостійної роботи студентів.

Структура середовища програмування Delphi

Зовнішній вигляд середовища програмування Delphi відрізняється від багатьох інших, які можна побачити у Windows. Приміром, Borland Pascal for Windows 7.0, Borland C++ 4.0, Word for Windows, Program Manager - це всі MDI додатки і мають інший вигляд, ніж Delphi. MDI (Multiple Document Interface) - визначає особливий спосіб керування декількома дочірніми вікнами усередині одного великого вікна.

Середовище Delphi належить до іншої специфікації - Single Document Interface (SDI) і складається з декількох окремо розташованих вікон.

Перед початком роботи з будь-яким SDI-додатком краще мінімізувати інші додатки, щоб їх вікна не захаращували робочий простір. Якщо потрібно переключитися на інший додаток, то просто клацніть мишкою на системну кнопку мінімізації Delphi. Разом із головним вікном згорнуться всі інші вікна середовища програмування, звільнивши місце для роботи інших програм.

Головні складові частини Delphi

Нижче перелічені головні складові частини Delphi:

1. *Дизайнер Форм* (Form Designer).
2. Вікно *Редактора* Вихідного Тексту (Editor Window).
3. *Палітра Компонентів* (Component Palette).
4. *Інспектор Об'єктів* (Object Inspector).
5. *Довідник* (On-line help).

Є, звичайно, й інші важливі складові Delphi (лінійка інструментів, системне меню та інші), які необхідні для настроювання програми і середовища програмування.

При створенні програми-додатка в середовищі Delphi програмісти більшість часу проводять у вікнах

Дизайнера Форм і *Редактора Вихідного Тексту* (яке для стислості називають *Редактор*).

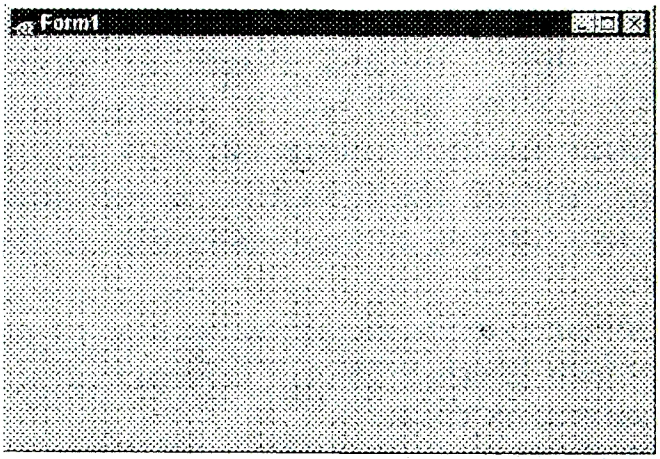

Рисунок 1 - Дизайнер Форм

Порада. Перш ніж розпочати роботу, слід переконатися, що ви в змозі розпізнати ці два важливих елементи. *Дизайнер Форм* (рис.1) - місце, де створюється візуальний інтерфейс програми, вікно *Редактора* (рис.2) – місце, де створюється логіка керування програмою.

Дизайнер Форм у Delphi настільки інтуїтивно зрозумілий і простий у використанні, що створення візуального інтерфейсу перетворюється у дитячу гру. *Дизайнер Форм* спочатку складається з одного порожнього вікна, що заповнюється всілякими об'єктами, обраними на *Палітрі Компонентів*.

Незважаючи на всю важливість *Дизайнера Форм*, основним залишається *Редактор*. Логіка є рушійною силою програми, і *Редактор* - те місце, де програміст її "кодує".

Палітра Компонентів (рис.3) дозволяє вибрати потрібні об'єкти для їх розміщення на *Дизайнері Форм*. Для використання *Палітри Компонентів* необхідно перший раз

клацнути мишкою на один із об'єктів, а другий раз - на *Дизайнері Форм*.

Обраний у такий спосіб об'єкт з'явиться на проектованому вікні. Ним можна маніпулювати за допомогою миші.

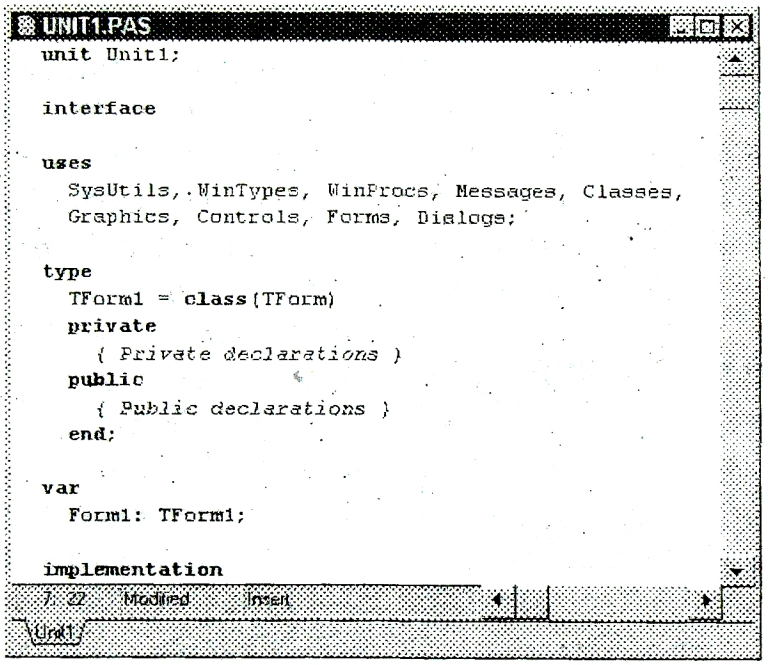

Рисунок 2 - Вікно Редактора

Палітра Компонентів використовує посторінкове угрупування об'єктів. Унизу (вгорі) *Палітри* міститься набір закладок - *Standard, Additional, Dialogs* та інші. Якщо вибрати одну із закладок, можна перейти на наступну сторінку *Палітри Компонентів*.

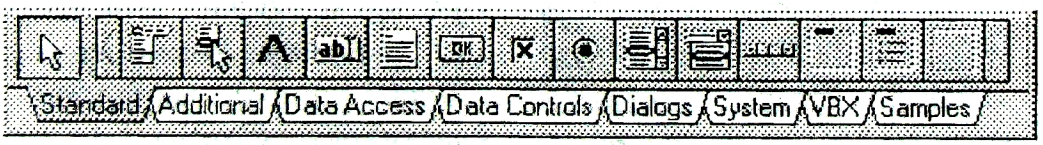

Рисунок 3 - Палітра Компонентів

Припустимо, що на формі розташований компонент *TEdit*. Його можна пересувати з місця на місце;

змінювати його розміри, використовуючи межу, накреслену навколо об'єкта. Таким же чином можна маніпулювати і більшістю інших компонентів. Однак невидимі під час виконання програми компоненти (наприклад, *TMenu, TDataBase* та інші) не змінюють своєї форми.

Ліворуч від *Дизайнера Форм* розташований *Інспектор Об'єктів* (рис.4). Слід відзначити, що інформація в *Інспекторі Об'єктів* змінюється в залежності від об'єкта, обраного на формі. Важливо зрозуміти, що кожен *компонент_є об'єктом* і за допомогою *Інспектора Об'єктів* можна змінювати його вигляд і поведінку.

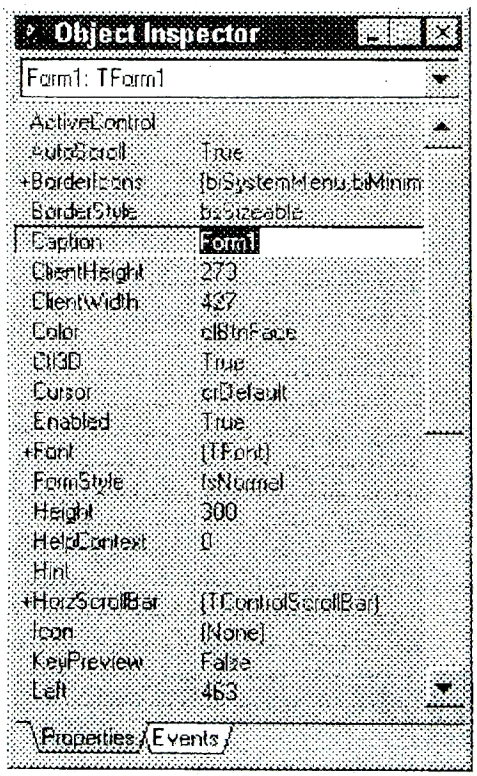

Рисунок 4- Інспектор Об'єктів

Інспектор Об'єктів складається з двох сторінок, кожну з яких можна використовувати для визначення поведінки даного компонента. Перша сторінка - це список властивостей (*Properties*), друга - список подій (*Events*). Якщо потрібно змінити що-небудь, пов'язане із визначеним компонентом, то це звичайно робиться в *Інспекторі Об'єктів*. Приміром, можна змінити ім'я і розмір компонента *Tlabel*, змінюючи властивості *Caption, Left, Top, Height,* і *Width.*

Можна використовувати закладки внизу (вгорі) *Інспектора Об'єктів* для переключення між сторінками

властивостей і подій. Сторінка подій пов'язана з *Редактором*. Якщо двічі клацнути мишкою на правий бік якого-небудь пункту, то відповідний даній події код автоматично запишеться в *Редактор*, сам *Редактор* негайно одержить фокус і програміст відразу одержує можливість додати код оброблювача даної події.

Остання важлива частина середовища Delphi - *Довідник* (on-line help). Для доступу до цього інструмента потрібно просто вибрати в системному меню пункт *Help* і потім *Contents*.

Довідник Delphi є контекстно-залежним: при натисканні клавіші F1 програміст одержить підказку, що відповідає поточній ситуації.

Додаткові елементи Delphi

У даному параграфі основна увага приділяється трьом інструментам, які можна сприймати як допоміжні у середовищі Delphi:

- Меню (*Menu System*).
- Панель із кнопками для швидкого доступу (*SpeedBar*).
- Редактор рисунків (*Image Editor*).

Меню надає швидкий і гнучкий інтерфейс середовищу Delphi, тому що може керуватися набором "гарячих клавіш". Це зручно ще і тому, що тут використовуються слова або короткі фрази, більш точні і зрозумілі, ніж іконки чи піктограми. Можна використовувати меню для виконання широкого кола найбільш загальних задач: відкриття і закриття файлів, керування налагоджувачем, настроювання середовища програмування і т.п.

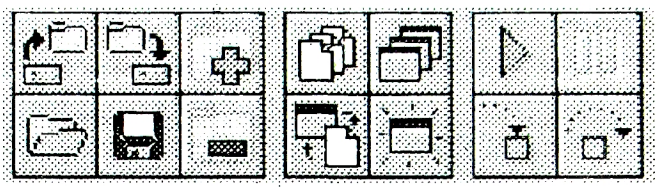

Рисунок 5 - Панель *SpeedBar*

Панель *SpeedBar* (рис.5) розміщена безпосередньо під меню, ліворуч від *Палітри Компонентів*. *SpeedBar* виконує багато з того, що можна зробити через меню. Якщо затримати мишу над кожною з іконок *SpeedBar*, то з'явиться підказка, що пояснює призначення даної іконки.

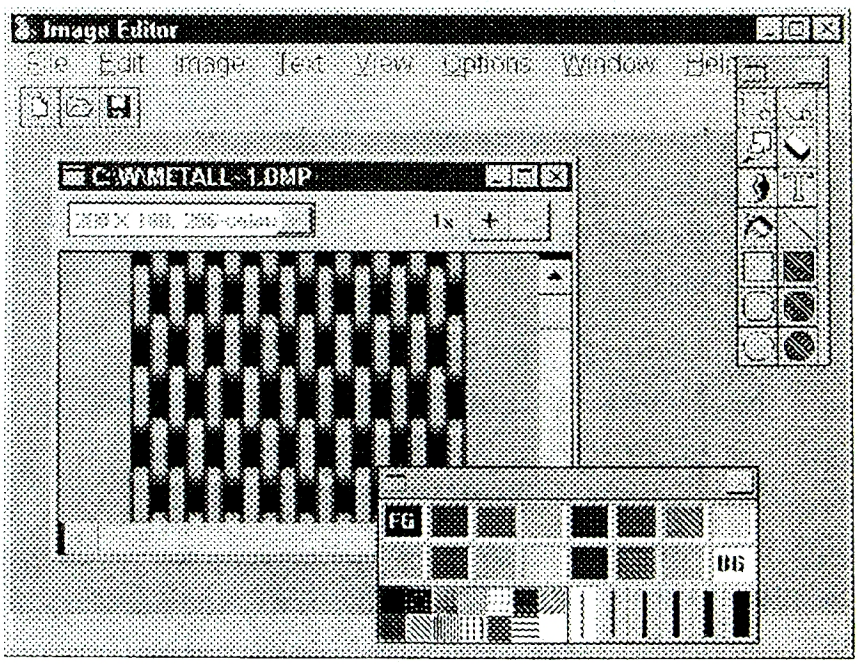

Рисунок 6 - Редактор Малюнків

Редактор Малюнків (рис.6) призначений для створення малюнків, кнопок, іконок та інших візуальних частин для програми-додатка. Редактор працює аналогічно до програми Paintbrush з Windows. Одержати доступ до цього модуля можна, якщо вибрати пункт меню *Tools → Image Editor*.

Стандартні компоненти

Для подальшого знайомства із середовищем програмування Delphi потрібно розповісти про склад першої сторінки *Палітри Компонентів - Standard* (рис.7).

На першій сторінці *Standard* розміщено 14 об'єктів, набір і порядок яких є конфігурованим: можна додати до наявних компонентів нові, змінити їх кількість і порядок.

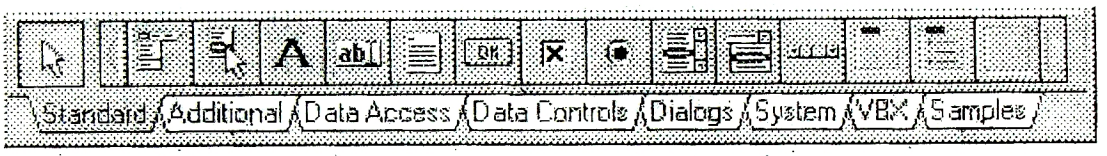

Рисунок 7 – Сторінка Standard Палітри Компонент

Стандартні компоненти Delphi перелічені нижче з деякими коментарями щодо їх застосування.

> ***Порада.*** При вивченні даних компонентів було б корисно мати під рукою комп'ютер для того, щоб подивитися, як вони працюють і як ними маніпулювати.

TMainMenu  дозволяє помістити головне меню в програму. При розташуванні об'єкта *TMainMenu* на форму має вигляд іконки. Об'єкти даного типу називають *невидимими компонентами*, оскільки вони не видимі під час виконання програми. Створення меню відбувається в три етапи:

1) розташування *TMainMenu* на формі;
2) виклик *Дизайнера Меню* через властивість *Items* в *Інспекторі Об'єктів;*
3) визначення пунктів меню в *Дизайнері Меню.*

TPopupMenu  дозволяє створювати висхідні меню. Цей тип меню з'являється при одноразовому натисканні правої кнопки миші.

TLabel  застосовується для відображення тексту на екрані; дозволяє змінити шрифт і колір тексту позначки, якщо двічі натиснути ліву кнопку миші на властивість *Font* в *Інспекторі Об'єктів*.

TEdit  - стандартний керуючий елемент Windows для введення інформації . Він може бути використаний для відображення короткого фрагмента тексту і дозволяє користувачу вводити текст під час виконання програми.

TMemo  - інша форма TEdit. Використовується при роботі з великими текстами. TMemo може переносити слова, зберігати в Clipboard фрагменти тексту і відновлювати їх, а також виконує основні функції редактора. TMemo має обмеження на обсяг тексту в 32Кб, це складає 10-20 сторінок. (Існують VBX і "рідні" компоненти Delphi, де ця межа стерта).

TButton  дозволяє виконати які-небудь дії при натисканні кнопки під час виконання програми. У Delphi все це робиться дуже просто. Об'єкт *TButton* розташовується на формі; подвійним натисканням лівої кнопки миші на *TButton* здійснюється перехід до *Редактора*, де автоматично розкривається те місце модуля, яке відповідає за *обробку події натискання кнопки*. Далі потрібно заповнити шаблон кодом (підкреслене те, що потрібно написати вручну):

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  MessageDlg('Are you there?', mtConfirmation, mbYesNoCancel, 0);
end;
```

TCheckBox  відображає рядок тексту з маленьким віконцем поруч, в якому можна поставити відмітку, що вказує на вибір даної опції. Дозволяє створити набір опцій з можливістю вибору декількох із них. Наприклад, якщо подивитися вікно діалогу настроювань компілятора (пункт меню *Options → Project*, сторінка *Compiler*), то можна побачити, що воно складається переважно із *CheckBox*'ов.

TRadioButton  дозволяє створювати набір опцій з можливістю вибору лише однієї опції з декількох. Якщо відкрити діалог *Options → Project* і вибрати сторінку *Linker Options*, то можна побачити, що секції *Map file* і *Link buffer file* складаються з наборів *RadioButton*.

TListBox  використовується для показу списку, з можливістю його прокручування. Класичний приклад *ListBox*'а в середовищі Windows - вибір файлу із списку в пункті меню *File → Open* багатьох додатків. Назви файлів чи директорій і знаходяться в *ListBox*'е.

TComboBox  багато в чому нагадує *ListBox*, за винятком того, що дозволяє вводити інформацію в рисенькому полі введення зверху *ListBox'a*. Існує декілька типів *ComboBox*, але найбільш популярний спадний униз (*drop-down combo box*), який можна бачити внизу вікна діалогу вибору файлу.

TScrollbar  використовується для вставки смуги прокручування.

TGroupBox  використовується для вказівки Windows, у якому порядку виконувати переміщення по компонентах на формі (при натисканні клавіші *TAB*), а також для візуальної мети.

TPanel  - керуючий елемент, схожий на *TGroupBox*, використовується з декоративною метою. Для використання *TPanel* необхідно, по-перше, розташувати його на формі, по-друге, покласти на нього інші компоненти. Тепер при переміщенні *TPanel* будуть пересуватися всі компоненти, розташовані на *TPanel*. Прикладом використання *TPanel* є лінійка інструментів будь-якого додатка Windows.

TRadioBox  являє собою спеціальне місце на формі, що містить тільки об'єкти *TRadioButton*.

За необхідності можна одержати додаткову інформацію, вибравши на *Палітрі Об'єктів* об'єкт і натиснувши клавішу F1 - з'явиться *Довідник* з повним описом даного об'єкта.

Інспектор Об'єктів

Розглянемо докладніше *Інспектор Об'єктів* (*Object Inspector*). Основне для розуміння *Інспектора Об'єктів* полягає в тому, що *він використовується для зміни характеристик будь-якого об'єкта, розташованого на формі і для зміни властивостей самої форми.*

Розглянемо на прикладі, як працює *Інспектор Об'єктів*.

1 Відкрийте новий проект (пункт меню *File* → *New Project)*.

2 Розташуйте на формі об'єкти *TMemo*, *TButton*, і *TListBox*, як показано на рисунку 8.

Спершу розглянемо роботу з властивостями об'єкта на прикладі властивості *Ctl3D* (за замовчуванням властивість активізована).

3 Виберіть *Форму* за допомогою одноразового натискання на ній лівою кнопкою миші. Перейдіть в *Інспектор Об'єктів* і кілька разів змініть значення властивості *Ctl3D* . Зверніть увагу на те, що ця дія радикально змінює зовнішній вигляд *Форми*. Зміна властивості *Ctl3D Форми* автоматично змінює властивість *Ctl3D* кожного дочірнього вікна, розташованого на цій *Формі*.

4 Перейдіть на Форму і змініть значення Ctl3D на *True*. Тримаючи натиснутою клавішу *Shift*, виберіть *TMemo* , а потім *TListBox*. Тепер обидва об'єкти мають по краях рисенькі квадратики, які вказують на те, що об'єкти обрані.

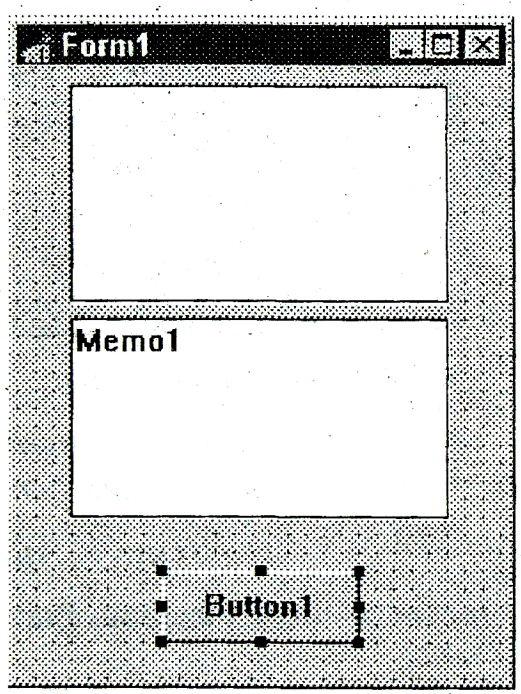

Рисунок 8 - Простий об'єкт *TForm* з компонентами *TMemo*, *TButton* і *TListBox*

Якщо вибрати одночасно декілька об'єктів, можна виконати над ними велику кількість операцій, наприклад,

пересувати по *Формі*, змінювати розмір і розташування об'єктів і т.п. При виборі декількох компонентів вміст *Інспектора Об'єктів* зміниться - будуть показані тільки ті поля, які є загальними для обраних об'єктів. Це означає, що зміни у властивостях, зроблені користувачем, вплинуть не на один, а на всі обрані об'єкти.

5 Виберіть пункт меню *Edit* → *Size* і встановіть значення Ширини (*Width*) та Висоти (*Height*) у *Grow to Largest*, як показано на рисунку 9.

Тепер обидва об'єкти стали однакового розміру.

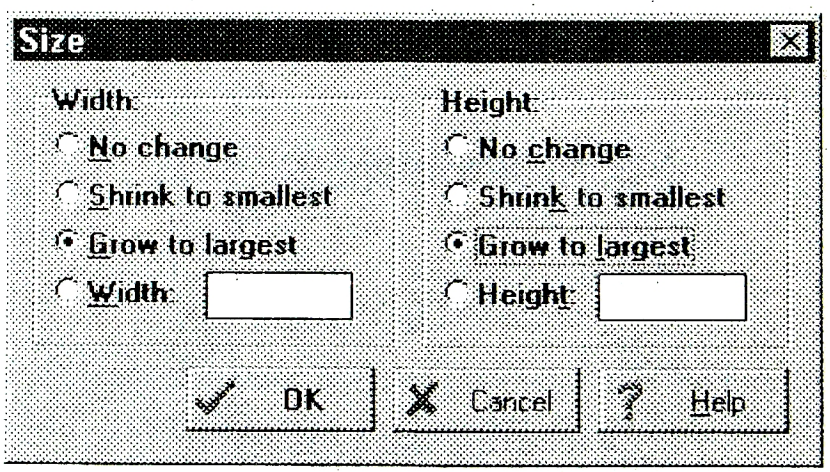

Рисунок 9 - Пункт меню Edit

6 Виберіть пункт меню *Edit* → *Align* і встановіть вирівнювання по горизонталі в значення *Center* (рис.10).

7 Змініть властивість *Color* об'єкта *TListBox*.

Існує три способи змінити значення властивості *Color* в *Інспекторі Об'єктів:*

- просто надрукувати ім'я кольору (*clRed*) або номер кольору;
- натиснути на рисеньку стрілку праворуч і вибрати колір із списку;

- двічі натиснути лівою кнопкою миші на поле введення властивості *Color*. При цьому з'явиться діалог вибору кольору.

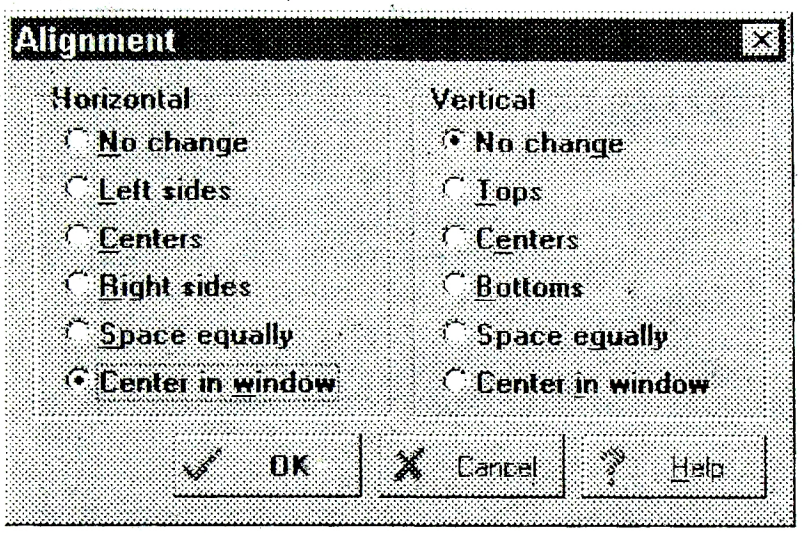

Рисунок 10 - Діалог Alignment

8 Змініть властивість *Font* об'єкта *TMemo*. Для цього необхідно спочатку вибрати властивість *Font* для об'єкта *TMemo* і двічі натиснути лівою кнопкою миші на поле введення. З'явиться діалог настроювання шрифту, як показано на рисунку 11. Потім вибрати шрифт *New Times Roman*, встановити його розмір 26 і змінити колір. Після натискання кнопки *OK* вигляд тексту в об'єкті *TMemo* радикально зміниться.

9 Введіть текст у *ListBox*. Для цього необхідно двічі натиснути лівою кнопкою миші на властивість *Items* об'єкта *ListBox*. З'явиться діалог, у який можна ввести рядки тексту.

Надрукуйте кілька слів, по одному на кожному рядку, і натисніть кнопку *OK*. Текст відобразиться в *ListBox*.

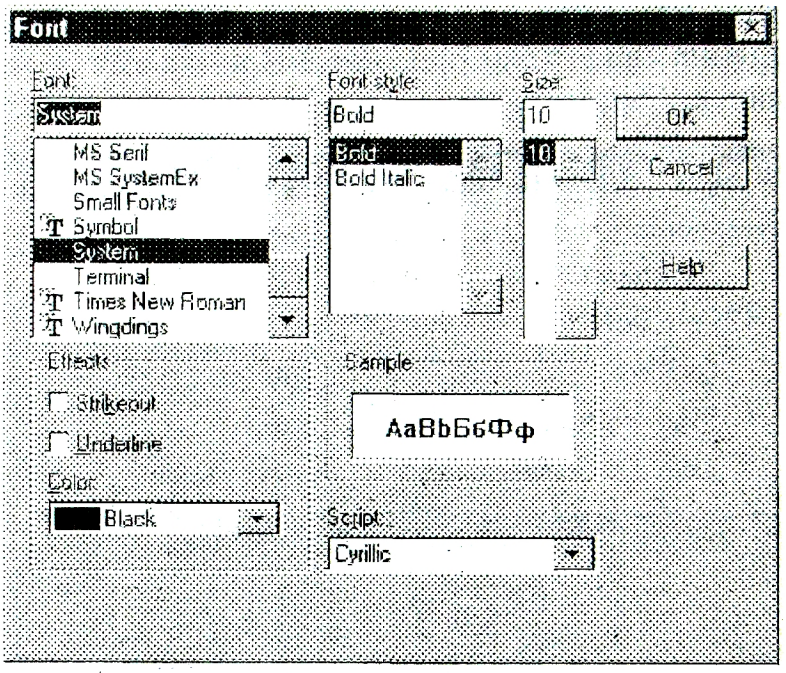

Рисунок 11 - Діалог вибору шрифту

10 Збережіть проект. Збереження проекту в Delphi відбувається у два етапи.

Перший етап. Створіть піддиректорію для проекту.

> ***Порада.*** Найкраще створити директорію, де будуть зберігатися усі Ваші програми, а у ній - створити піддиректорію для даної конкретної програми.

Другий етап. Виберіть пункт меню *File→ Save Project.* Збережіть два файли: перший - модуль (***unit***), завжди зберігається з розширенням ***.pas***; другий - головний файл проекту, що "володіє" програмою – із розширенням ***.dpr***. Файл модуля і файл проекту обов'язково повинні мати *різні імена.*

Розглянемо ще деякі можливості *Інспектора Об'єктів* і *Дизайнера Форм.*

1 Створіть новий проект. Розташуйте на *Формі* об'єкт *TMemo*, а потім *TEdit* так, щоб він наполовину перекривав *TMemo*, як показано на рисунку 12.

2 Виберіть пункт меню *Edit* → *Send to Back*, що приведе до переміщення *TEdit* усередину форми, за об'єкт *TMemo*. Це називається *зміною Z-порядку компонент*. Буква *Z* використовується тому, що так звичайно математики позначають третій вимір – глибину (букви *X* і *Y* використовуються для позначення ширини і висоти).

> ***Порада.*** Якщо Ви "втратили" на формі якийсь об'єкт, то знайти його можна в списку *Combobox*, що розміщений у верхній частині *Інспектора Об'єктів*.

3 Розташуйте кнопку *TButton* у нижній частині форми. Змініть розміри *Інспектора Об'єктів* так, щоб властивості *Name* і *Caption* було видно одночасно на екрані.

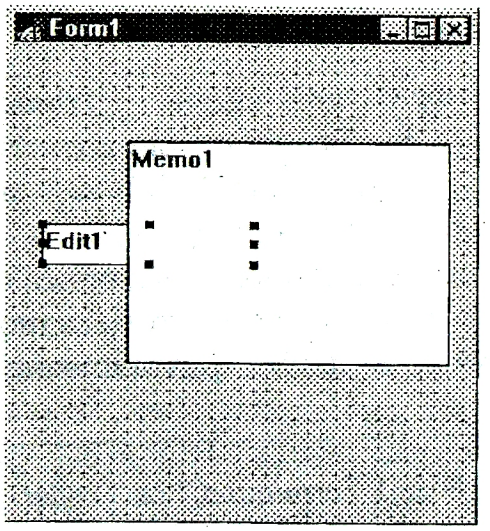

Рисунок 12 - Об'єкт *TEdit* перекривається об'єктом *TMemo*

4 Змініть ім'я (*Name*) кнопки на *Terminate*. Зверніть увагу на те, що заголовок (*Caption*) відразу змінився.

> ***Застереження.*** Така подвійна зміна спостерігається, тільки якщо раніше не змінювалася властивість *Caption*.

Текст, який розташовано на поверхні кнопки, - це вміст властивості *Caption*. Властивість *Name* застосовується для внутрішніх посилань і використовується при написанні коду програми. Якщо в даний момент відкриєте вікно *Редактора*, то побачите наступний фрагмент коду:

```
TForm1 = class(TForm)
        Edit1: TEdit;
        Memo1: TMemo;
        Terminate: TButton;
        private
            { Private declarations }
        public
            { Public declarations }
end;
```

У цьому фрагменті кнопка *TButton* називається *Terminate* через те, що Ви присвоїли цю назву властивості *Name*. Зверніть увагу, що *TMemo* має ім'я, яке присвоюється за замовчуванням.

5 Перейдіть на форму і двічі натисніть клавішею миші на об'єкт *TButton*. Вікно *Форми* автоматично заміниться на вікно *Редактора*, в якому розташований наступний фрагмент коду:

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

Даний код був створений автоматично і буде виконуватися кожного разу, коли під час роботи програми користувач натисне кнопку *Terminate*. Зверніть увагу

також на те, що визначення класу на початку файлу тепер містить посилання на метод *TerminateClick*:

```
TForm1 = class(TForm)
  Edit1: TEdit;
  Memo1: TMemo;
  Terminate: TButton;
  procedure TerminateClick(Sender: TObject);
  private
    {Private declarations}
  public
    {Public declarations}
end;
```

6 Напишіть код для кнопки *Terminate*.

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

Коли цей код виконується, то головна форма (отже, і весь додаток) закривається. Для перевірки коду запустіть програму на виконання і натисніть кнопку *Terminate*. Якщо все зроблено правильно, програма закриється і Ви повернетеся в режим дизайну.

7 Перейдіть в *Інспектор Об'єктів* і змінить значення властивості *Name* для кнопки на будь-яке інше, наприклад, OK. Натисніть Enter для внесення змін. Подивіться у *Редактор*, код, який був написаний, змінився:

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

Зверніть увагу, що аналогічні зміни відбулися й у визначенні класу:

```
TForm1 = class(TForm)
        Edit1: TEdit;
        Memo1: TMemo;
        Ok: TButton;
        procedure OkClick(Sender: TObject);
        private
                {Private declarations}
        public
                {Public declarations}
end;
```

ГОЛОВНЕ МЕНЮ DELPHI

Пункт меню File

Пункт меню *File* складається з шести секцій:

New Project Open Project Save Project Save Project As Close Project	Секція керування проектом у цілому.
New Form New Unit New Component Open File Save File Save File As Close File	Секція контролю над формами, модулями і компонентами проекту.
Add File Remove File Print	Секція керування файлами проекту та друком.
Exit	Вихід з Delphi.
PREV1.DPR PREV2.DPR	Список проектів, які раніше редагувалися.

Більшість операцій цього пункту можна виконати за допомогою *Менеджера Проекту* (Project Manager), що викликається з пункту меню *View*. Деякі операції доступні і через *SpeedBar*. Дана стратегія типова для Delphi: вона надає кілька шляхів для розв'язання однієї і тієї самої задачі, програміст сам може вирішувати, який з них більш ефективний у даній ситуації.

Більшість з пунктів першої секції очевидні.

New Project починає новий проект, ***Open Project*** відкриває існуючий проект і т.д.

Перші два пункти другої секції дозволяють створити нову форму чи новий модуль. Вибір ***New Form*** приводить до створення нової форми та пов'язаного з нею нового модуля; вибір ***New Unit*** приводить до створення одного модуля.

New Component викликає діалог для побудови шаблону нового візуального компонента. У результаті створюється модуль, який можна скомпілювати та внести в *Палітру Компонентів*.

Open File відкриває за необхідності будь-який модуль чи просто текстовий файл. Якщо модуль описує форму, то ця форма теж з'явиться на екрані.

При створенні нового модуля Delphi дає йому ім'я за замовчуванням. Можна змінити це ім'я на що-небудь більш осмислене (наприклад, *main.pas*) за допомогою пункту ***Save File As***.

Save File зберігає тільки той файл, що редагується, але не весь проект.

Close File видаляє файл із вікна *Редактора*.

Пункт меню Edit

Пункт *Edit* містить команди *Undo* і *Redo*, що можуть бути дуже корисними при роботі у *Редакторі* для усунення наслідків при неправильних діях, наприклад, якщо випадково вилучений потрібний фрагмент тексту.

Слід зазначити, що *Довідник* (on-line help) пояснює, як потрібно використовувати пункт меню *Options* → *Environment* для настроювання команди *Undo*. Можливість обмежити кількість команд *Undo* може знадобитися при роботі на машині з обмеженими ресурсами.

Команди *Bring To Front*, *Send To Back*, *Align* і *Size* розглядалися вище. Чотири пункти, що залишилися,

допомагають швидко "прикрасити" зовнішній вигляд форми.

Пункт меню Menu

У пункті *Search* є команда *Find Error* (пошук помилки), за допомогою якої можна відстежити помилку періоду виконання програми. Коли в повідомленні про помилку зазначена її адреса, можна вибрати пункт меню Search → Find Error і ввести цю адресу. Якщо це можливо, то середовище перемістить курсор у те місце програми, де відбулася помилка.

Пункт меню View

Складові пункту меню "*View*":

- ***Project Manager*** (Менеджер Проекту).
- ***Project Source*** - завантажує головний файл проекту (*dpr*) у *Редактор*.
- Установка, показувати чи ні *Інспектор Об'єктів* (***Object Inspector)*** на екрані.
- Установка, показувати чи ні діалог ***Alignment Palette***. (це ж доступно із пункту меню *Edit* → *Align*).
- ***Browser*** - виклик засобу для перегляду ієрархії об'єктів програми, пошуку ідентифікатора у вихідних текстах і т.п.
- ***Watch, Breakpoint*** і ***Call Stack*** - пов'язані з процедурою налагодження програми.
- ***Component List*** - список компонентів, альтернатива *Палітрі Компонентів*. Використовується для пошуку компонента по імені або за відсутності миші.
- ***Window List*** - список вікон, відкритих у середовищі *Delphi*.

- ***Toggle Form/Unit, Units, Forms*** - переключення між формою і відповідним модулем, вибір модуля чи форми із списку.
- ***New Edit Window*** - відкриває додаткове вікно *Редактора*. Корисно, наприклад, за необхідності перегляду двох різних версій одного файлу.
- ***SpeedBar*** і ***Component Palette*** – установки, які вказують, чи потрібно відображати відповідні об'єкти *SpeedBar* і *Component Palette* .

Пункт меню Compile

У пункті меню *Compile* можна скомпілювати (*compile*) чи перебудувати (*build*) проект. Якщо вибрати *Compile* чи *Run*, то Delphi перекомпілює тільки ті модулі, що змінилися з часу останньої компіляції. *Build all* перекомпілює всі модулі, вихідні тексти яких доступні. Команда *Syntax Check* перевіряє правильність коду програми, але не обновлює *.dcu* файли.

Пункт *Information* видає інформацію про програму: розміри сегментів коду, даних і стека, розмір локальної динамічної пам'яті і кількість скомпільованих рядків.

Пункт меню Run

Пункт *Run* можна використовувати для компіляції і запуску програми, а також при вказівці параметрів командного рядка для передачі в програму. Тут же розташовані опції для режиму налагодження.

Пункт меню Options → Project

Options - найбільш складна частина системного меню. Це центр керування, з якого можна змінювати установки для проекту і для всього робочого середовища Delphi. Пункт складається із семи пунктів:

Project - вибір установок, що прямо впливають на поточний проект, це можуть бути, приміром, директиви компілятора перевірки стека (stack checking) чи діапазону (range checking).

Environment - конфігурація самого середовища програмування (IDE). Наприклад, тут можна змінити кольори, які використовуються в *Редакторі*.

Tools - дозволяє додати (видалити) виклик зовнішніх програм у пункт головного меню *Tools*. Наприклад, при частому використанні якого-небудь редактора чи налагоджувача саме тут його виклик можна додати в меню.

Gallery - дозволяє визначити специфічні установки для *Експерта Форм* і *Експерта Проектів* і їх шаблонів. Експерти і шаблони надають можливість для прискорення конструювання інтерфейсом програми.

Open Library
Install Components
Rebuild Library
} дозволяють сконфігурувати *Палітру Компонентів*.

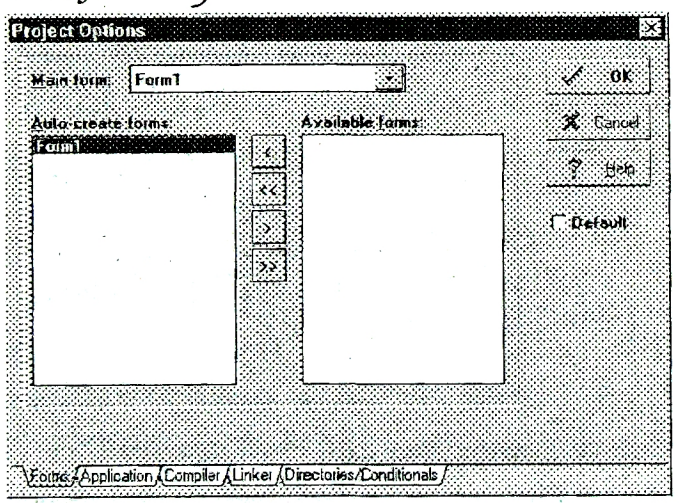

Рисунок 13 - Сторінка Forms

Діалог з пункту ***Options → Project*** містить п'ять сторінок:

- На сторінці ***Forms*** (рис. 13) перелічені всі форми, внесені в проект; можна вибрати головну форму проекту;

можна вказати чи потрібно автоматично створювати форму при старті програми, чи користувач побудує її самостійно.

Зміни, які зробить програміст, відобразяться у відповідному файлі *dpr*. Наприклад, у нижченаведеному проекті, Form1 є головною, оскільки з'являється першою в головному блоці програми:

```
program Project1;
uses
  Forms,
  Unit1 in 'UNIT1.PAS' {Form1},
  Unit2 in 'UNIT2.PAS' {Form2};
{$R *.RES}
begin
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.Run;
end.
```

Якщо змінити код так, щоб він читався так:

```
begin
        Application.CreateForm(TForm2, Form2);
        Application.CreateForm(TForm1, Form1);
        Application.Run;
end.
```

то Form2 стане головною формою проекту.

Також можна використовувати цю сторінку для визначення того, чи буде дана форма створюватися автоматично при старті програми. Якщо форма створюється не автоматично, а по ходу виконання програми, то для цього потрібно використовувати процедуру ***Create***.

> ***Порада***. У секції ***Uses*** ім'я форми у фігурних дужках є істотним для *Менеджера Проектів* і видаляти його не варто. Не потрібно взагалі нічого змінювати вручну у файлі проекту, якщо тільки Ви не захотіли створити DLL, але про це пізніше.

- На сторінці ***Applications*** (рис.14) можна задати заголовок (*Title*), файл допомоги (*Help file*) і піктограму (*Icon*) для проекту.

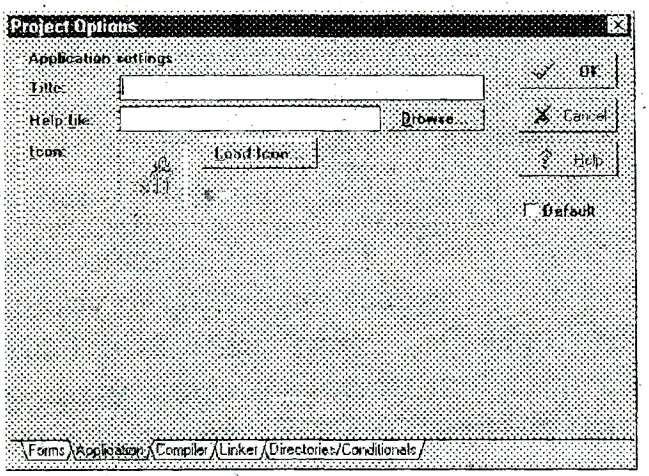

Рисунок 14 - Сторінка загальних установок додатка

- Сторінка ***Compiler*** містить установки для генерації коду, керування обробкою помилок часу виконання, синтаксису, налагодження та інші.

Раніше вже говорилося про те, що установки з пункту меню "*Options → Project*" зберігаються у відповідному файлі з розширенням ***.opt***. Розглянемо директиви компілятора на сторінці *Compiler* (рис.15).

Таблиця А.1 показує, як різні директиви відображаються в *.opt* файлі, на сторінці *Compiler* і усередині коду програми.

На сторінці ***Linker*** (рис. 16) можна визначити умови для процесу лінковки додатка. Слід зазначити, якщо буфер

лінковщика розташований у пам'яті, то лінковка відбувається швидше.

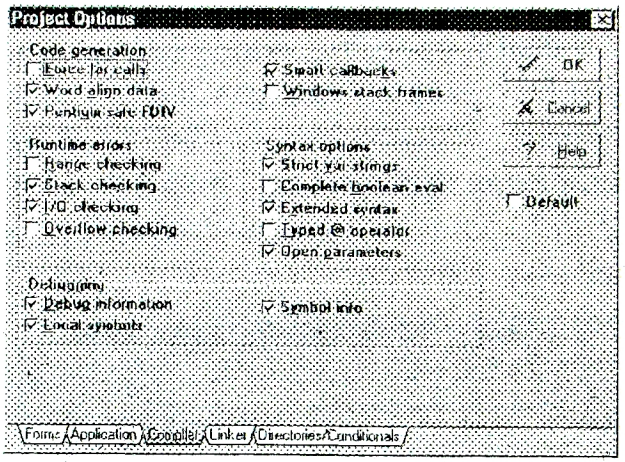

Рисунок 15- Сторінка визначення директив компілятора

Розміри стека (*Stack Size*) і локальної динамічної пам'яті (*Heap Size*) дуже важливі. *Delphi* установлює за замовчуванням і *Stack Size*, і *Heap Size* у 8192 байт кожний. Може знадобитися змінити розмір стека у програмі, але звичайно це не більше 32Кб. У сумі ці два розміри не повинні перевищувати 64Кб, інакше буде видаватися помилка при компіляції програми.

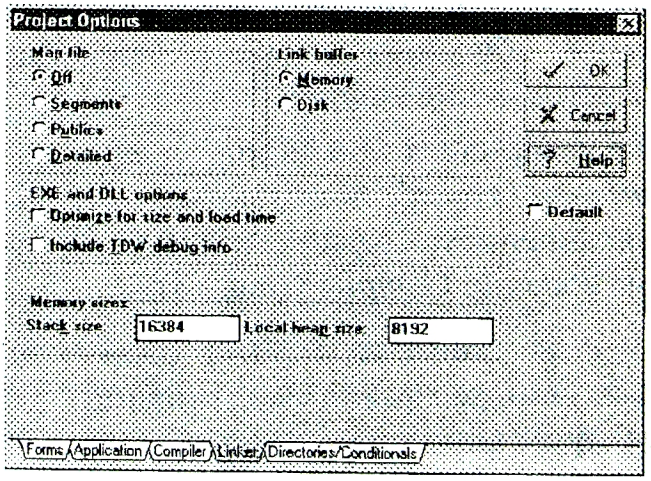

Рисунок 16 - Сторінка *Linker*

- Сторінка ***Directories/Conditionals*** (рис.17) надає можливість розширити кількість директорій, у яких компілятор і лінковщик виконують пошук *.dcu* файлів.

Варто пам'ятати, що існує два види файлів, що містять список директорій:

- *.delphi.ini* – список директорій, що входять до будь-якого проекту;
- у файлах **.opt* – список директорій, що входять до конкретного проекту.

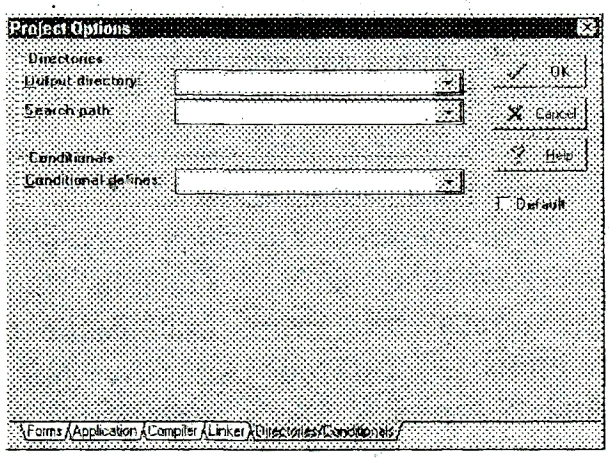

Рисунок 17 - Сторінка *Directories→Conditionals*

Output directory - вихідна директорія, куди складаються *.exe* і *.dcu* файли, отримувані при компіляції.

Search path - список директорій для пошуку *.dcu* файлів при лінковці (директорії перелічуються через крапку з комою).

Опція ***Conditional defines*** використовується досвідченими програмістами і на перших етапах роботи з середовищем Delphi не використовується. Для інформації можна викликати Довідник (on-line help).

> ***Зауваження.*** Всі настройки для проекту, які здійснені в меню Options/Project, зберігаються у текстовому файлі з розширенням ***.opt***, і надалі можливе ручне виправлення цих настройок.

Конфігурація середовища програмування (IDE)

Пункт меню *"Options → Environment"* надає великий набір сторінок і керуючих елементів, що визначають зовнішній вигляд і роботу IDE. Delphi дозволяє зробити такі важливі настроювання:

1 Визначити, які дані з проекту будуть зберігатися автоматично.
2 Змінити кольори IDE.
3 Змінити підсвічування синтаксису в *Редакторі*.
4 Змінити склад *Палітри Компонентів*.
5 Вказати "гарячі клавіші" IDE.

Перша сторінка пункту меню *Options/Environment - **Preferences*** показана на рис.19.

У групі ***Desktop Contents*** визначається, які дані будуть зберігатися при виході з Delphi. При виборі *Desktop Only* відбувається збереження інформації про директорії і відкриті вікна, Desktop And Symbols - забезпечує збереження інформації про директорії і відкриті вікна, а також інформацію для браузера.

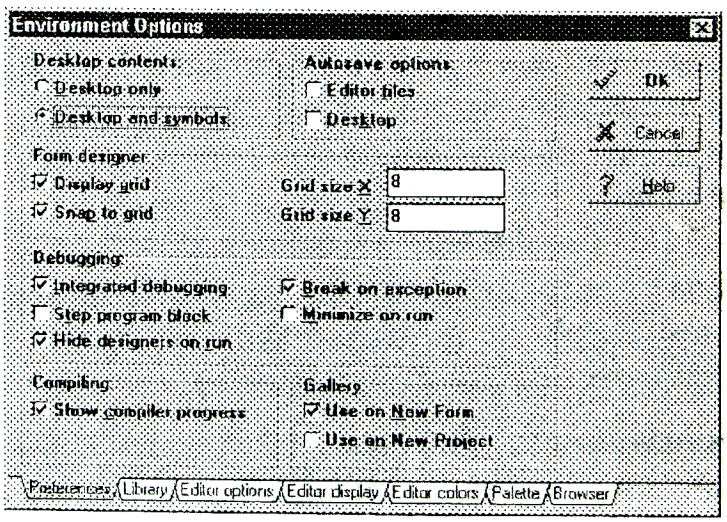

Рисунок 19 - Сторінка *Preferences*

У групі ***Autosave*** вказується, що потрібно зберігати при запуску програми. Якщо обрана позиція *Editor Files*, то зберігаються всі модифіковані файли з *Редактора* при виконанні команд *Run→Run*, *Run→Trace Into*, *Run→Step Over*, *Run→Run To Cursor* чи при виході з Delphi. Обрана позиція *Desktop* забезпечує зберігання робочого середовища при закритті проекту або при виході з Delphi. Якщо проект відкрити пізніше, то він матиме той же вигляд, що і при його закритті.

У групі ***Form Designer*** можна вказати, чи показувати сітку (*grid*) на екрані і чи вирівнювати об'єкти по ній, а також встановити розмір осередків сітки.

У групі ***Debugging*** встановлюється чи використовувати вбудований налагоджувач (*Integrated Debugging*); зупинятися налагоджувачу на першому рядку модуля, у якому є налагоджувальна інформація, чи ні *(Step Program Block)*; чи зупиняти програму при виникненні виняткової ситуації (*Break On Exception)*; чи згортати Delphi при запуску програми *(Minimize On Run)*, після закриття програми середовище Delphi відновлюється; чи ховати вікна *Дизайнера Форм*, *Інспектора Об'єктів* та інші при запуску додатка *(Hide Designers On Run)*.

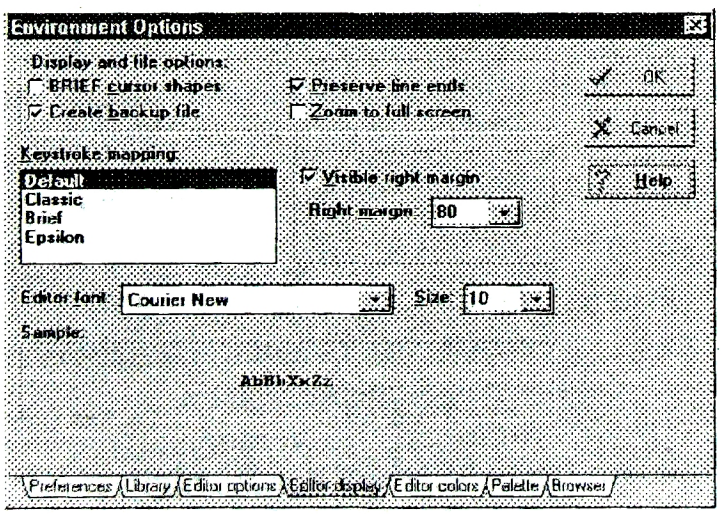

Рисунок 20 - Сторінка *Editor Display*

Show Compiler Progress – вказує, чи показувати вікно, у якому відображується процес компіляції програми.

Gallery - вказує, в яких випадках потрібно надавати "галерею" (колекцію шаблонів і експертів).

Сторінки ***Editor Options, Editor Display*** і ***Editor Colors*** дозволяють змінити кольори та "гарячі" клавіші, які використовуються IDE. Сторінка ***Editor Display*** показана на рисунку 20, а ***Editor Colors*** - на рисунку 21.

Існує кілька способів змінити призначення "гарячих" клавіш, використовуваних *Редактором*. Наприклад, багато користувачів звикли, що по клавіші F5 максимізується вікно *Редактора*. Для цього їм треба використовувати розташування клавіш, назване *Classic* (Keystroke mapping: Classic).

Існує чотири види конфігурації клавіш. Два найбільш поширених:

- *Default* - характерно для Microsoft. Часто використовується новачками чи користувачами, які звикли до цього розташування клавіш.
- *Classic* - більш відома програмістам, які працюють в середовищах Borland C++ і Borland Pascal. Підтримує багато комбінацій клавіш WordStar.

Інші два види - імітують редактори Epsilon і Brief.

Точний опис призначення клавіш можна знайти у *Довіднику* (у *Help* → *Topic Search* набрати "key mapping").

Кольори IDE можна змінити на сторінці *Editor Colors* (рис.21).

І, нарешті, *Editor Options* (рис.22). Багато з установок даної сторінки для більшості користувачів не важливі, тому зупинимося лише на деяких.

Use syntax highlight – установка, яка вказує, виділяти кольором синтаксичні конструкції у *Редакторі* чи ні.

Find text at cursor - якщо опція активізована, то при пошуку (Ctrl+F) за підстроку для пошуку буде братися те слово, під яким розташований курсор.

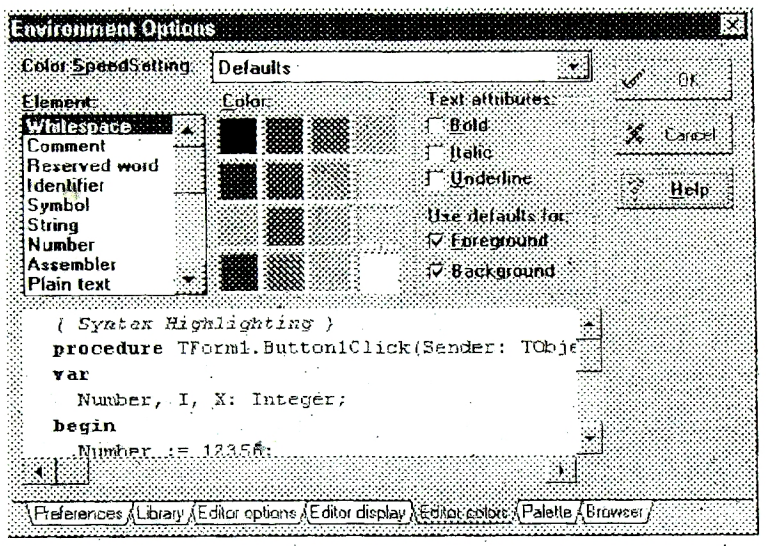

Рисунок 21- Сторінка *Editor Colors*

Про всі опції можна докладніше довідатися у *Довіднику.*

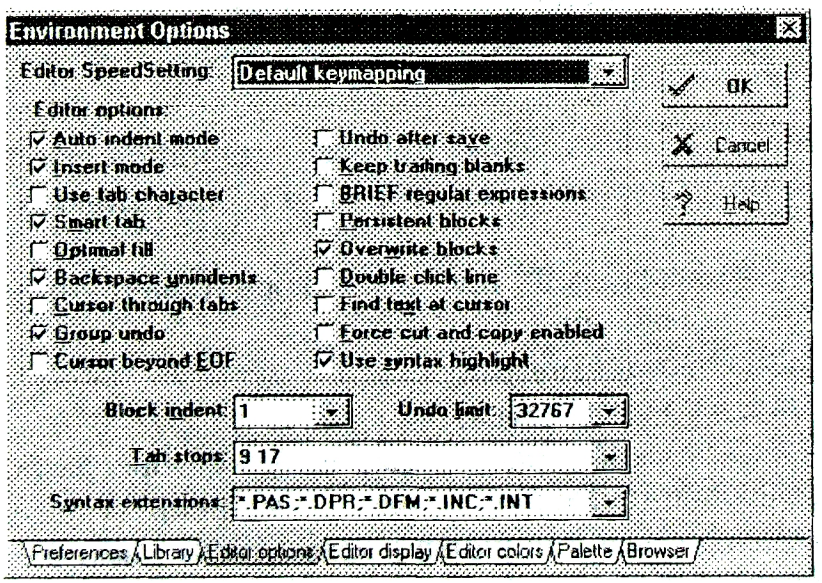

Рисунок 22 - Сторінка *Editor Options*

Всі вищенаведені установки зберігаються у файлі ***delphi.ini***, що міститься у директорії Windows.

7 Проект Delphi

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

1 Головний файл проекту споконвічно називається *project1.dpr*.

2 Перший модуль програми (unit), який автоматично з'являється на початку роботи, за замовчуванням цей файл називається *unit1.pas*, але його можна назвати будь-яким іншим ім'ям, наприклад, *main.pas*.

3 Файл головної форми за замовчуванням - *unit1.dfm*, використовується для збереження інформації про зовнішній вигляд головної форми.

4 Файл *project1.res* містить іконку для проекту, створюється автоматично.

5 Файл за замовчуванням - *project1.opt*, є текстовим файлом для збереження установок, пов'язаних з даним проектом. Наприклад, тут зберігаються директиви компілятора, встановлені користувачем.

6 Файл *project1.dsk* містить інформацію про стан робочого простору.

Зрозуміло, якщо зберегти проект під іншим ім'ям, то автоматично зміняться назви файлів із розширеннями *res*, *opt* і *dsk*.

Після компіляції програми користувач отримує файли з розширеннями:

dcu - скомпільовані модулі;

exe - файл, що виконується;

dsm - службовий файл для запуску програми у середовищі Delphi, має великий розмір, рекомендується знищити його після закінчення роботи;

~pa, ~dp – backup-файли *Редактора*.

8 Керування проектом

Тепер, коли відомо про створення проекту за допомогою пункту меню *File*, перейдемо до *Менеджера Проектів*, що допомагає керувати проектом. *Менеджер Проектів* (рис.23) поділений на дві частини. Верхня - панель з керуючими кнопками. Нижня - список модулів, що входять у проект.

Можна використовувати кнопки з плюсом (***Add***) і мінусом (***Remove***) для додавання і видалення файлів у проекті. Ці зміни впливають на файли з вихідним текстом, тобто, якщо додати в проект модуль, то посилання на нього з'явиться у файлі з розширенням *dpr*.

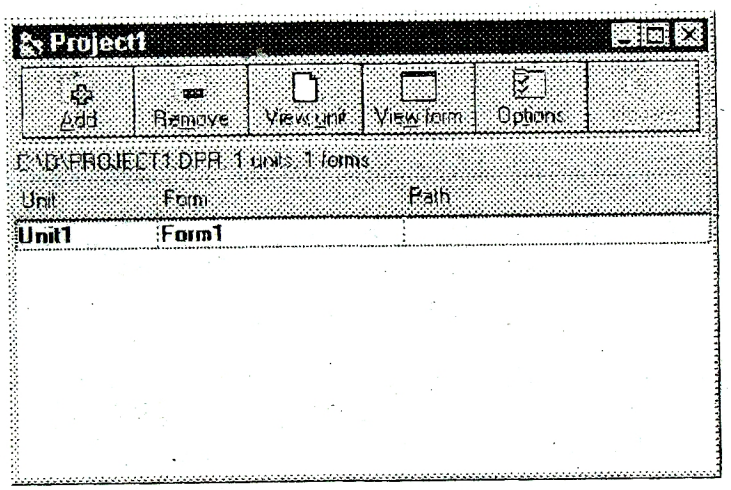

Рисунок 23 - Менеджер Проектів

Короткий опис інших кнопок :

- третя ліворуч кнопка(***View unit***) - перегляд тексту модуля, на якому стоїть курсор;
- четверта (***View form***)- перегляд форми, якщо така є для даного модуля;
- п'ята (***Options***)- виклик діалогу настроювання проекту, сам діалог буде розглянутий пізніше;
- шоста (***Update***) - збереження змін на диску.

9 Створення простого проекту Delphi

Задача 1 Створити проект для перерахування ваги з фунтів у кілограми. Вихідні дані - значення ваги у фунтах - повинні задаватися користувачем.

У Windows дані з клавіатури вводяться в поля редагування. Тому у форму треба додати компонент - *поле редагування.*

Для того щоб додати будь-який компонент у форму, необхідно в *Палітрі Компонентів* (рис. 24) на відповідній вкладці вибрати піктограму необхідного компонента, а потім за допомогою миші вказати те місце на *Формі*, де повинен знаходитися правий верхній кут компонента. В результаті на формі з'явиться компонент стандартного розміру.

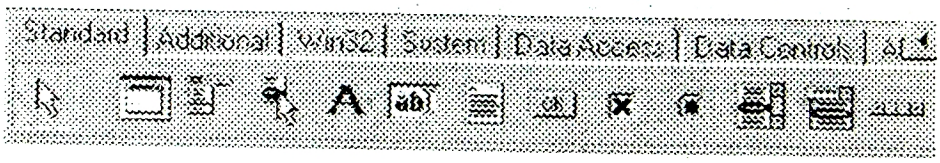

Рисунок 24 - Вкладка *Standard*

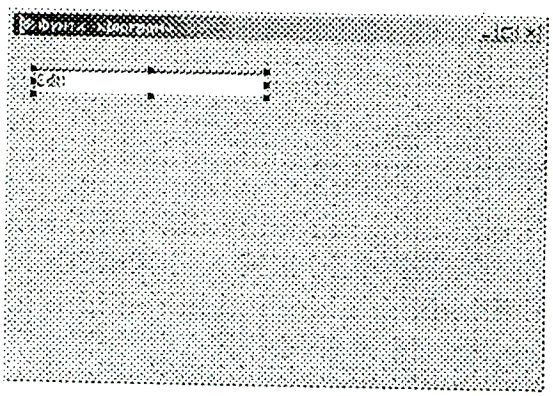

Рисунок 25

На рис. 25 подано вигляд форми після додавання в неї *поля редагування* (***Edit1***).

Компонент форми, оточений вісьмома маркерами, називається *виділеним або марко-ваним.*

Властивості *виділеного* компонента відображаються в діалоговому вікні *Object Inspector* (Інспектор об'єктів).

Delphi надає можливість змінити розмір компонента і його положення на поверхні *Форми.*

Властивості компонента так само, як і властивості форми, можна змінити за допомогою *Інспектора Об'єктів*. Для того щоб властивості необхідного компонента відображалися в діалоговому вікні *Інспектора об'єктів* (***Object Inspector***), необхідно виділити компонент чи вибрати його ім'я у списку, що розкривається, у верхній частині цього вікна (безпосередньо під заголовком).

Таблиця 2

Властивість (Name)	Значення
Text	
Top	48
Left	24
Height	24
Width	121

У табл. 2 наведені значення властивостей поля редагування, призначеного для введення ваги у фунтах.

Крім полів редагування, вікно форми повинне містити пояснюючий текст (коротке інформаційне повідомлення).

Текст, що знаходиться у формі, називають ***міткою***. Мітка додається у форму точно так само, як і поле редагування. Позначка компонента «мітка» () знаходиться на вкладці *Standard* (Стандартна) *Палітри Компонентів* (рис. 24). Після того як мітка додана у форму, можна змінити її властивості в діалоговому вікні *Інспектора Об'єктів*.

У форму розроблюваного додатка треба внести дві мітки. Перша мітка буде являти собою інформаційне повідомлення. Друга мітка призначена для виведення результату перерахування ваги з фунтів у кілограми.

Після додавання міток і встановлення значень властивостей (табл.3) форма додатка, який розроблюється, набуває вигляду, який поданий на рис. 26.

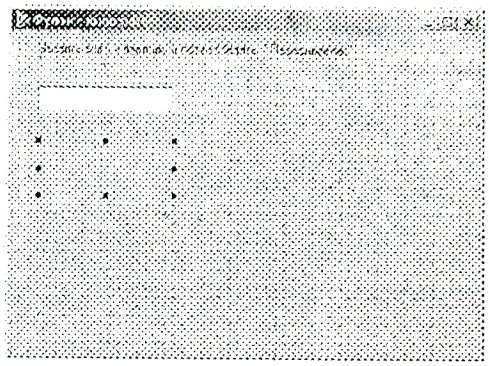

Рисунок 26

Властивість *ParentFont* (Спадкування параметрів шрифту батьківської форми) мітки Label2 має значення false. Тому властивість *Font* цієї мітки не успадковує значення властивості *Font* «батька» - у даному випадку основної форми. Це дає можливість установити інші властивості шрифту мітки, ніж у форми. У мітки *Label1* значення властивості *ParentFont* залишимо без зміни.

Таблиця 3 – Значення властивостей міток Label1 і Label2

Властивість (Name)	**Значення мітки Labe1**	**Значення мітки Label2**
AutoSize	False	False
Caption	Уведіть вагу у фунтах, а потім оберіть “Перерахування”	
Height	33	49
Left	24	24
ParentFont	True	False
Top	8	96
Width	209	121
Wordwrap	True	True

Якщо властивість *AutoSize* (Автоматичний підгін розміру) має значення *true*, то *Delphi* автоматично встановлює розміри мітки в залежності від кількості

символів тексту мітки, використовуваного шрифту і його розміру.

За необхідності розмістити текст мітки в кілька рядків слід властивості *AutoSize* привласнити значення *false* і вручну встановити значення властивостей, що визначають розмір мітки.

На завершення до форми треба додати *командну* кнопку, при натисканні на яку буде виконуватися перерахування ваги у фунтах, уведеної в поле введення, у вагу в кілограмах.

Кнопка додається у форму, як і інші компоненти.

Піктограма командної кнопки () розміщена на вкладці *Standard Палітри Компонентів*.

Після додавання командної кнопки в діалоговому вікні Object Inspector (Інспектор об'єкта) необхідно встановити необхідні значення її властивостей, що подані в табл. 4.

Остаточний вигляд форми розроблюваного додатка перерахування ваги з фунтів у кілограми поданий на рис. 27.

Таблиця 4

Власти-вість	Значення Властивості
Caption	Перерахування
Top	56
Left	184
Height	25
Width	75

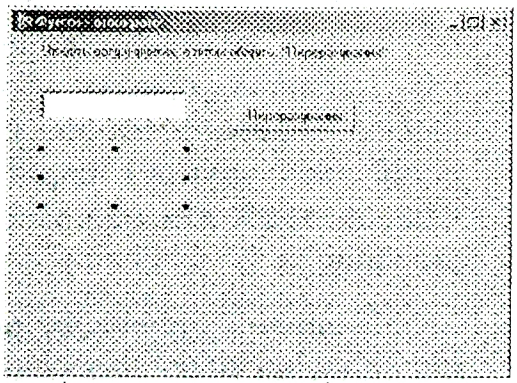

Рисунок 27

9.1 Подія та процедура обробки події

Вигляд створеної форми підказує, як працює додаток. Очевидно, що користувач повинен ввести в поле редагування значення ваги у фунтах і натиснути кнопку "Перерахування". Клацання по зображенню командної кнопки - це приклад того, що в Windows називається *подією*.

Подія - це те, що відбувається під час роботи додатка. У Delphi у кожної події є ім'я. Наприклад, клацання кнопкою миші- це подія OnClick, подвійне клацання мишею - подія OnDblClick.

Реакцією на подію повинна бути яка-небудь дія. Наприклад, реакцією на подію OnClick, що відбулася на кнопці «Перерахування», повинно бути перерахування ваги з фунтів у кілограми. У Delphi реакція на подію реалізується як *процедура обробки події*. Таким чином, завдання програміста полягає в написанні необхідних процедур обробки подій. Методику створення подібних процедур розглянемо на прикладі процедури обробки події для командної кнопки.

Спочатку необхідно виділити об'єкт, для якого створюється процедура обробки події (у прикладі таким об'єктом є командна кнопка «Перерахування»), а потім слід вибрати вкладку *Events* (події) діалогового вікна *Object Inspector*. На вкладці буде відображений список подій, що здатний сприймати *маркований компонент*. Список подій для нашої командної кнопки наведений на рисунку 28.

У лівому стовпчику вкладки *Events* перелічені імена подій, на які може реагувати маркований об'єкт. Якщо для події визначена процедура обробки, то в правому стовпчику поруч з ім'ям події виводиться ім'я цієї процедури.

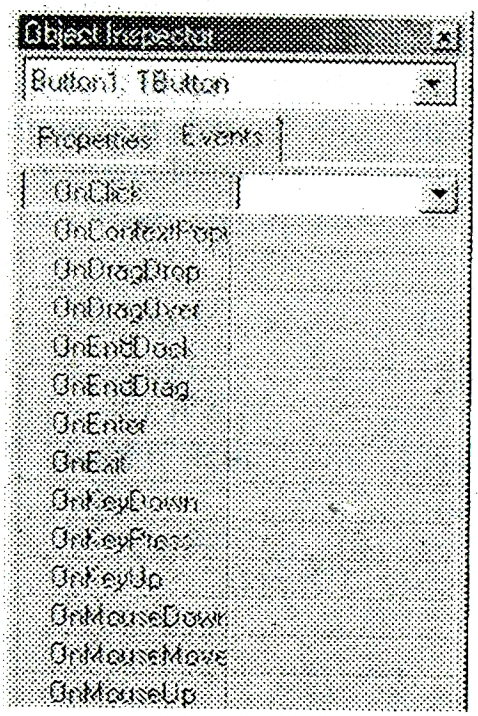

Рисунок 28

Для того щоб створити процедуру обробки події, необхідно виконати подвійне клацання лівою кнопкою миші у *полі імені процедури обробки події* (тобто в правому стовпчику).

У результаті відкриється *вікно редактора коду* Unit1.pas

Кодом називається текст програми із шаблоном процедури обробки події (рис. 29), що створюється автоматично.

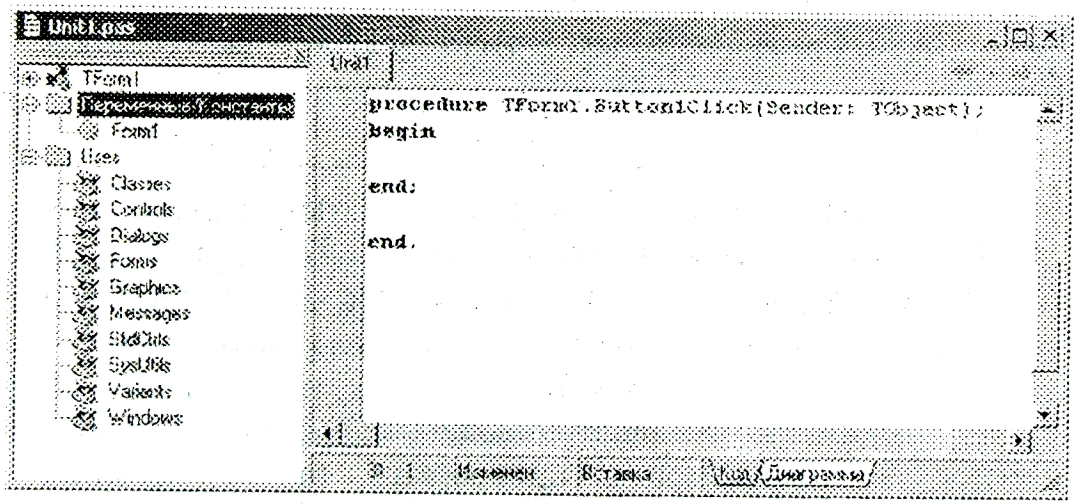

Рисунок 29

Delphi привласнює процедурі обробки події ім'я, що складається з двох частин. Перша частина імені ідентифікує форму, що містить об'єкт, для якого створюється процедура обробки події. Друга частина імені ідентифікує сам об'єкт і подію. У прикладі, який розглядається, ім'я форми - *Form1*, ім'я командної кнопки - *Button1*, а ім'я події Click, саме тому процедура має назву *TForm1.Button1Click*.

У вікні редактора коду між *begin* і *end* можна друкувати інструкції мови *Object Pascal*, що реалізують процедуру обробки події. Нижче наведено текст процедури обробки події *OnClick* для командної кнопки «Перерахування» :

```
procedure TForm1.Button1Click(Sender: TObject);
var
   f:real; //вага у фунтах
   k:real; //вага в кілограмах
begin
   f:=StrToFloat(Edit1.Text); //перевели вміст Edit1 у число
   k:=f*0.4059;
   label2.Caption:=Edit1.text + ' фунт(а/ов) це ' + FloatToStr(k)+
                   ' кілограм';
end;
```

Програма одержує вихідні дані з поля редагування *Edit1* шляхом звернення до властивості *Text*. Властивість *Text* містить рядок символів (текст), тому в програмі для перетворення зображення числа в дійсне число використовується функція *StrToFloat*.

Обчислене значення виводиться програмою в поле мітки *Label2* шляхом присвоєння цього значення властивості *Caption*. Для перетворення дійсного числа в його зображення (текст повідомлення з результатом перерахування) використовується функція *FloatToStr*.

9.2 Контекстно-залежна довідкова система

Редактор коду оснащений контекстно-залежною довідковою системою, що під час набору тексту програми автоматично виводить довідкову інформацію про процедури і функції мови програмування.

Наприклад, якщо під час набору тексту програми надрукувати слово «*StrToFloat*», що є ім'ям функції перетворення рядка символів у дійсне число, і після нього поставити дужку, що відкривається, то на екрані з'явиться маленьке вікно, у якому буде зазначений тип параметра функції (рис. 30).

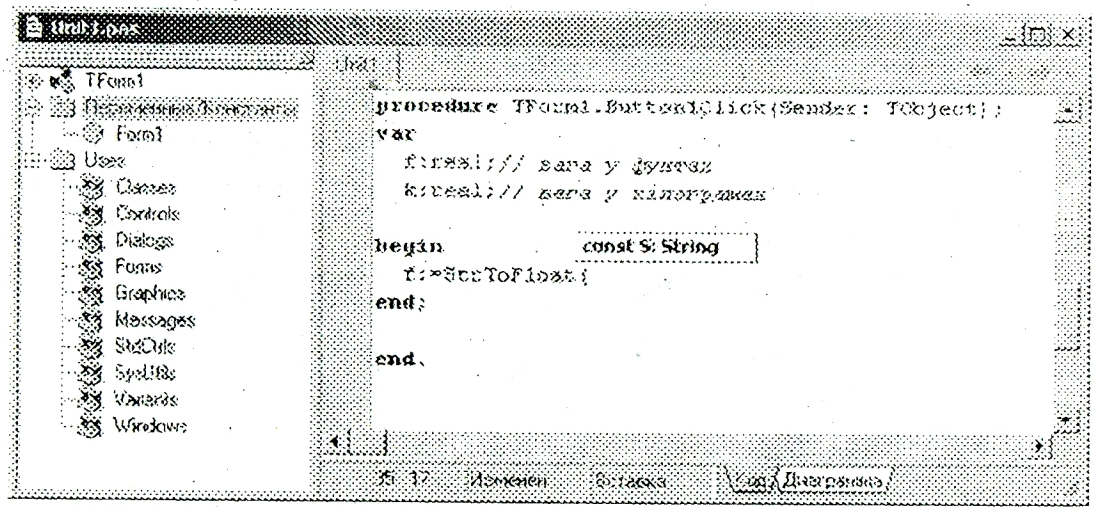

Рисунок 30

При наборі тексту програми можна одержати довідку про конструкцію мови програмування, чи процедуру функції. Для цього необхідно у вікні редактора коду надрукувати слово, про яке треба одержати довідку, потім установити курсор на будь-яку його букву і натиснути функціональну клавішу «*F1*».

Будь-яку допоміжну інформацію, що стосується середовища *Delphi* та мови програмування, можна одержати за допомогою пункта *Help* головного меню *Delphi*.

9.3 Збереження проекту

У термінології Delphi ***проект*** - це набір файлів, використовуючи які компілятор створює файл програми, що виконується. Проект вміщує файл проекту й один чи кілька файлів модулів (*Unit* - модуль).

Файл проекту має розширення *.dpr* і містить загальний опис проекту. Файли модулів проекту мають розширення *.pas* і містять тексти процедур, функцій, опису типів і іншу інформацію, необхідну компілятору для створення програми, що виконується.

Для того щоб зберегти проект, необхідно вибрати команду *Save Project As* (Зберегти проект як) меню *File*. Якщо даний проект зберігається вперше, то у відповідь на команду збереження проекту *Save Project As Delphi* спочатку виводиться діалогове вікно *Save Unit As* (Зберегти модуль як) (рис. 31).

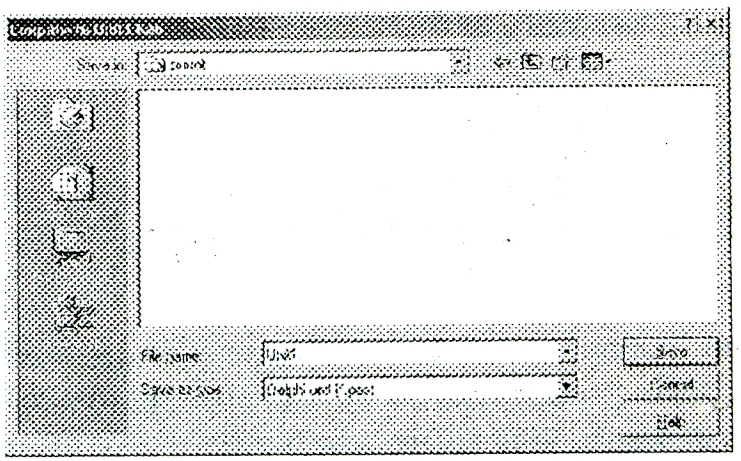

Рисунок 31

У цьому вікні необхідно вибрати папку, призначену для збереження проектів Delphi. У папці проектів Delphi слід створити нову, окрему папку для файлів проекту, що зберігається.

Після збереження файлу модуля проекту відкриється діалогове вікно *Save Project1 As* (Зберегти проект як) (рис.32). У поле «*File name*» (Ім'я файлу) варто ввести ім'я файлу проекту.

Треба зауважити, що ***не можна давати файлу модуля проекту і файлу проекту однакові імена***. Хоча з погляду операційної системи це різні файли (оскільки мають різні розширення), *Delphi* сприйме це як помилку.

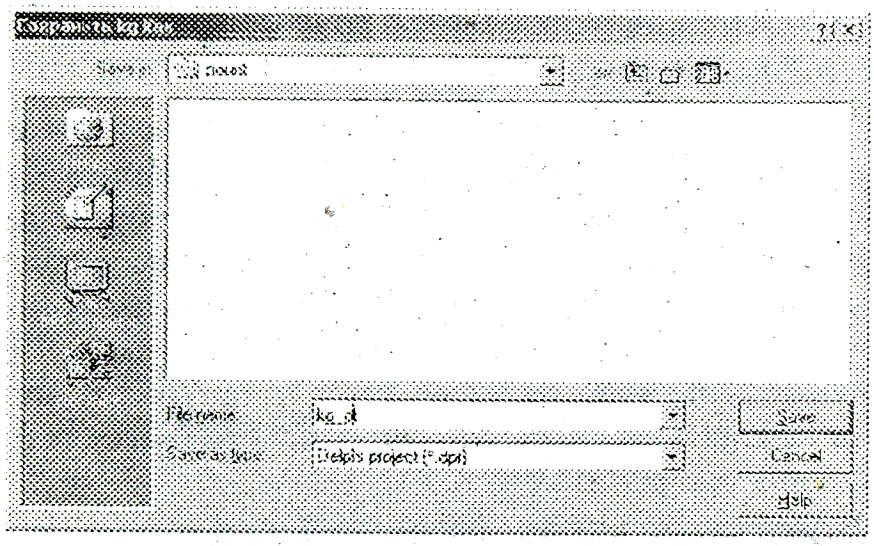

Рисунок 32

9.4 Компіляція та помилки часу компіляції

Після написання коду і збереження проекту можна, вибравши команду *Compile* (Компілювати) у меню *Project* (Проект), відкомпілювати створений додаток. Якщо в програмі немає синтаксичних помилок, то компіляція відбудеться успішно й у результаті буде створений файл, що виконується, *(.exe)* з таким же ім'ям, як і ім'я вашого проекту.

Можна налагодити Delphi так, щоб по завершенні компіляції виводилося діалогове вікно з результатами компіляції. Для цього необхідно вибрати команду *Enviroment Options* (Настроювання) меню *Tools*

(Інструменти/Сервіс). У діалоговому вікні, що *відкрилося*, *Enviroment* Options на *вкладці* Preferences (Установки) варто установити *прапорець Show compiler* progress (Показати хід виконання компіляції), що знаходиться в *групі Compiling and* Running, а потім натиснути кнопку *OK*.

Під час компіляції текст програми перевіряється на відсутність синтаксичних помилок. Компілятор переглядає програму від початку. Якщо виявляється помилка, то процес компіляції припиняється, і у вікні редактора коду виділяється рядок, що містить помилку (рис. 33).

У нижню частину вікна редактора коду компілятор виводить повідомлення про помилки. Якщо розмір вікна редактора коду недостатній для відображення повідомлення про помилку в повному обсязі, то наприкінці повідомлення з'являються крапки. У цьому випадку можна розгорнути вікно редактора коду на весь екран чи помістити курсор миші на слово Error, у результаті чого з'явиться спливаюча підказка, що містить весь текст повідомлення про помилку.

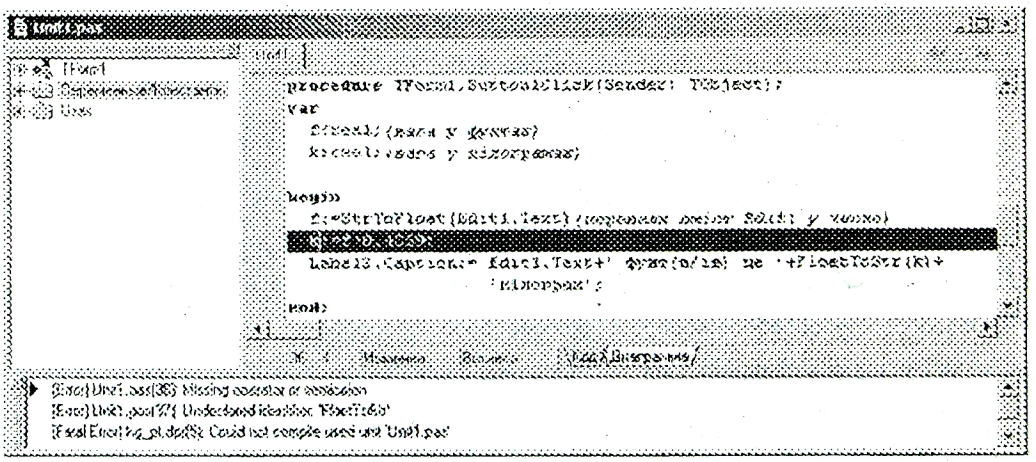

Рисунок 33

Наявність у тексті хоча б однієї синтаксичної помилки, призводить до виникнення другої помилки -

Fatal Error, що говорить про неможливість генерації програми, яка виконується.

Рядок, який виділений компілятором, не завжди містить помилку. Часто помилковою є інструкція, що знаходиться на попередньому рядку. Наприклад, на рисунку 33 інструкція *k:=f*0.4059* правильна, хоча і позначена компілятором як помилкова. Але помилку містить попередній рядок, у якій після інструкції немає символу «крапка з комою».

9.5 Запуск додатка із середовища програмування

Тестовий запуск додатка можна виконати безпосередньо із середовища програмування. Для цього необхідно вибрати команду *Run* меню *Run* (Виконання).

На рис. 34 наведено вид діалогового вікна працюючого додатка *Фунти-Кілограми* після введення ваги у фунтах і натискання кнопки Перерахування.

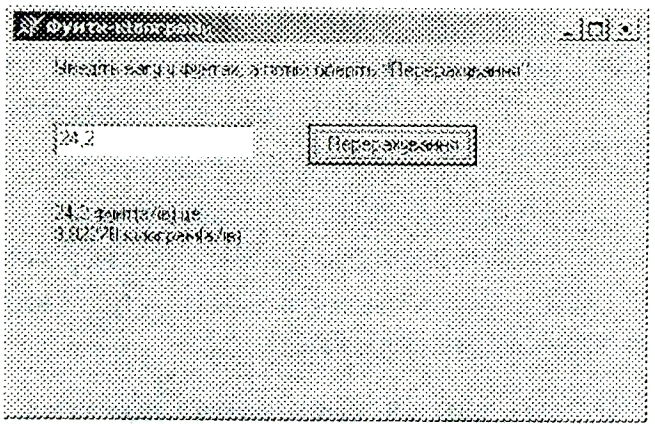

Рисунок 34

9.6 Помилки часу виконання додатка

Під час роботи додатка можуть виникати помилки, що називаються *помилками часу виконання* чи *виключеннями* (exceptions). У більшості випадків причинами *виключень* є неправильні вихідні дані.

Наприклад, якщо в додатку *Фунти-Кілограми* число фунтів увести не цифрами, а словом, наприклад, «півтора», то після натискання кнопки «Перерахування» на екран буде виведене вікно з повідомленням про помилку.

У тексті програми дробова частина числової константи відокремлюється від цілої частини крапкою. Але при введенні вихідних даних користувач може відокремити дробову частину числа від цілої крапкою чи комою. Який з цих двох символів є припустимим, залежить від настроювань Windows.

Якщо програма запускається з *Delphi*, то при виникненні *виключення* також з'являється вікно з повідомленням про помилку, і виконання програми припиняється. У цьому вікні, крім повідомлення про помилку, вказується тип помилки.

При розробці проекту програміст повинен передбачити всі можливі варіанти некоректних дій користувача, що можуть призвести до виникнення помилок часу виконання додатка.

9.7 Остаточне настроювання додатка

Після того як програма налагоджена, необхідно виконати її остаточне настроювання: призначити додатку *піктограму*, що буде зображувати файл додатка, що виконується, у папках чи на робочому столі.. Ця позначка буде знаходиться під час роботи додатка поруч з її ім'ям на панелі задач Windows.

Для того щоб призначити додатку позначку, необхідно вибрати команду *Options* меню *Project* (чи натиснути комбінацію клавіш *Shift+Ctrl+F11*) і в діалоговому вікні, що *відкрилося*, вибрати *закладку* Application. (рис. 35)

У поле *Title* (Заголовок) варто ввести назву додатка. Під час роботи додатка його назва виводиться на панелі задач Windows поряд з позначкою, що зображує цей додаток.

Щоб призначити додатку піктограму, необхідно натиснути кнопку *Load Icon* ("Выбрать"). У діалоговому вікні *Application Icon* потрібно знайти файл (з розширенням ***.ico***) потрібної позначки, вибрати його та натиснути кнопку *Open*. У результаті діалогове вікно *Application Icon* закриється, і знову з'явиться діалогове вікно *Project Options*, у якому тепер відображається вибрана позначка додатка (рис.35). Для завершення процесу слід натиснути кнопку *OK*. Власну піктограму можна створити за допомогою команди *Image Editor* пункту *Tools* головного меню.

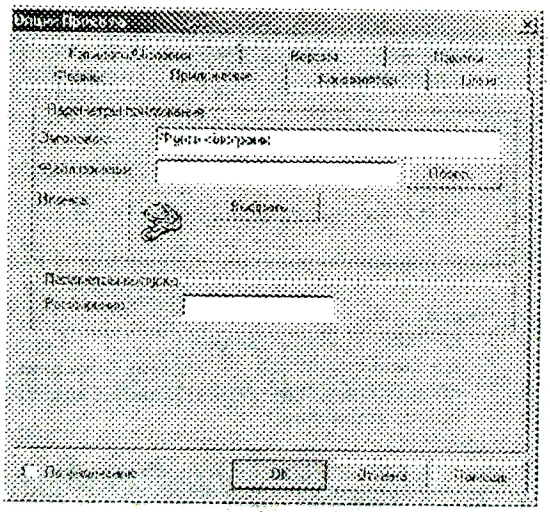

Рисунок 35

9.8 Структура простого проекту Delphi

Проект *Delphi* являє собою набір програмних одиниць - модулів. Один з модулів, названий глобальним, формується *Delphi* автоматично та містить інструкції, з яких починається виконання програми.

> ***Примітка***. Головний модуль зберігається у файлі з розширенням ***.dpr***. Щоб переглянути текст головного модуля додатка, необхідно вибрати команду *View Source* ("Исходный текст") меню *Project*.

На рисунку 36 наведено приклад головного модуля програми *Фунти-Кілограми*.

Головний модуль починається словом *Program*, за яким слідує ім'я програми, що збігається з ім'ям проекту. Ім'я проекту задається програмістом у момент збереження файлу проекту, і воно визначає ім'я створюваного компілятором файлу програми, що виконується. Далі за словом *uses* ідуть імена використовуваних модулів: бібліотечного модуля *Forms* і модуля форми *Unti1.pas*

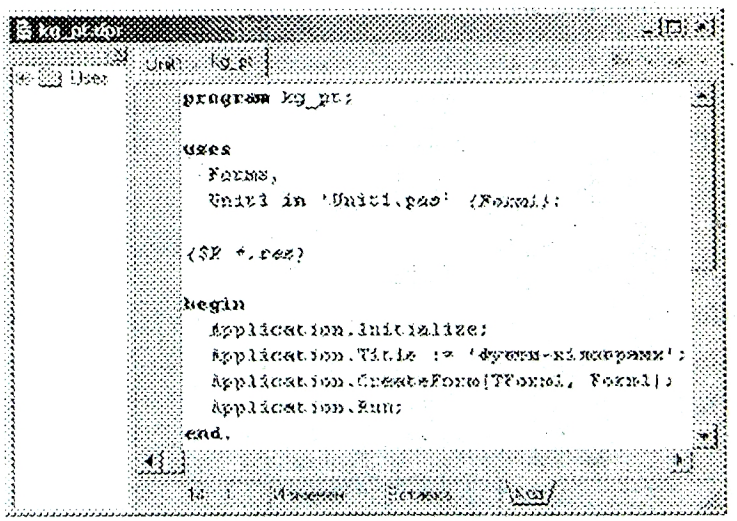

Рисунок 36

Схожа на коментар директива *{$R *.RES}* вказує компілятору на те, що потрібно використовувати файл ресурсів, який містить опис ресурсів додатка, наприклад, піктограми. Зірочка вказує, що ім'я файлу ресурсів таке ж, як і ім'я файлу проекту, але з розширенням ***.res***

> ***Примітка*** Файл ресурсів не є текстовим файлом, тому переглянути його за допомогою текстового редактора не можна.

Частина головного модуля, що виконується, знаходиться між інструкціями *begin* і *end*. Інструкції частини, що виконується, забезпечують ініціалізацію додатка і висновок на екран стартового вікна.

Крім головного модуля, кожна програма вміщує як мінімум один модуль форми, що містить опис стартової форми додатка і підтримує роботу її процедур. У *Delphi* кожній формі відповідає свій модуль.

Нижче наведено повний текст модуля програми перерахування ваги з фунтів у кілограми.

```
unit Unit1;
interface
uses Windows, Messages, SysUtils, Classes, Graphics,
     Controls, Forms, Dialogs, StdCtrls;
type TForm1 = class(TForm)
            Edit1: TEdit;
            Label1: TLabel;
            Label2: TLabel;
            Button1: TButton;
            procedure Button1Click(Sender: TObject);
            private
            { Private declarations }
            public
            { Public declarations }
```

```
    end;
var Form1: TForm1;
  Implementation
  {$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject);
  var
      f:real;//вага у фунтах
      k:real;//вага в кілограмах
  begin
      f:=StrToFloat(Edit1.Text);//перевели вміст Edit1 у
                                          число
      k:=f*0.4059;
      label2.Caption:=Edit1.text+' фунт(а/ів) це '+
                          FloatToStr(k)+' кілограм(а/ів)';
  end;
end.
```

Директива *{$R *.DFM}* вказує компілятору, що до розділу реалізації слід додати інструкції з встановлення значень властивостей форми, що знаходяться у файлі з розширенням ***.dfm***. Файл у форматі DFM генерується *Delphi* на основі зовнішнього вигляду форми. Нижче наведено текст файлу *unit1.dfm* проекту *Фунти-кілограми*. На прикладі цього модуля добре видно, як *Delphi* зберігає властивості об'єктів.

```
object Form1: TForm1
      Left = 191
      Top = 108
      Width = 290
      Height = 185
      Caption = 'Фунти-кілограми'
      Color = clBtnFace
      Font.Charset = DEFAULT_CHARSET
```

```
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = 'MS Sans Serif'
        Font.Style = []
        OldCreateOrder = False
        PixelsPerInch = 96
        TextHeight = 13
    Object Label1: TLabel
        Left = 24
        Top = 16
        Width = 250
        Height = 13
        Caption = 'Уведіть вагу у фунтах, а потім оберіть
       «Перерахування»'
    end
    Object Label2: TLabel
        Left = 24
        Top = 80
        Width = 121
        Height = 73
        WordWrap = True
    End
    Object Edit1: TEdit
        Left = 24
        Top = 48
        Width = 121
        Height = 21
        TabOrder = 0
    end
    Object Button1: TButton
        Left = 160
        Top = 48
```

```
        Width = 75
        Height = 25
        Caption = 'Перерахування'
        TabOrder = 1
        OnClick = Button1Click
    end;
end.
```

10 РОБОТА З ФАЙЛАМИ В DELPHI

10.1 Вивід даних у файл

Досить часто результати виконання тієї чи іншої програми є громіздкими для виведення на екран, що завжди незручно для їх контролю.

У загальному вигляді оголошення файлу таке:

ім'я змінної: ***file of*** *Тип_Елементів* – для типізованих файлів;

ім'я змінної: ***TextFile*** - для текстових файлів.

Оголошення файлової змінної задає тільки тип компонентів файлу. Для того щоб програма могла виводити дані в файл чи зчитувати їх із файлу, необхідно вказати конкретний файл, тобто задати ім'я фізичного файлу.

У *Delphi ім'я файлу* задається за допомогою процедури ***AssignFile***, що зв'язує файлову змінну з конкретним файлом.

Опис процедури AssignFile має такий вигляд:

AssignFile(var f, Ім'я_файлу: string);

Ім'я файлу задається відповідно до прийнятих в Windows правил. Воно може бути повним, тобто складатися не тільки з імені файлу, але і містити шлях до файлу. Наприклад:

```
AssignFile(f,'\students\results.txt');
```

Вивід даних у текстовий файл здійснюється за допомогою відомих процедур ***write*** чи ***writeln***.

Перед виведенням даних у файл його необхідно відкрити. Якщо програма, що формує файл, уже використовувалася, то можливо, що файл із результатами вже існує на диску. Тому програміст повинен вирішити, як використати старий файл: замінити старі дані на нові чи нові дані додати до старого файлу. Спосіб використання старого файлу визначається під час відкриття файлу.

Можливі такі режими відкриття файлу для запису в нього даних:

- Перезапис (запис нового файлу поверх існуючого чи створення нового файлу, якщо його не існує). Для цього використовується процедура ***Rewrite (f)***, де *f* - файлова змінна типу *TextFile*.
- Додавання в існуючий файл (як правило, в кінець файлу, але можна записати дані й у довільне місце). Використовується процедура ***Append (f)***, де *f* - файлова змінна типу *TextFile*.

10.2 Помилки відкриття файлу

Спроба відкрити файл може завершитися невдало і викликати помилку часу виконання. Причин невдачі відкриття файлу може бути декілька. Наприклад, програма може спробувати відкрити файл на гнучкому диску, що не готовий до роботи (не закрита шторка гнучкого диску чи диск нагромаджувача не вставлений у нагромаджувач). Інша причина - відсутність файлу, що відкривається в режимі додавання.

Програма може взяти на себе завдання контролю за результатом виконання інструкцій відкриття файлу. Зробити це можна, перевіривши значення функції ***IOResult*** (Input - Output Result - результат введення-виведення). Функція *IOResult* повертає 0, якщо операція введення-

виведення завершилася успішно, у протилежному разі - не нуль.

Для того щоб програма змогла перевірити результат виконання операції введення-виведення, потрібно помістити відповідну директиву компілятору - рядок ***{$I-}***, що забороняє автоматичну обробку помилок введення-виведення (ця директива повідомляє компілятору, що програма бере на себе контроль за помилками). Після інструкції відкриття файлу слід помістити директиву ***{$I+}***, що відновлює режим автоматичної обробки помилок введення-виведення.

```
AssignFile (f, 'D:\testfile.txt');
{$I-}
 Append (f); {Намагаємося відкрити для додавання}
{$I+}
 If IOResult<>0 {тобто помилка відкриття}
     then Rewrite(f); {створимо новий файл}
```

10.3 Закриття файлу

Перед закінченням роботи програма повинна закрити всі відкриті файли. Це робиться перед викликом процедур ***Close (f)*** чи ***CloseFile(f)***.

Щоб остаточно в усьому розібратися, розглянемо на прикладі, як виконується виведення даних (результат роботи програми) у файл.

Задача 2 Створити проект, який буде вести базу даних, до полів якої користувач зможе вводити дату та відповідну температуру повітря.

На рисунку 37 наведено вигляд форми після додавання ***полів-міток, полів-редагування*** та ***командної кнопки.***

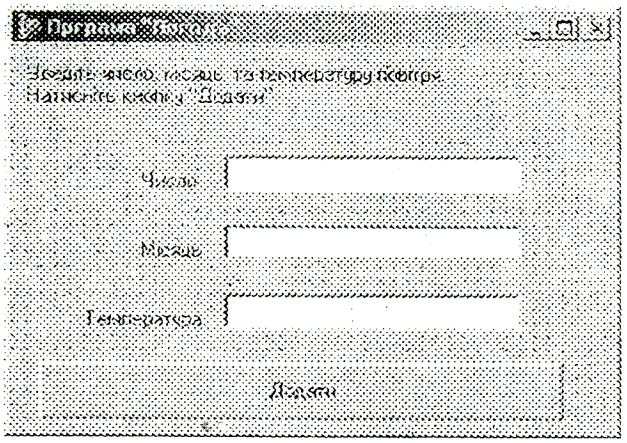

Рисунок 37

Дата вводиться в поля *Edit1* (число - номер дня в місяці) і *Edit2* (номер місяця). Для введення температури призначене поле *Edit3*. При створенні форми програми властивості *Enabled* кнопки «Додати» (*Button1*) треба привласнити значення False.

Текст модуля "Погода"

```
unit Unit1;
interface
uses   Windows, Messages, SysUtils, Variants, Classes,
       Graphics, Controls, Forms,  Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Button1: TButton;
    Label2: TLabel;
    Label3: TLabel;
```

```
    Label4: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action:
                                                TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  f: TextFile;
implementation

{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
     if  (length(Edit1.text) = 0) or
         (length(Edit2.text) = 0) or
         (length(Edit3.text)=0)
     then
       showmessage('Слід заповнити всі поля')
     else
       writeln(f,Edit1.Text,' ',Edit2.Text,' ',Edit3.Text);
end;
procedure TForm1.FormActivate(Sender: TObject);
begin
  AssignFile(f,'c:\weather.txt');
  {$I-}
  Append(f);
  {$I+}
  if IOResult=0
     then
```

```
            button1.Enabled:=true
      else
         begin
            rewrite(f);
            if IOResult=0
                  then
                        Button1.Enabled:=true
                  else
                        showmessage('Помилка створення
      файлу');
         end;
end;
procedure TForm1.FormClose(Sender: TObject; var Action:
                                                TCloseActi
                                                on);

begin

   CloseFile(f);
end;

end.
```

Файл бази даних відкриває процедура ***TForm1.FormAcivate*** (процедура обробки події *OnActivate* об'єкта *TForm1*), що запускається автоматично при активізації форми додатка. Якщо операція відкриття файлу закінчується успішно, то процедура робить доступною кнопку «Додати». У протилежному разі кнопка «Додати» залишається недоступною.

Процедура ***TForm1.Button1Click*** (процедура обробки події *OnClick об'єкта Button1*) запускається натисканням кнопки «Додати». У результаті введена інформація записується в базу даних - файл *weather.txt*. Перед виконанням запису програма перевіряє, чи всі поля

форми заповнені і, якщо не всі, то з'являється інформаційне повідомлення *"Слід заповнити всі поля"*. У результаті роботи процедури в кінець файлу буде доданий рядок, що містить три числа: номер числа, місяця і температуру повітря.

Закриває базу даних процедура ***TForm1.FormClose*** (обробляє подію *OnClose* об'єкта *TForm1*, що виникає при закритті користувачем форми додатка).

11 СТВОРЕННЯ ПРОЕКТУ "РЕДАКТОР"

Задача 3 Створити власний текстовий редактор.

Основним компонентом текстового редактора є поле для введення та редагування тексту. Зробимо вибір між компонентами *Memo* і *RichEdit* на користь компонента *RichEdit* (знаходиться на вкладці *Win32* Палітри *Компонентів*), тому що він підтримує абзаци, нумерацію рядків, роботу зі шрифтами, кольорове виділення тексту та ін.

Крок 1 Розмістимо компонент *RichEdit* на формі так, щоб він займав майже всю робочу поверхню вікна (рис. 38), але треба пам'ятати, що при зміні розмірів вікна користувачем, компонент також повинен змінювати свої розміри. Для цього встановимо властивості *Align* компонента *RichEdit* значення *AlClient*.

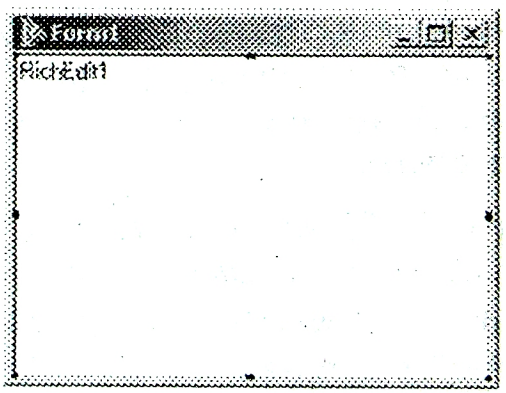

Рисунок 38

При встановленні компонента *RichEdit* в ньому вже міститься один рядок тексту (за замовчуванням це ім'я знову створеного компонента). Текст в *RichEdit* розміщується в рядках, кожен з яких має свій порядковий номер (відлік починається з нуля). За відображення тексту в *RichEdit* відповідає властивість *Lines* типу *Tstrings*, яку можна уявити як масив рядків. Щоб змінити, наприклад, перший рядок, необхідно написати так:

```
RichEdit1.Lines[0]:='Новий рядок';
```

Крок 2 Видалимо текст RichEdit1 з першого рядка компонента *RichEdit*. Це можна зробити двома способами.

Перший спосіб. Помістити рядок RichEdit1.Lines[0]:=' ' в подію форми *OnCreate* (рис. 39).

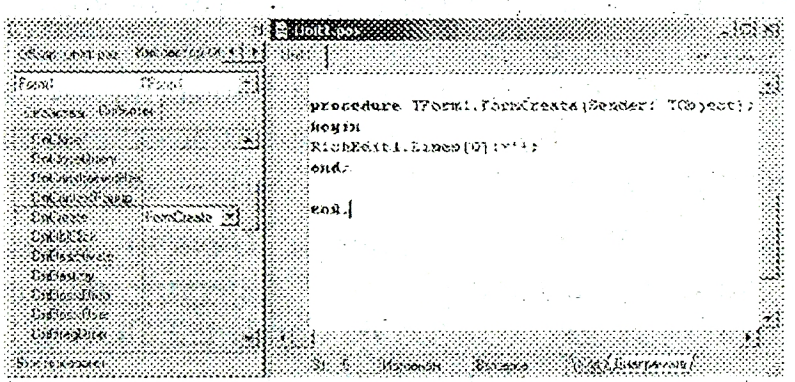

Рисунок 39

Другий спосіб. Виділити компонент RichEdit, в *Інспекторі Об'єктів* обрати властивість *Lines*, а потім натиснути на кнопку з трьома крапками. У редакторі тексту (рис.40) необхідно стерти весь вміст.

Крок 3 Присвоїмо текстовому редактору, який створюється, назву "Редактор" і розмістимо її в заголовок форми. Для цього в *Інспекторі Об'єктів* властивості *Caption* створюваної форми необхідно надати значення "Редактор".

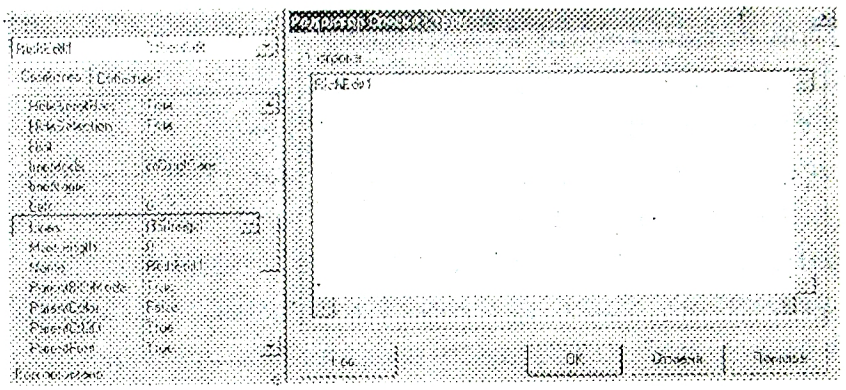

Рисунок 40

Крок 4 Створимо панель інструментів (ToolBar), на якій будуть розміщені кнопки швидкого виклику команд (наприклад, Відкрити, Зберегти і т.ін.).

Для цього слід розмістити на формі компонент *Panel* (вкладка *Standart Палітри Компонентів*). Стерти у неї властивість *Caption*. Встановити для властивості панелі *Align* значення *alTop* (щоб панель завжди розміщувалася у верхній частині вікна).

Крок 5 Розмістимо кнопки (*Botton*) швидкого виклику команд "Відкрити" та "Зберегти" на панелі інструментів, причому *Botton1.Caption=Відкрити*, *Botton2.Caption=Зберегти*.

Щоб відкривати і зберігати текстові файли, знадобляться ще два компоненти - це *OpenDialog* і *SaveDialog* (розміщені на вкладці *Dialogs Палітри Компонентів*). Це *невізуальні* компоненти і їх не буде видно під час роботи додатка, тому розмістити їх можна у будь-яке місце форми.

Результат роботи п'яти кроків зображено на рис.41.

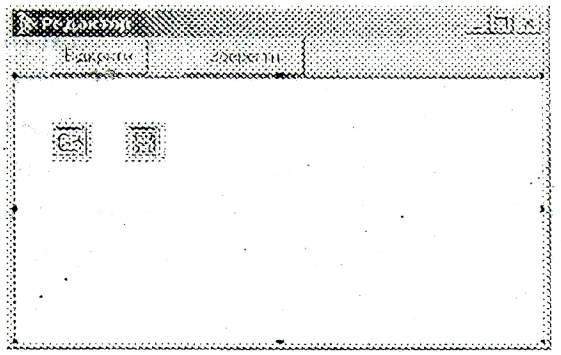

Рисунок 41

Крок 6 Напишемо процедуру відкриття файлу. Для цього необхідно обробити подію ***OnClick*** командної кнопки *Button1* - "Відкрити" (у формі двічі натиснути на кнопку "Відкрити").

```
procedure TForm1.Button1Click(Sender: TObject);
begin
if OpenDialog1.Execute
   then
      RichEdit1.Lines.LoadFromFile(OpenDialog1.Filename);
end;
```

Таким чином, коли користувач під час виконання програми натисне на кнопку "Відкрити", з'явиться діалогове вікно *Open* (рис. 42).

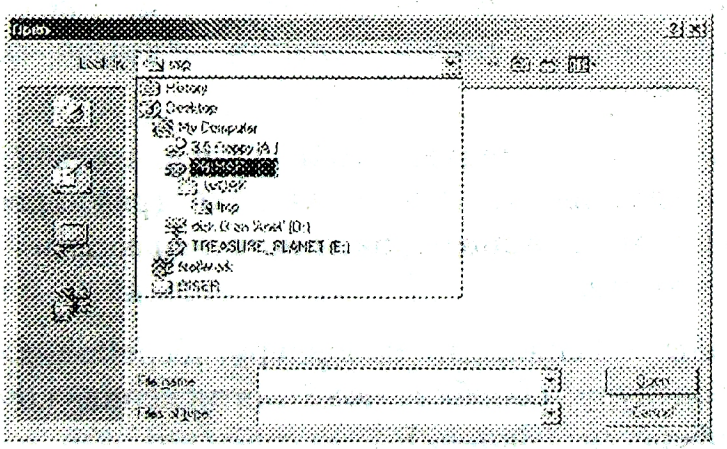

Рисунок 42

Якщо при запуску діалогу відкриття файлу користувач вибере не текстовий файл, то відбудеться помилка. Щоб її уникнути, треба дозволити користувачу вибирати тільки текстові файли. Для цього необхідно вибрати на формі компонент *OpenDialog* і в *Інспекторі Об'єктів* натиснути на кнопку з трьома крапками поряд з властивістю *Filter* - відкриється діалогове вікно *Filter Editor* (Редактор фільтра) (рис.43).

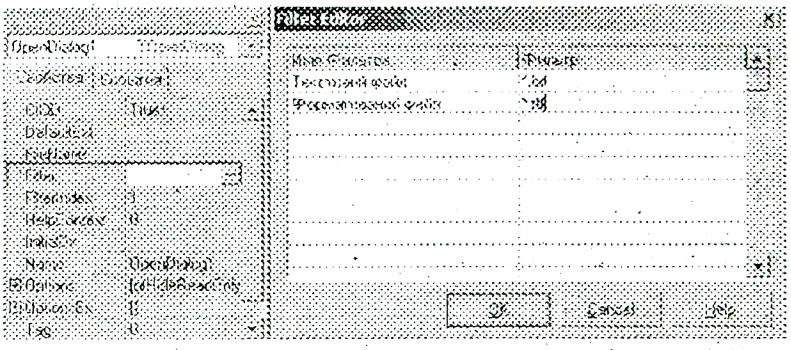

Рисунок 43

У поле *Filter Name* (ім'я фільтра) слід ввести назву, яка б характеризувала тип файлів, для яких створюється фільтр. У поле *Filter* (фільтр) – маску файлів. Додамо два фільтри для текстових (*.txt) та форматованих (*.rtf.) файлів.

Аналогічні дії виконаємо для компонента *SaveDialog*, щоб текстовий редактор зберігав файли в текстовому форматі.

До речі, в компонентів *SaveDialog* і *OpenDialog* є одна корисна властивість *DefaultExt* (розширення за замовчуванням). Бажано зробити його однаковим **.txt* для обох компонентів.

Крок 7 Напишемо процедуру збереження файлу, тобто процедуру обробки події натиснення командної кнопки "Зберегти" (*Button2*).

```
procedure TForm1.Button2Click(Sender: TObject);
begin
if SaveDialog1.Execute
   then
      RichEdit1.Lines.SaveToFile(SaveDialog1.Filename);
end;
```

При тестовому виконанні цього проекту неважко помітити, що при відкритті великих файлів не з'являється смуга прокручування. Звичайно ж це недолік і його слід виправити.

Крок 8 Додамо смуги прокручування.

За відображення смуг прокручування компонента *RichEdit* відповідає властивість *ScrollBars*. Властивість складна і надає можливість вибору з:

- *ssNone* - смуги прокручування не відображаються;
- *ssBoth* - відображаються як горизонтальна, так і вертикальна смуги;
- *ssVertical* – відображається тільки вертикальна смуга;
- *ssHorizontal* – відображається тільки горизонтальна смуга.

Виберіть ssVertical. Це забезпечить появу вертикальної смуги прокручування при редагуванні великих текстів.

Крок 9 Додамо можливість форматування тексту (зміна розміру, стилю, кольору та ін.). Використаємо компонент *FontDialog* (вкладка *Dialogs Палітри Компонентів*), який реалізує стандартний діалог Windows настроювання шрифту.

Оскільки компонент *FontDialog є також не візуальним,* то встановимо його на форму в будь-яке місце. До того ж нам знадобиться ще і командна кнопка, щоб викликати це діалогове вікно. Розмістимо її на панелі

поряд з кнопками "Відкрити" та "Зберегти". Надамо властивості *Caption* командної кнопки *Button3* значення "Шрифт". А в оброблювачі події *OnCLick* напишемо

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  if FontDialog1.Execute
    then
  Richedit1.SelAttributes.Assign(FontDialog1.Font);.
end;
```

Цей код дозволить за допомогою діалогового вікна настроювання шрифту змінювати шрифт для кожного окремого абзацу, слова, символу чи всього тексту.

На рисунку 44 подано вигляд додатка після додавання третьої кнопки і компонента *FontDialog*.

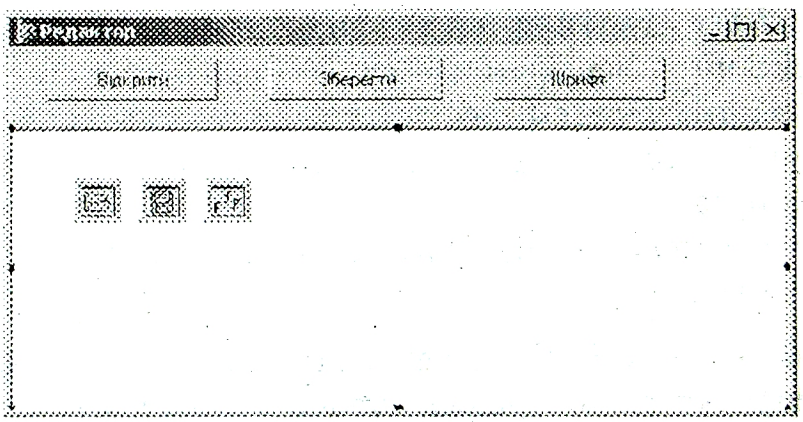

Рисунок 44

Крок 10 Розробимо головне меню додатка "Редактор".

Скористаємося компонентом *MainMenu* (вкладка *Standart Палітри Компонентів*).

Для того щоб можна було редагувати пункти меню, слід скористатися вбудованим редактором меню. Виділіть компонент *MainMenu*, а потім в *Інспекторі Об'єктів* знайдіть властивість *Items* і натисніть на кнопку з трьома

крапками. Відкриється діалогове вікно (рис. 45), і поля *Інспектора Об'єктів* зміняться відповідно.

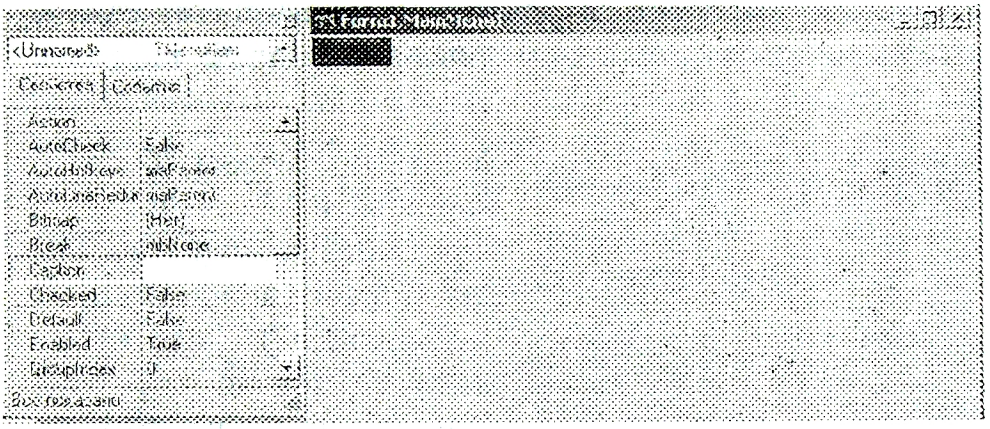

Рисунок 45

Розробимо наступну структуру меню.

Файл	Редагування	Довідка
Відкрити	Копіювати	Про програму
Зберегти	Вирізати	
Вихід	Вставити	
	Шрифти	

Натискаючи на елемент меню в *Редакторі Меню*, програміст одержує доступ до його властивостей, а також доступ до наступного елемента, що створюється автоматично. Втім, якщо автоматично створений елемент залишився незадіяним, то при виконанні програми в меню його не буде видно.

Остаточний вигляд створеного меню наведений на рис.46.

Крок 11 Створимо оброблювачі подій для пунктів головного меню.

Перейдіть на *Форму*. В пункті *Файл* головного меню подвійним клацанням миші виберіть пункт «*Відкрити*». Відкриється *Редактор коду* зі створеним шаблоном обробки події, де між *begin* і *end* треба написати код.

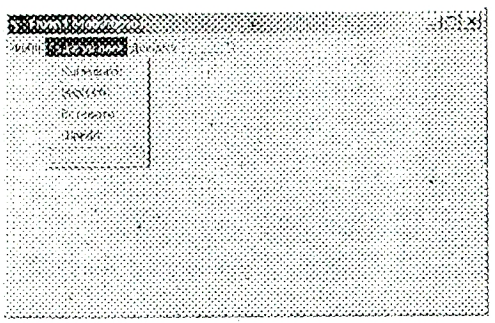

Рисунок 46

Звичайно, можна було б скопіювати код з оброблювача події *OnClick* для першої кнопки (*Button1* – "Відкрити"), але це не зовсім зручно і не раціонально: якщо є безліч кнопок і стільки ж пунктів меню, то, по-перше, розмір програми збільшується вдвічі, а, по-друге, за необхідності внести зміни потрібно буде перебирати вихідні коди. Тому зробимо лише посилання на обробку події для кнопки *Button1*. Напишіть *Button1* і поставте крапку, коли з'явитися список можливих процедур, функцій і властивостей починайте набирати *On*. Тепер видно події кнопки *Button1*, на які можна послатися, оберіть *OnClick*. У цієї функції є один параметр: *Source*: *TObject*. Тобто треба вказати джерело події - встановимо *Self*. Таким чином, посилання на оброблювач події натискання першої кнопки буде виглядати так:

button1.OnClick(self).

А код події

```
procedure TForm1.N2Click(Sender: Tobject);
 begin
        button1.OnClick(self);
end;
```

Аналогічний вигляд будуть мати і коди подій для пунктів меню «*Зберегти*» (*button2.OnClick(self)*) і «*Шрифт*» (*button3.OnClick(self)*).

Для завершення роботи програми досить викликати метод Close головної форми. Але оскільки форма в нас одна, вона і є головною. І якщо властивість належить до форми, то в коді можна не писати ім'я форми, тобто закрити нашу програму можна так: *Form1.Close* чи *Close*.

Код події, що виникає при виборі пункту "Вихід", матиме такий вигляд:

```
procedure TForm1.N4Click(Sender: TObject);
    begin
        Close;
end;
```

Код реалізації пункту "Копіювати" має такий вигляд:

```
procedure TForm1.N6Click(Sender: TObject);
    begin
        RichEdit1.CopyToClipboard;
end;
```

Аналогічний вигляд будуть мати і коди для пунктів меню "Вирізати" (*RichEdit1.CutToClipboard*) і "Вставити" (*RichEdit1.PasteFromClipboard*).

У меню «Довідка» є пункт «Про програму». Зробимо так, щоб при натисканні відкривалася нова форма з інформацією про програму й автора. Для цього в меню *File* основного меню *Delphi* виберіть пункт *New*, а в діалоговому вікні, що відкрилося, на закладці *Forms* виберіть *AboutBox* і натисніть Ok. Перед вами з'явиться шаблон форми «Про програму» (рис. 47).

Змініть інформацію цієї форми на свій розсуд. А в оброблювач події кнопки з написом "*Ok*" (*OKButton*) напишіть тільки одне слово *Close*.

Тепер перейдіть до основної форми, створіть оброблювач події *OnClick* для пункту меню "Про програму". Туди треба написати

AboutBox.ShowModal;

Тепер запустіть програму. З'явиться повідомлення про те, що форма *AboutBox* не оголошена в секції Uses. Відповісти на запит позитивно.

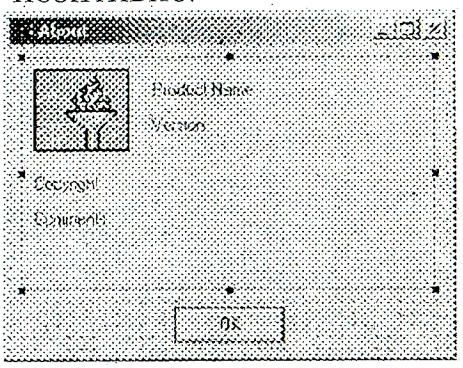

Рисунок 47

Крок 12 Привласнимо проекту "Редактор" відповідну піктограму та виконаємо збереження проекту (див. задачу 1).

Текст модулів проекту "Редактор"

```
unit Unit1;
interface
uses    Windows, Messages, SysUtils, Variants, Classes,
        Graphics, Controls, Forms, Dialogs, StdCtrls,
        ExtCtrls, ComCtrls, Menus;
type
  TForm1 = class(TForm)
    RichEdit1: TRichEdit;
    Panel1: TPanel;
    Button1: TButton;
    Button2: TButton;
```

```
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    FontDialog1: TFontDialog;
    Button3: TButton;
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    N5: TMenuItem;
    N6: TMenuItem;
    N7: TMenuItem;
    N8: TMenuItem;
    N9: TMenuItem;
    N10: TMenuItem;
    N11: TMenuItem;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure N2Click(Sender: TObject);
    procedure N3Click(Sender: TObject);
    procedure N11Click(Sender: TObject);
    procedure N4Click(Sender: TObject);
    procedure N6Click(Sender: TObject);
    procedure N7Click(Sender: TObject);
    procedure N8Click(Sender: TObject);
    procedure N10Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
```

```
  Form1: TForm1;

implementation
uses Unit2;
    {$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
if Opendialog1.Execute
    then
Richedit1.Lines.LoadFromFile(OpenDialog1.FileName);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
if SaveDialog1.Execute
    then RichEdit1.Lines.SaveToFile(SaveDialog1.Filename);
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
    if fontDialog1.execute
    then Richedit1.SelAttributes.Assign(FontDialog1.Font);
end;

procedure TForm1.N2Click(Sender: TObject);
begin
   button1.OnClick(self)
end;

procedure TForm1.N3Click(Sender: TObject);
begin
  button2.OnClick(self)
end;

procedure TForm1.N11Click(Sender: TObject);
begin
  button3.OnClick(self)
```

```
end;
procedure TForm1.N4Click(Sender: TObject);
begin
  Close;
end;

procedure TForm1.N6Click(Sender: TObject);
begin
  RichEdit1.CopyToClipboard;
end;

procedure TForm1.N7Click(Sender: TObject);
begin
   RichEdit1.CutToClipboard
end;

procedure TForm1.N8Click(Sender: TObject);
begin
   RichEdit1.PasteFromClipboard;
end;

procedure TForm1.N10Click(Sender: TObject);
begin
   AboutBox.ShowModal;
end;
end.
```

```
Unit Unit2;
interface
uses Windows, SysUtils, Classes, Graphics, Forms, Controls,
       StdCtrls, Buttons, ExtCtrls;
type
 TAboutBox = class(TForm)
   Panel1: TPanel;
   ProgramIcon: TImage;
   ProductName: TLabel;
   Version: TLabel;
```

```
    Copyright: TLabel;
    Comments: TLabel;
    OKButton: TButton;
    procedure OKButtonClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var   AboutBox: TAboutBox;
implementation

{$R *.dfm}
procedure TAboutBox.OKButtonClick(Sender: TObject);
begin
      Close
end;
end.
```

ДОДАТОК А
(обов'язковий)

Таблиця А.1

OPT File	Options Page	Editor Symbol
F	Force Far Calls	{$F+}
A	Word Align Date	{$A+}
U	Pentium-Safe FDIV	{$U+}
K	Smart Callbacks	{$K+}
W	Windows (3.0) Stack Frame	{$W+}
R	Range Checking	{$R+}
S	Stack Checking	{$S+}
I	IO Checking	{$I+}
Q	Overflow Checking	{$Q+}
V	Strict Var Strings	{$V+}
B	Complete Boolean Evaluation	{$B+}
X	Extended Syntax	{$X+}
T	Typed @ Operator	{$T+}
P	Open Parameters	{$P+}
D	Debug Information	{$D+}
L	Local Symbols	{$L+}
Y	Symbol Information	{$Y+}
N	Numeric Processing	{$N+}

ДОДАТОК Б

(обов'язковий)

Завдання для самостійної роботи

Завдання 1

Створити додаток для обчислення значень функції в кожній точці x від x_{min} до x_{max} з кроком Δx Вивести на екран значення аргументів та функції у вузлах табулювання.

1	$z=Sin^2(x+b)$	$x_{min}=-3,\ x_{max}=1$	$\Delta x=0.2$
2	$z=Sin^2(x)+ln(x)$	$x_{min}=0,\ x_{max}=5$	$\Delta x=0.5$
3	$z=ln^3(x)$	$x_{min}=1,\ x_{max}=6$	$\Delta x=0.5$
4	$z=0.5ln^3(x)+ax$	$x_{min}=1,\ x_{max}=6$	$\Delta x=0.4$
5	$z=Sin(x)Cos(x)+ax$	$x_{min}=-1,\ x_{max}=5$	$\Delta x=0.5$
6	$z=x^3+ln(x+3)$	$x_{min}=-2,\ x_{max}=5$	$\Delta x=0.25$
7	$z=e^{3x}+Cos(x)$	$x_{min}=1,\ x_{max}=6$	$\Delta x=0.4$
8	$z=e^x+tg(x)$	$x_{min}=1,\ x_{max}=6$	$\Delta x=0.4$
9	$z=Sin(x)+Cos(x)$	$x_{min}=1,\ x_{max}=6$	$\Delta x=0.5$
10	$z=(a+x)e^{3x}$	$x_{min}=6,\ x_{max}=10$	$\Delta x=0.4$
11	$z=Sin(x)/x^3$	$x_{min}=1,\ x_{max}=8$	$\Delta x=0.5$
12	$z=Sin(x)\ e^x+ln^3(x)$	$x_{min}=1,\ x_{max}=6$	$\Delta x=0.4$
13	$z=x^3/(x^3+0.5ln^3(x))$	$x_{min}=3,\ x_{max}=1$	$\Delta x=0.25$
14	$z=x-sin(x)$	$x_{min}=-3,\ x_{max}=6$	$\Delta x=0.5$
15	$z=sin(x)/\sqrt{x-0.2x}$	$x_{min}=-2,\ x_{max}=1$	$\Delta x=0.2$
16	$z=cos(x)-log_{2x}5$	$x_{min}=-1,\ x_{max}=1$	$\Delta x=0.1$
17	$z=tg(x)-ln(\pi\ cosx)$	$x_{min}=-3,\ x_{max}=1$	$\Delta x=0.5$
18	$z=arcsin(x)$	$x_{min}=-3,\ x_{max}=1$	$\Delta x=0.25$

Продовження додатка Б

19	$z=arccos(x)$	$x_{min}=-3,\ x_{max}=3$	$\Delta x=0.3$
20	$z=arctg(x)\dfrac{x}{x-\sin x}$	$x_{min}=-1,\ x_{max}=1$	$\Delta x=0.1$
21	$z=Sin(x)-cos(x)$	$x_{min}=-3,\ x_{max}=0$	$\Delta x=0.3$
22	$z=x\ sin(x)\ x^{\cos(5)}$	$x_{min}=-4,\ x_{max}=1$	$\Delta x=0.5$
23	$z=ctg(x)\sqrt[3]{\lvert x-5\rvert}$	$x_{min}=-3,\ x_{max}=6$	$\Delta x=0.3$
24	$z=x\ sin(x)$	$x_{min}=-7\ x_{max}=10$	$\Delta x=0.3$
25	$z=e^{-5x}+Sin(x-\dfrac{\pi}{x})$	$x_{min}=1,\ x_{max}=5$	$\Delta x=0.4$
26	$z=e^{x}+ctg(x^2)$	$x_{min}=1,\ x_{max}=6$	$\Delta x=0.4$
27	$z=Sin^2(x)+Cos^3(x)$	$x_{min}=1,\ x_{max}=6$	$\Delta x=0.5$
28	$z=Sin(x^2)+lg^3(x)$	$x_{min}=0,\ x_{max}=5$	$\Delta x=0.5$
29	$z=ln^3(x)-sin(2x)$	$x_{min}=1,\ x_{max}=6$	$\Delta x=0.5$
30	$z=ln(Sin(x))+aCos(x)$	$x_{min}=1,\ x_{max}=6$	$\Delta x=0.5$

Завдання 2

Створити проект згідно з варіантом. Вихідні дані – ім'я текстового файлу, який підлягає обробці, текст для пошуку та ін. - роботи програми вивести на екран та дописати повинні задаватися користувачем.

1 Визначити кількість ком та крапок у текстовому файлі. Результат в кінець файлу.

2 У текстовому файлі замінити слово "перший" на "другий". Вивести повідомлення про кількість замін на екран та додати його в кінець файлу.

3 Знайти суму чисел, які знаходяться в текстовому файлі. Вивести значення суми на екран та дописати його в кінець файлу.

4 У текстовому файлі замінити символ " - " на символ " _ ". Вивести повідомлення про кількість замін.

Продовження додатка Б

5 У текстовому файлі замінити маленькі літери на великі.

6 Створити текстовий файл *text.txt*. Внести в нього рядки символів. Створити два файли *text1.txt* і *text2*.txt. До першого файлу внести цифри із файлу *text.txt*, до другого останні символи.

7 Текстовий файл містить прізвища співробітників та номери їх телефонів. Визначити телефонний номер, який відповідає введеному із клавіатури прізвищу.

8 Створити текстовий файл та задати рядок s. Вивести на екран рядки текстового файлу, які містять рядок s, та підрахувати кількість таких рядків.

9 Створити текстовий файл. Переписати із нього в інший файл рядки, довжина яких менша 5 символів.

10 Створити текстовий файл. Встановити наявність цифр у створеному файлі. За наявності цифр знайти їх добуток. Результат вивести на екран та дописати в кінець файлу.

11 Створити текстовий файл. Знайти кількість символів у кожному рядку та переписати в інший текстовий файл, у кожному рядку якого в дужках зазначити довжину рядка, а потім і сам рядок.

12 Створити текстовий файл. Внести в нього декілька рядків символів. У створеному файлі видалити рядки, номери яких назве користувач.

13 Створити текстовий файл. Визначити кількість літер 'а', 'б', 'т'. Результат вивести на екран та дописати в кінець файлу.

14 Створити текстовий файл *words.txt*, який містить рядки символів. Додати до файлу *numbers.txt* рядки чисел, які вказують на кількість літер у словах відповідного рядка файлу *words.txt*.

Список літератури

1. Фаронов В.В. Турбо Паскаль: В 3 кн. – М.: Учебно-инженерный центр «МВТУ-ФЕСТО ДИДАКТИК», 1992.- Кн.1.
2. Фаронов В.В. Delphi 3:Учебный курс. – М.: Нолидж, 1998. – 400с.
3. Бобровский С.Н. Delphi 5: Учебный курс. – СПб.: Питер, 2001. - 640 с.
4. Баас Р., Фервей М., Гюнтер Х. Delphi 5 для пользователя /Пер. с нем. –К.: Издательская группа BHV, 2000. – 496с.
5. Джонс Ж., Харроу К. Решение задач в системе Турбо Паскаль /Пер. с англ.; Предисл. Ю.П.Широкого. – М.: Финансы и статистика, 1991.
6. Федоров А., Рогатин Д. Borland Pascal в среде Windows. – К.: Диалектика, 1994.