МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ
УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

# ІНФОРМАТИКА:

# КОНСПЕКТ ЛЕКЦІЙ

Рекомендовано Методичною радою КПІ
ім. Ігоря Сікорського як навчальний
посібник для здобувачів ступеня
бакалавра за спеціальністю 133 "Галузеве
машинобудування"

Київ

КПІ ім. Ігоря Сікорського

2019

Рецензент: *Черьопкін Євгеній Сергійович*, канд. техн. наук, доцент

Відповідальний редактор: *Корнієнко Ярослав Микитович*, д-р техн. наук, проф.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 8 від 25.04.2019 р.) за поданням Вченої ради Інженерно-хімічного факультету (протокол № 4 від 22.04.2019 р.)

Електронне мережне навчальне видання

*Сачок Роман Володимирович*, канд. техн. наук.

# ІНФОРМАТИКА:
# КОНСПЕКТ ЛЕКЦІЙ

Інформатика: конспект лекцій [Електронний ресурс]: навч. посіб. для студ. спеціальності 133 «Галузеве машинобудування», спеціалізації «Інжиніринг, комп‘ютерне моделювання та проектування обладнання хімічних і нафтопереробних виробництв» / Р.В. Сачок. –Електронні текстові данні (1 файл: 1.28 Мбайт). –Київ : КПІ ім. Ігоря Сікорського, 2019. –94 с.

Навчальний посібник присвячений розгляду основних конструкцій алгоритмів для розв’язку інженерних та наукових задач. Розглянуто методи запису лінійних, розгалужених, циклічних та комбінованих алгоритмів та приклади їх розв’язку. Викладено основи програмування в середовищі Borland Delphi, включаючи інтерфейс користувача та методи роботи з масивами. Приведено також методи побудови графіків у середовищі для подальшого користування їми як допоміжним інструментом розв’язку рівнянь, апроксимації та інтерполяції функцій.

Р.В. Сачок

КПІ ім. Ігоря Сікорського, 2019

**ЗМІСТ**

# ВСТУП

Для сучасного спеціаліста, зокрема, інженера, вкрай важливим є використання комп’ютерної техніки. Вміння грамотно скласти алгоритм для розв’язку задачі згідно вимог до нього є однією з найважливіших задач сучасного інженера та науковця.

Таким чином, вивчення студентами дисципліни “Інформатика” має на меті поєднати практичні й теоретичні відомості фізики, математики з ефективним застосуванням комп'ютерної техніки для розв’язання практичних та наукових задач і отримання результатів у наочній формі.

В посібнику розглянуті методи запису алгоритмів та їх реалізації.

Посібник призначено для студентів спеціальності «Галузеве машинобудування», що вивчають дисципліну «Інформатика».

Виходячи з того, що в повному обсязі інформація курсу не викладена в жодному посібнику або методичних вказівках, публікація є актуальною.

# 1 ІНФОРМАТИКА ТА АЛГОРИТМІЗАЦІЯ

## 1.1 Предмет та основні поняття інформатики

**Інформатика** – це комплексна, технічна наука, що систематизує прийоми створення, збереження, відтворення, обробки та передачі даних засобами обчислювальної техніки, а також принципи функціонування цих засобів та методи керування ними. Термін "інформатика" походить від французького слова Informatique і утворене з двох слів: інформація та автоматика. Запроваджено цей термін у Франції в середині 60-х років XX ст., коли розпочалося широке використання обчислювальної техніки. Тоді в англомовних країнах увійшов до вжитку термін "Computer Science" для позначення науки про перетворення інформації, що ґрунтується на використанні обчислювальної техніки. Тепер ці терміни є синонімами.

Поява інформатики зумовлена виникненням і поширенням нової технології збирання, оброблення і передачі інформації, пов'язаної з фіксацією даних на машинних носіях.

**Предмет інформатики як науки складають:**

- апаратне забезпечення засобів обчислювальної техніки;
- програмне забезпечення засобів обчислювальної техніки;
- засоби взаємодії апаратного та програмного забезпечення;
- засоби взаємодії людини з апаратними та програмними засобами.

Засоби взаємодії в інформ  атиці прийнято називати інтерфейсом. Тому засоби взаємодії апаратного та програмного забезпечення інколи називають також програмно-апаратним інтерфейсом, а засоби взаємодії людини з апаратними та програмними засобами – інтерфейсом користувача.

**Основною задачею інформатики** як науки є систематизація прийомів та методів роботи з апаратними та програмними засобами обчислювальної техніки. Мета систематизації полягає у тому, щоб виділити, впровадити та

розвинути передові, найбільш ефективні технології автоматизації етапів роботи з даними, а також методично забезпечити нові технологічні дослідження.

Інформатика - практична наука. Її досягнення повинні проходити перевірку на практиці і прийматися в тих випадках, коли вони відповідають критерію підвищення ефективності. У складі основної задачі сьогодні можна виділити такі основні напрямки інформатики для практичного застосування :

- архітектура обчислювальних систем (прийоми та методи побудови систем, призначених для автоматичної обробки даних);
- інтерфейси обчислювальних систем (прийоми та методи керування апаратним та програмним забезпеченням);
- програмування (прийоми, методи та засоби розробки комплексних задач);
- перетворення даних (прийоми та методи перетворення структур даних);
- захист інформації (узагальнення прийомів, розробка методів і засобів захисту даних);
- автоматизація (функціонування програмно-апаратних засобів без участі людини);
- стандартизація (забезпечення сумісності між апаратними та програмними засобами, між форматами представлення даних, що відносяться до різних типів обчислювальних систем).

На всіх етапах технічного забезпечення інформаційних процесів для інформатики ключовим питанням є ефективність. Для апаратних засобів під ефективністю розуміють співвідношення продуктивності обладнання до його вартості. Для програмного забезпечення під ефективністю прийнято розуміти продуктивність користувачів, які з ним працюють. У програмуванні під ефективністю розуміють обсяг програмного коду, створеного програмістами за одиницю часу. В інформатиці все жорстко орієнтоване на ефективність. Питання як здійснити ту чи іншу операцію, для інформатики є важливим, але не основним. Основним є питання як здійснити дану операцію ефективно.

В межах інформатики, як технічної науки можна сформулювати поняття інформації, інформаційної системи та інформаційної технології.

**Інформація** – це сукупність відомостей (даних), які сприймають із навколишнього середовища (вхідна інформація), видають у навколишнє середовище (вихідна інформація) або зберігають всередині певної системи.

Інформація існує у вигляді документів, креслень, рисунків, текстів, звукових й світлових сигналів, електричних та нервових імпульсів тощо. Саме слово «інформатика» походить від латинського information, що означає виклад, роз'яснення факту, події.

**Найбільш важливими властивостями інформації є:**

– об'єктивність та суб'єктивність;
– повнота;
– достовірність;
– адекватність;
– доступність;
– актуальність.

Дані є складовою частиною інформації, що являють собою зареєстровані сигнали. Під час інформаційного процесу дані перетворюються з одного виду в інший за допомогою методів. Обробка даних містить в собі множину різних операцій. Основними операціями є:

– збір даних – накопичення інформації з метою забезпечення достатньої повноти для прийняття рішення;
– формалізація даних – приведення даних, що надходять із різних джерел до однакової форми;
– фільтрація даних – усунення зайвих даних, які не потрібні для прийняття рішень;
– сортування даних – впорядкування даних за заданою ознакою з метою зручності використання;
– архівація даних – збереження даних у зручній та доступній формі;
– захист даних – комплекс дій, що скеровані на запобігання втрат, відтворення та модифікації даних;

– транспортування даних – прийом та передача даних між віддаленими користувачами інформаційного процесу. Джерело даних прийнято називати сервером, а споживача – клієнтом;

– перетворення даних – перетворення даних з однієї форми в іншу, або з однієї структури в іншу, або зміна типу носія.

**Інформаційна система**

В інформатиці поняття *"система"* найчастіше використовують стосовно набору технічних засобів і програм. Системою називають також апаратну частину комп'ютера. Доповнення поняття "система" словом "інформаційна" відображає мету її створення і функціонування.

*Інформаційна система* – взаємозв'язана сукупність засобів, методів і персоналу, використовувана для зберігання, оброблення та видачі інформації з метою вирішення конкретного завдання.

Сучасне розуміння інформаційної системи передбачає використання комп'ютера як основного технічного засобу обробки інформації. Комп'ютери, оснащені спеціалізованими програмними засобами, є технічної базою та інструментом інформаційної системи.

У роботі інформаційної системи можна виділити такі етапи:

1. Зародження даних – формування первинних повідомлень, що фіксують результати певних операцій, властивості об'єктів і суб'єктів управління, параметри процесів, зміст нормативних та юридичних актів тощо.

2. Накопичення і систематизація даних – організація такого їх розміщення, яке б забезпечувало б швидкий пошук і відбір потрібних відомостей, методичне оновлення даних, захист їх від спотворень, втрати, деформування цілісності та ін.

3. Обробка даних – процеси, внаслідок яких на підставі раніше накопичених даних формуються нові види даних: узагальнюючі, аналітичні, рекомендаційні, прогнозні. Похідні дані також можуть

зазнавати подальшого оброблення, даючи відомості глибшої узагальненості і т.д.

4. Відображення даних – подання їх у формі, придатній для сприйняття людиною. Передусім, це виведення на друк, тобто виготовлення документів на так званих твердих (паперових) носіях. Широко використовують побудову графічних ілюстративних матеріалів (графіків, діаграм) і формування звукових сигналів.

Повідомлення, що формуються на першому етапі, можуть бути звичайним паперовим документом, повідомленням у "машинному вигляді" або тим й іншим одночасно. В сучасних інформаційних системах повідомлення масового характеру здебільшого мають "машинний вигляд". Апаратура, що використовується при цьому, має назву засоби реєстрації первинної інформації.

Потреби другого і третього етапів задовольняються в сучасних інформаційних системах в основному засобами обчислювальної техніки. Засоби, що забезпечують доступність інформації для людини, тобто засоби відображення даних, є компонентами обчислювальної техніки.

Переважна більшість інформаційних систем працює в режимі діалогу з користувачем. Типові програмні компоненти інформаційних систем включають: діалогову підсистему введення-виведення, підсистему, яка реалізує логіку діалогу, підсистему прикладної логіки обробки даних, підсистему логіки управління даними. Для мережевих інформаційних систем важливим елементом є комунікаційний сервіс, який забезпечує взаємодію вузлів мережі при спільному вирішенні задачі. Значна частина функціональних можливостей інформаційних систем закладається в системному програмному забезпеченні: операційних системах, системних бібліотеках та конструкціях інструментальних засобів розробки. Крім програмної складової інформаційних систем важливу роль відіграє інформаційна складова, яка задає структуру, атрибутику та типи даних, а також тісно пов'язана з логікою управління даними.

**Інформаційні технології**

В широкому сенсі слово технологія – це спосіб освоєння людиною матеріального світу за допомогою соціально організованої діяльності, що включає три компоненти: інформаційну(наукові принципи та обґрунтування), матеріальну(знаряддя праці) та соціальну(фахівці, які мають професійні навички). Ця тріада становить сутність сучасного розуміння поняття технологія.

Поняття інформаційної технології з'явилося з виникненням інформаційного суспільства, основою соціальної динаміки в якому є не традиційні матеріальні, а інформаційні ресурси: знання, наука, організаційні чинники, інтелектуальні здібності, ініціатива, творчість і т.д. На жаль, це поняття є настільки загальним та всеохоплюючим, що до сих пір фахівці не прийшли до чіткого, формалізованого формулювання. На думку авторів, найбільш вдалим є визначення поняття інформаційної технології дане академіком Глушковим В.М., який трактував її як людино-машинну технологію збирання, обробки та передачі інформації, що ґрунтується на використанні обчислювальної техніки. Ця технологія швидко розвивається, охоплюючи всі види суспільної діяльності: виробництво, управління, науку, освіту, фінансово-банківські операції, медицину, побут та ін.

**Кодування даних**

Для автоматизації роботи з даними, що відносяться до різних типів, важливо уніфікувати їх форму представлення. Для цього, як правило, використовується прийом кодування, тобто представлення даних одного типу через дані іншого типу. Звичайні людські мови можна розглядати як системи кодування ідей та понять для вираження думок за допомогою мовлення. Іншим прикладом загальновживаних систем кодування може бути азбука, як система кодування компонентів мови за допомогою графічних символів. Універсальні засоби кодування успішно втілюються в різноманітних галузях техніки, науки та культури - математичні вирази, телеграфна азбука, морська азбука, азбука для

сліпих тощо. Своя система кодування існує й в інформатиці, і називається вона двійковим кодом. Ґрунтується вона на представленні даних послідовністю двох знаків: 0 та 1. Ці знаки називають двійковими цифрами або бітами (від скорочення англійських слів binary digit). Слід зауважити, що вся інформація, що зберігається та обробляється засобами обчислювальної техніки, незалежно від її типу (числа, текст, графіка, звук, відео), представлена у двійковому коді.

Одним бітом можна виразити два поняття: 0 або 1 (ні або так, хибне або істинне). Якщо кількість бітів збільшити до двох, то тоді можна вже закодувати чотири поняття : 00, 01, 10, 11. Трьома бітами кодують вісім понять: 000, 001, 010, 011, 100, 101, 110, 111. Збільшуючи на одиницю кількість розрядів в системі двійкового кодування, ми збільшуємо в два рази кількість значень, які можуть бути виражені в цій системі кодування, тобто кількість значень вираховується за формулою:

$N = 2^m$

де N – кількість незалежних значень, що кодуються, m – розрядність двійкового кодування.

- Найменшою одиницею об'єму даних прийнято вважати байт – групу з 8 бітів. Байтом можна закодувати, наприклад, один символ текстової інформації. Наступним одиницями кодування є:
- кілобайт (Кбайт): 1 Кбайт = $10^{10}$ байт = 1024 байт;
- мегабайт (Мбайт): 1 Мбайт = $10^{10}$ Кбайт = 1024 Кбайт;
- гігабайт (Гбайт): 1 Гбайт = $10^{10}$ Мбайт = 1024 Мбайт;
- терабайт (Тбайт): 1 Тбайт = $10^{10}$ Гбайт = 1024 Гбайт.

Саме в таких одиницях вимірюється ємність даних в інформатиці.

## 1.2 Алгоритми

### Визначення алгоритму

Алгоритм – одне з основних понять математики, що неможливо строго визначити, а лише можна пояснити за допомогою інших слів. Близькими за змістом є слова «рецепт», «метод», програма й ін. Можна сказати, що:

*алгоритм* – це точний і зрозумілий опис послідовності дій над заданими об'єктами, що дозволяє отримати кінцевий результат.

Одним з найдавніших є алгоритм Евкліда для знаходження найбільшого спільного дільника двох натуральних чисел. Одне з можливих формулювань цього алгоритму:

1. Присвоїти змінним X та Y значення, найбільший спільний дільник яких шукається.
2. Якщо X>Y, перейти до п.5
3. Якщо X<Y, перейти до п.6
4. (X=Y)Видати X в якості результату. Кінець роботи.
5. Замінити пари (X,Y) парою (X – Y, Y) і повернутися до п.2.
6. Замінити пари (X,Y) парою (X , Y- X ) і повернутися до п.2.

| X | Y | X>Y? | X<Y? |
|---|---|---|---|
| 100 | 80 | так | ні |
| 20 | 80 | ні | так |
| 20 | 60 | ні | так |
| 20 | 40 | ні | так |
| 20 | 20 | ні | ні |

Алгоритм повинний мати такі властивості:

1. Кінцівка. Робота алгоритму повинна закінчуватися за кінцеве число кроків.
2. Визначеність. Всі розпорядження алгоритму повинні допускати однозначне трактування і бути зрозумілі тому, хто буде виконувати алгоритм – виконавцю. Розпорядження, зрозумілому одному виконавцю, можуть бути незрозумілі для іншого.
3. Введення. Алгоритм повинний мати властивість масовості, тобто давати рішення цілої групи задач, що відрізняються вихідними

даними.

4. Виведення. Алгоритм повинний давати деякі результати.
5. Ефективність. Усі кроки алгоритму повинні бути такими, щоб виконавець міг виконати їх за кінцевий час. Крім того, час роботи алгоритму повинний не просто кінцевим, але і лежати в деяких розумних межах.

**Методи запису алгоритмів**

Для представлення алгоритмів можна користатися різними способами їхнього запису, що відрізняються за ступенем наочності. Одні способи орієнтовані на виконавця – людину, інші – на виконання комп'ютером, треті є допоміжними (використовуються для полегшення міркувань). Розглянемо два найбільш прості способи - запис за допомогою звичайної мови спілкування і запис з використанням блок-схем. Однак перш, ніж перейти до цих питань, ми пояснимо поняття «величина» і «присвоювання», що часто використовуються при складанні й аналізі алгоритмів.

**Величини й ідентифікатори**

Об'єкти матеріального світу описуються за допомогою величин. Наприклад, для опису руху автомобіля вводяться величини швидкості, прискорення, часу в дорозі. Інші приклади величин, з якими ви зустрічаєтеся щодня, це: відстань між будинком і школою, температура повітря і т.д. Кожна величина характеризується деяким значенням, наприклад, швидкість може бути дорівнює 80 км/година, відстань - 700 м, а температура - 25°С.

Величина - це об'єкт, з яким зв'язується визначена множина значень.

Величини можуть змінюватися в часі, у цьому випадку вони називаються *змінними.* Якщо ж величина незмінна, то неї називають постійною чи *константою.*

Алгоритм можна розглядати як спосіб визначення значень деяких величин. Так, якщо ви створюєте алгоритм розв’язку системи рівнянь, то вихідними

даними повинні бути значення невідомих. Якщо створюється комп'ютерна модель парового котла, то на виході будуть значення температури і тиску в різних частинах котла. Можна придумати безліч прикладів, коли в результаті комп'ютерного моделювання визначаються значення величин.

З визначенням значень величин зв'язане також створення текстів, різних списків, графічних зображень та інш. У цих випадках величини приймають значення, що відповідають фрагментам тексту, елементам списків, компонентам зображень і т.д. Взагалі мовою величин можна виразити будь-яку інформацію.

Величини звичайно позначаються окремими символами (наприклад, a, b, c, j, x, z) чи послідовностями символів (наприклад, al, b13d, x10, itog). Позначення тієї чи іншої величини в програмі називається *ідентифікатором* величини.

*Ідентифікатор* – це ім'я, що вибирається для елементів алгоритму: змінних, констант, заголовків.

Прикладами ідентифікаторів величин можуть бути наступні послідовності символів: A, Y2C, 15, X, Y, SI, DAT_33, My_program і т.д.

**Присвоювання**

Найпростішою і найбільш вживаною інструкцією алгоритму є *присвоювання.* Приведемо приклади інструкції присвоювання:

a:=13;

dl:=c;

x:=x+l.

У лівій частині інструкції ставиться ідентифікатор величини, а в правій частині – звичайна форма алгебраїчного виразу. В операторах присвоювання використовується або звичний знак рівності, або сполучення двокрапки і знака рівності : =. Оскільки знак присвоювання – це не знак рівності, можливі записи типу приведеної вище (x: =x+1) чи a: =a-b.

Після виконання інструкції присвоювання змінній присвоюється значення деякого виразу. Потрібно враховувати, що інструкція присвоювання буде

виконуватися тільки в тому випадку, якщо значення всіх змінних у правій частині уже визначені.

**Словесний запис алгоритму**

Словесний спосіб запису заснований на тій чи іншій природній мові спілкування (див. приведений вище алгоритм Евклида). Однак словесний запис алгоритму відрізняється від звичайних мовних конструкцій більш ретельним підбором слів і фраз, при якому не допускається повторень або двозначного тлумачення. Крім того, у записі алгоритму можуть використовуватися математичні символи і вирази.

Розглянемо словесний спосіб запису ще на одному простому прикладі. Нехай потрібно знайти модуль величини X (тобто значення |X|) і присвоїти це значення змінній Y. При цьому скористаємося визначенням модуля: |x| =x при x>0 і |x| =-x при x<0.

Алгоритм розрахунку модуля можна записати в такий спосіб:

1. Початок.
2. Ввести числове значення величини X.
3. Якщо X>0, то Y присвоїти значення X, інакше Y присвоїти значення –X.;
4. Вивести значення Y.
5. Кінець.

Словесний запис найчастіше застосовується на початковому етапі вивчення алгоритмів і призначається для використання алгоритму людиною. Однак ця форма запису алгоритму має два істотних недоліки. По-перше, вона недостатньо наочна, тому що з її допомогою не можна графічно представити алгоритм рішення задачі. По-друге, словесний запис важко безпосередньо перекласти на мову програми.

**Блок-схеми**

Найбільш наочною формою запису алгоритмів є блок-схеми, що складаються з геометричних фігур – блоків. Кожен блок відповідає визначеній дії. Наприклад, запис алгоритму починається і закінчується наступними блоками:


Рисунок 1.1 – Блоки початку і кінця у блок-схемі алгоритму

Ці елементи називаються *блоками початку і кінця алгоритму (термінаторами)*. Стрілки позначають напрямок виконання алгоритму. Блок Початок має одну вихідну стрілку, а блок Кінець - одну вхідну стрілку.

В алгоритмах часто зустрічаються команди введення і виведення значень. Цим командам відповідають *блоки введення-виведення,* що мають наступний вигляд:


Рисунок 1.2 – Блоки введення та виведення

Тут лівий блок позначає введення величини x, а правий блок - виведення y. За допомогою приведених вище блоків ви можете скласти найпростіший алгоритм введення величини x.

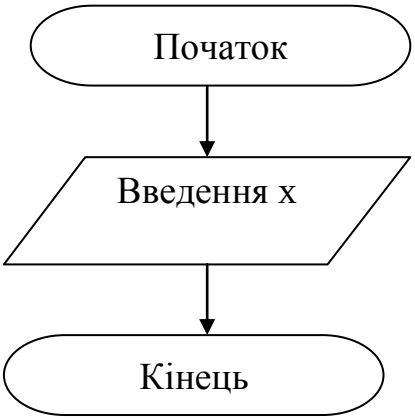



Рисунок 1.3 – Блок-схема введення

Відповідно до цього алгоритму в програму вводиться значення величини x. Однак програма, що складається тільки з операції введення, навряд чи мала би зміст. Звичайно над уведеною величиною виконуються визначені дії.

Дії, виконувані в програмі, позначаються прямокутними блоками виду:

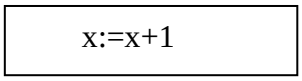



Рисунок 1.4 – Блок дії

які називаються *операторними блоками.* Всередині прямокутників записані вирази, виконувані над величинами. Лівий блок відповідає операції присвоювання: змінній x присвоюється значення суми x+1. Запишемо найпростіший алгоритм обчислення квадрата введеної змінної, рисунок 1.5

Відповідно до цього ал горитму виконується введення величини x, потім обчислюється квадрат цієї величини (добуток x*x) і виводиться отримане значення.

Всі приведені вище блоки дозволяють організувати послідовне виконання інструкцій алгоритму. Однак на практиці часто виникають ситуації, коли в залежності від виконань якої-небудь умови потрібно змінити послідовний хід обчислень.

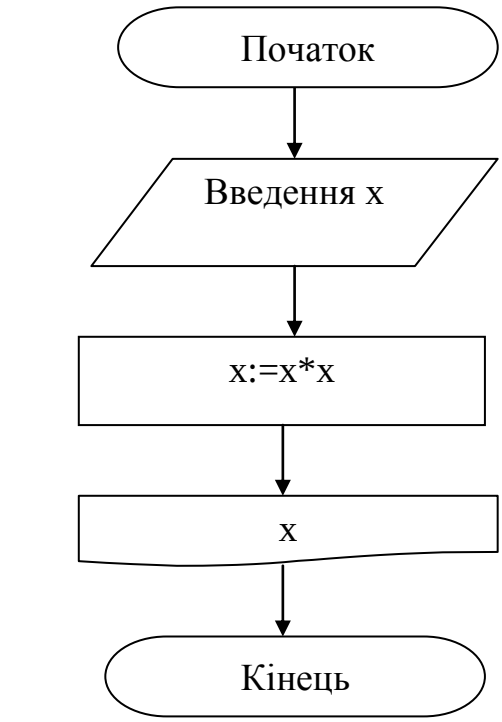



Рисунок 1.5 – Блок-схема обчислення квадрату введеної змінної

Прикладом такої умови є нерівність X>=0 в алгоритмі знаходження модуля числа X (див. попередній пункт). У схему алгоритму логічна умова вводиться за допомогою *умовного блоку*.

Цей блок прийнятий зображувати у виді ромба з одним входом і двома виходами:


Рисунок 1.6 – Умовний блок (розгалуження)

Якщо умова, зазначена на зображенні блоку, виконується (умова має значення Істина), то відбувається перехід по стрілці Так, якщо не виконується (значення Неправда) - по стрілці Ні.

Завдяки умовному блоку обчислювальний процес як би розгалужується: у залежності від чи виконання невиконання умови реалізується та чи інша гілка алгоритму. Таким чином, умовний блок використовується для

організації *розгалуження.* Наведемо як приклад алгоритм обчислення модуля числа.

Запис цього алгоритму обмежують блоки початку і кінця алгоритму. За блоком початку алгоритму слідує блок уведення значень x, а за ним - умовний блок. В умовному блоці виконується перевірка умови x≤0 і в результаті перевірки здійснюється перехід по одній з гілок Так чи Ні. На кожній з гілок знаходиться операторний блок присвоювання значень змінної y. Після операції присвоювання галузі алгоритму сходяться, і наступна інструкція алгоритму міститься в блоці виведення отриманого значення y.

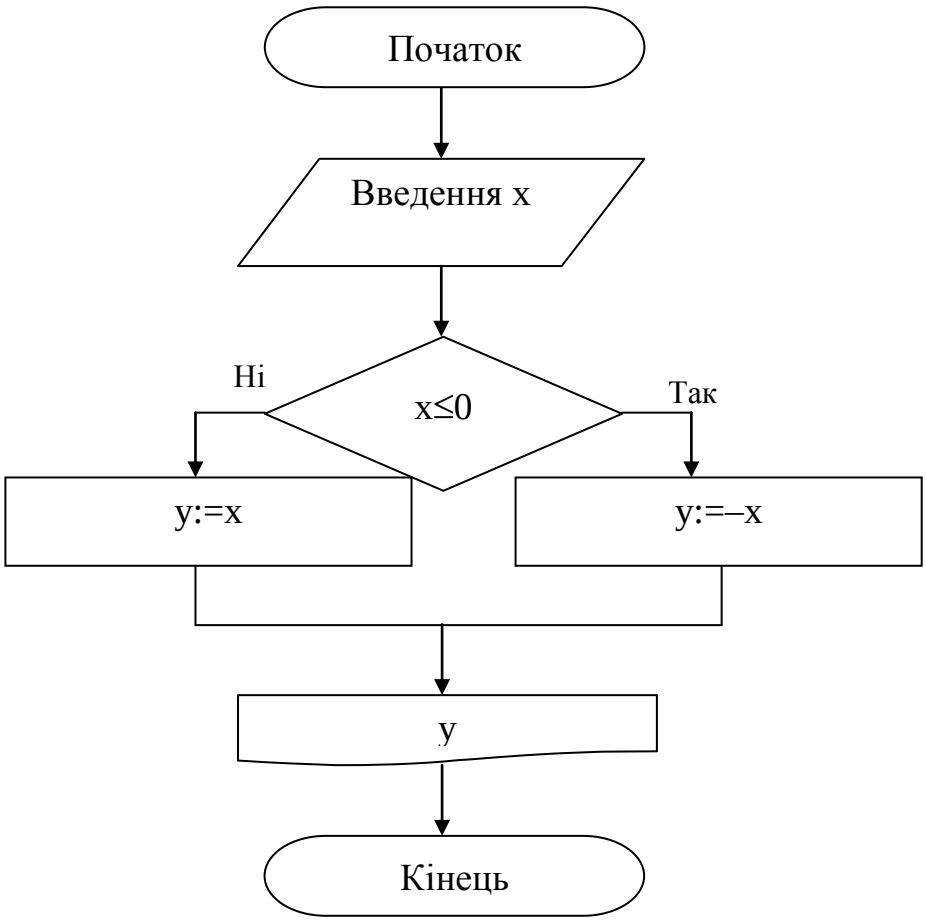



Рисунок 1.7 – Визначення модуля введеної змінної

Цикл схематично зображається одним із вказаних способів:


Рисунок 1.8 – Зображення циклу.

Фрагмент схеми, що зображає вихід із частини схеми та вхід з другої частини схеми :

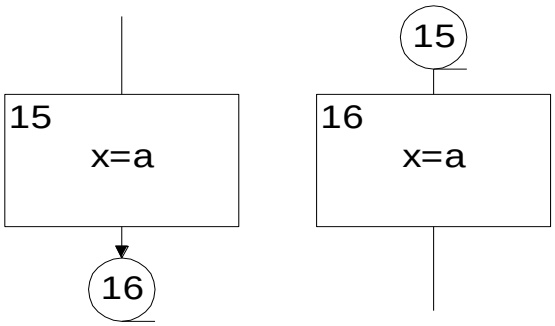



Рисунок 1.9 – Зображення коммунікатора (розривання блок-схеми)

Тобто, виходячи з вищенаведеного, основні типи алгоритмів такі:

- лінійні алгоритми;
- розгалужені алгоритми;
- циклічні алгоритми;
- комбіновані (де зустрічається усі три типи у різних комбінаціях).

Зазвичай будь-який не учбовий алгоритм є комбінованим.

**Циклічні алгоритми**

Циклом називається алгоритм, який повторюється певну кількість разів в залежності від деякої визначеної умови. Цикли класифікують на цикл з параметром (арифметичний) та ітераційний цикл.

*Цикл з параметром* – це конструкція, що виконується доти, доки значення параметру ***не стане більшим за кінцеве значення***.

**Приклад** Скласти алгоритм для розрахунку всіх значень функції $y = f(x)$, в інтервалі зміни $x$ від 1 до 100 включно, з кроком $\Delta x = 2$.

Розв'язок

1.

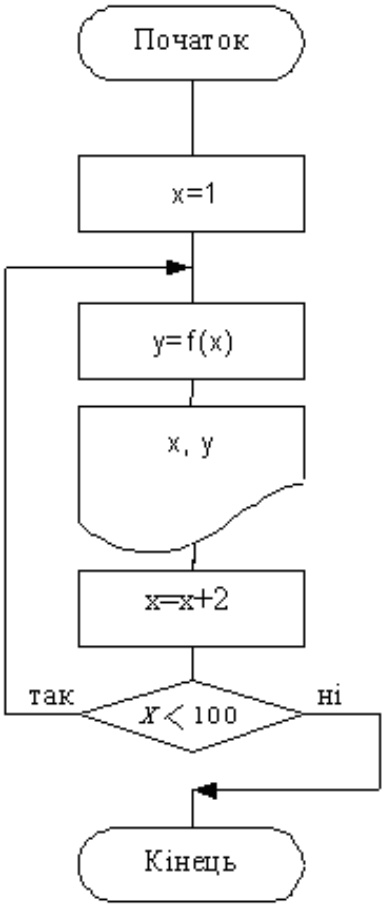



Схема без використання символу, що означає цикл.

2.

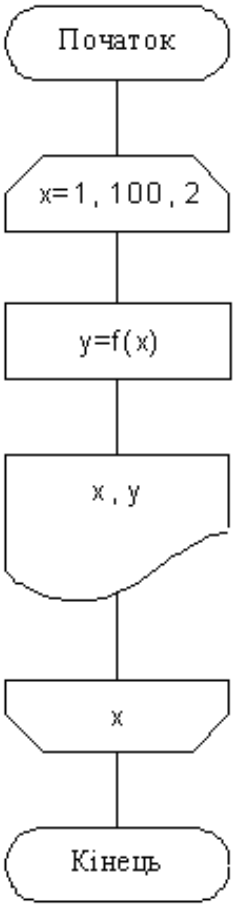



Схема з використанням символу "границя циклу".

Рисунок 1.10 – Приклад циклічного алгоритму

Друга схема – більш компактна.

Початкове, кінцеве значення змінної та крок змінної записуються в одній з двох часток символу “границя циклу” через кому або крапку з комою. Значення кроку можна не вказувати, якщо він дорівнює одиниці.

*Ітераційним циклом* називається конструкція, що виконується деяку, поперньо невідому кількість разів в залежності від поставленої алгоритмом умови. В більшості випадків використовується для розрахунку наближених значень – ітераційного процесу. Ітерація – це наближення значення змінної (функції) до значення з заданою точністю. В більшості випадків точність обраховується як модуль різниці поточного значення функції та попереднього її значення.

З ітераційним циклом тісно пов’язане визнаення рекурсії. *Рекурсія* – це визначення значення поточної змінної (функції) через її попереднє значення. Найпростішим прикладом рекурсії можна вважати x:=x+1 – зміну значення ітераційного циклу.

**Приклад 2.**Скласти алгоритм знаходження такого найменшого цілого додатного числа n, при якому функція $y=(0,5)^n/n < 10^{-5}$

***Розв’язок***

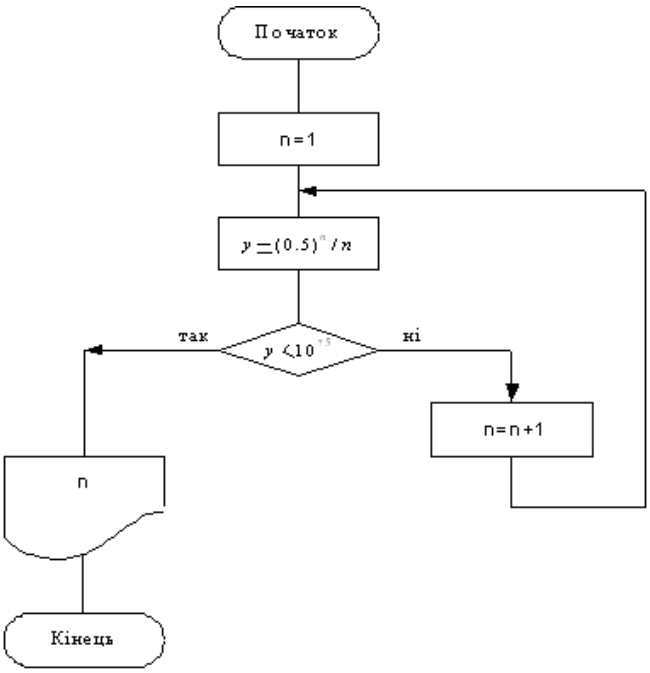



Рисунок 1.11 – Приклад складання ітераційного циклу.

*Вкладений цикл* - алгоритм відображає цикл в циклі, або один із циклів вкладений в тіло циклу. Допускається необмежена кількість вкладень. Взагалі вкладений цикл відпрацьовується таким чином: спочатку відпрацьовується перше значення параметру зовнішнього циклу, потім відповідно повністю виконується внутрішній цикл (знаечення парметру зовнішнього циклу зберігається), після цього відпрацьовується друге значення зовнішнього циклу і знову повністю виконується внутрішній цикл (від початкового значення його параметру до кінця), так доти, доки зовнішній цикл не перевищить кінцевого значення свого параметру.

Найпростіший приклад наведено на рисунку 1.12.

**Приклад 3.** Робота арифметичного циклу: i змінюється від 1 до 3 з кроком 1, j змінюється від 1 до 5 з кроком 1. На екран виводяться значення змінних i та j на кожнім кроці циклу.

**Розв'язок:**

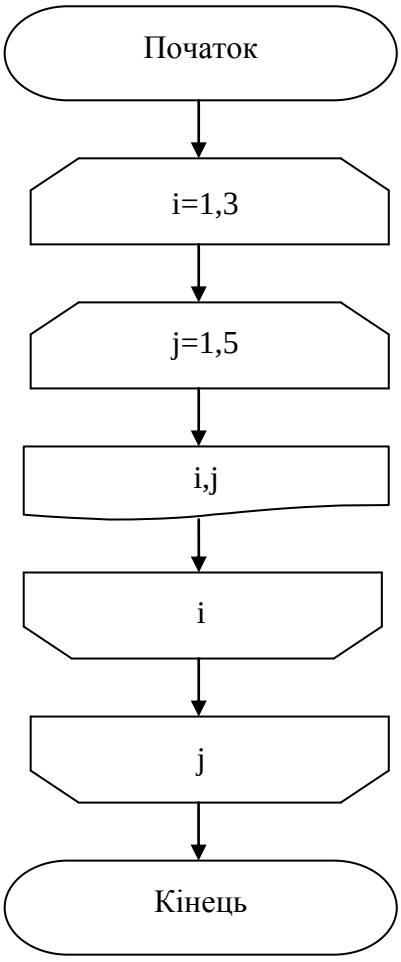



Рисунок 1.12 – Найпростіший приклад вкладеного циклу.

У відповідності із цим алгоритмом, буде виведено: 1,1 1,2 1,3 1,4 1,5 2,1 2,2 2,3 2,4 2,5 3,1 3,2 3,4 3,5.

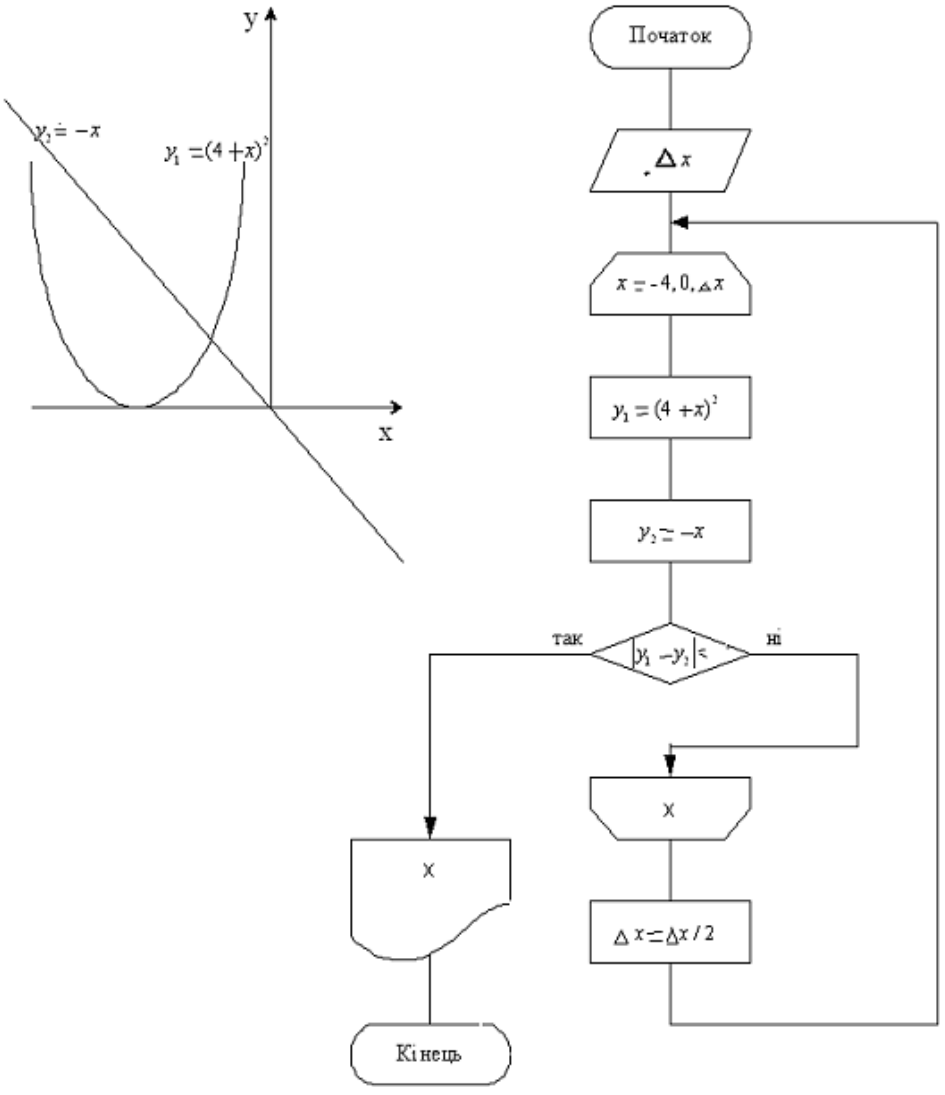
**Приклад 4.** Задано функції:

$$y_1=(4+x)^2,$$

$$y_2=-x.$$

Скласти алгоритм для знаходження точки перетину цих функцій при $-4 \le x \le 0$. Обчислення зробити з точністю до величини $\varepsilon$.

***Розв'язок***




Цей алгоритм реалізовує вкладення циклу з параметром в ітераційний цикл. Якщо значення за модулем різниці двох функцій менше заданої точності $\varepsilon$, розрахунок необхідно припинити. Якщо весь шлях по вісі x буде пройдено й умова виходу не виконана, необхідно зменшити крок та повторити цикл з

параметром з початку. Так до тих пір, доки умова ітераційного циклу не буде виконана. В загальному випадку, якщо функції не перетинаються, алгорим стане зацикленим, тому для початку розв’язку краще побудувати графіки функцій у будь-який відомий вам спосіб для визначення інтервалу їх перетину.

Якщо задані функції квадратична та лінійна, можливо також знайти точку перетину їх, прирівнявши та привевши отримане квадратне рівняння до канонічного вигляду $ax^2+bx+c = 0$.

**Одновимірні масиви**

*Одновимірний масив* — це послідовність однотипних даних.

Уважно проаналізувавши це означення, можна зробити висновок, що масив фактично поєднує в собі дві структури: множину елементів і заданий на цій множині порядок. Усі елементи масиву мають один і той самий тип, що називається базовим. З іншого боку, порядок теж визначається набором значень одного й того самого типу, що називається індексним, а самі ці значення називаються індексами. Кожному елементу масиву відповідає певний індекс. Індексний тип має бути простим порядковим типом даних. Кількість елементів в одновимірному масиві називається його розмірністю, або довжиною.

Щойно наведене визначення типу масиву можна застосувати до типів як одновимірних, так і багатовимірних масивів. Зауважимо також, що усі елементи одновимірного масиву записуються до розташованих поряд ділянок оперативної пам'яті. Тому і весь масив може розглядатися як одна нерозривна область пам'яті.

З точки зору математики одновимірний масив — це вектор. Наприклад, масив або вектор А, що має п'ять елементів, які записують у математиці у вигляді індексованих змінних $a_1$, $a_2$, $a_3$, $a_4$, $a_5$, можна зобразити значеннями цих змінних у сусідніх ділянках оперативної пам'яті.

**Двовимірні масиви**

В багатьох випадках інформація може бути представлена більш компактно, чітко та зрозуміло, якщо для її представлення використовується більше, ніж один просторивий вимір.

Можливо використовувати масиви від одного до восьми вимірів, хоч розмірність більше трьох неможливо представити графічно.

Двовимірні масиви даних найбільш просто представити у вигляді таблиці (матриці). Наприклад, таблиця множення може бути записана в масиві **TABL (9,9):**

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **2** | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 |
| **3** | 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 |
| **4** | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 |
| **5** | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| **6** | 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 |
| **7** | 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 |
| **8** | 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 |
| **9** | 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 |

Найпростішим прикладом тривимірного масиву є тривимірний простір.

Необхідно зауважити, що найчастіше робота з масивами відбувається з використанням циклічних алгоритмів. У випадку з одновимірними масивами – це простий цикл з параметром, для випадку двовимірних циклів, відповідно, вкладеного циклу і так далі.

# 2 ПРОГРАМУВАННЯ У СЕРЕДОВИЩІ BORLAND DELPHI

## 2.1 Основні відомості

Традиційно при вивченні програмування прийнято створювати першу програму, яка виводить текст «Hello, world!». Не будемо відступати від традиції і створимо програму, яка виводить цей текст трьома різними способами. Але спочатку познайомимося з самої середовище програмування Delphi. Передбачається, що на цей момент Delphi 7 вже встановлена на вашому ПК. Якщо це не так, то перед подальшим прочитанням лекції встановіть Delphi 7. При завантаженні Delphi 7 ви бачите таку картину:

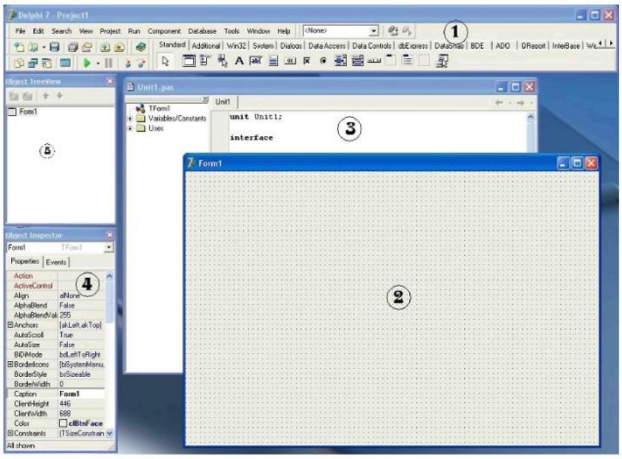



Рисунок 2.1 – Робоче середовище Delphi 7.

Основні вікна середовища:

1. Головне вікно Delphi. Тут знаходиться основне меню, різні панелі інструментів і палітра компонентів, що складається з вкладок.

2. Конструктор форми. Тут ми візуально бачимо, як буде виглядати форма програми, тут ми будемо створювати інтерфейс, переносячи на форму різні компоненти, і розставляючи їх таким чином, щоб інтерфейс виглядав привабливим. Нам часто доводиться перемикатися між конструктором форм і редактором коду, робиться це клавішу F12.

3. Редактор коду. Тут ми бачимо вихідний код програми, який створений самою Delphi. Тут же ми будемо вводити наш власний код.

4. Об'єктний інспектор. Він призначений для управління об'єктами проекту і

складається з двох вкладок – Properties (Властивості) і Events (Події).

5. Дерево об'єктів. Тут ми бачимо, який саме об'єкт в даний момент є поточним. Це вікно буде особливо корисно, коли на формі з'явиться безліч компонентів.

Коли відкривається Delphi, вона автоматично створює і підтримує новий проект (програму). На рисунку ви бачите проект, який містить тільки одну стандартну форму. Форма - це теж об'єкт, який являє собою вікно програми. Щоб програма робила щось корисне, нам доведеться вносити зміни до неї. Виведемо текст «Hello, world!» першим способом. Для цього в інспекторі об'єктів знайдіть властивість Caption. В даний момент поточних є об'єкт - форма, і властивість Caption форми відповідає за напис на системній рядку програми (синя смуга зверху будь-якого програмного вікна). За замовчуванням, властивість Caption містить напис «Form1», також називається і сама форма. Змініть цей напис на «Hello, world!» (звичайно, без лапок). Вже в процесі введення тексту ви бачите, що напис в системної рядку приймає новий вигляд. Ми ввели цей текст одним способом.

Тепер нам потрібно зберегти проект. Проект – сукупність файлів різних форматів, з яких створюється програма. Про це ми детальніше поговоримо в одній з наступних лекцій, а поки запам'ятайте правило - кожну програму (проект), яку ви створюєте, потрібно зберігати в окремій папці. Якщо ви всі проекти станете зберігати в одну папку, то дуже скоро заплутаєтеся. Щоб зберегти проект, виконаємо наступні дії:

1. Вибираємо пункт меню «File - Save All» (зберегти все), або натискаємо гарячі клавіші <Shift+Ctrl+S>, або натисніть однойменну кнопку на панелі інструментів

2. Потім виходить вікно з пропозицією зберегти модуль - текстовий файл з вихідним кодом, що належить формі. Файл має розширення *.pas. Маємо на увазі, що кожен проект зберігається в окрему папку, тому спочатку клацнемо правою кнопкою миші по вільного місця вікна з папками, і виберемо команду

«Створити Папку». Дамо папці ім'я, наприклад, «01». Після створення папки відкриваємо її.

3. Далі в полі «Ім'я файлу» вкажемо вибраний модуль. Ім'я може бути будь-яким, але обов'язково латинськими символами. Ще ім'я модуля не повинно співпадати з назвою форми. Звичайно, імена форм і модулів прагнуть робити інформативними, тобто, по імені можна буде здогадатися, що це за файл. Оскільки це головна форма проекту, дамо їй ім'я «Main», і натиснемо кнопку «Зберегти».

4. Потім нам буде запропоновано дати ім'я проекту в цілому. Ім'я проекту буде співпадати з назвою виконуваного програмного файлу. Якщо ми хочемо, наприклад, отримати файл «hello.exe», то дамо проекту ім'я «hello». Натиснемо кнопку «Зберегти».

Далі нам необхідно скомпілювати програму, тобто перевести вихідний код у виконуваний exe-файл. Для цього ми можемо вибрати команду меню «Run - Run», або натисніть гарячу клавішу F9 або натиснути кнопку «Run» на панелі інструментів (на кнопці зображення зеленої стрілки, що вказує праворуч). В результаті, програма була не тільки скомпільовано, але і запущена. Якщо ви подивіться на системну рядок Delphi, то побачите напис «Delphi 7 - hello [Running]», а вікна інспектора об'єктів і дерева об'єктів зникли. Це говорить про те, що програма знаходиться у режимі виконання. Що виконується програма має точно такий же вигляд, як наша головна форма, тільки на формі відсутній точкова сітка, призначена для полегшення дизайну. Вікно отриманої програми містить всі стандартні кнопки Windows - програми. Клацнувши по червоному хрестику в правій верхній частині вікна, закрийте програму (але не Delphi), і ви побачите колишню форму.

Зверніть увагу, що властивості в *Об'єктному Інспекторі* належать виділеному в цей момент компоненту. Виділяються компоненти простим клацанням миші. Майте на увазі, що клацати потрібно один раз. Подвійний щиглик створить обробник подій - процедуру. Якщо ви помилково створите таким чином процедуру, то просто збережіть проект, нічого в неї не вписуючи -

при збереженні останні порожні процедури автоматично видаляються. Видаляти їх вручну не рекомендується.

*Спробуємо другий спосіб.* Зверніть увагу на Палітру компонентів. Поточною є вкладка Standard, і на ній знаходиться багато значків - компонентів. Коли ви наводите вказівник миші на будь-який компонент, через деякий час вискакує підказка з ім'ям компонента. Нам потрібен компонент Label, який представлений на вкладці у вигляді кнопки із зображенням жирної букви «A». Клацніть на цій кнопці, потім клацніть по вільного місця на формі, щоб вставити компонент. Краще, якщо ви розмістите його ближче до лівого верхнього краю форми. Компонент Label з'явився на формі. Цей компонент являє собою звичайний напис. Зараз він виділений, і містить напис за замовчуванням, – «Label1». Тепер об'єктний інспектор показує властивості цього компонента, а не форми. Label також має властивість Caption, яку ви можете змінити в Інспекторі об'єктів. Знайдіть це властивість, і замість «Label1» впишіть «Hello, world!». Текст у вікні Label змінився. Якщо вам не подобається місце, в якому опинився компонент, ви можете перетягнути його мишею на інше місце. Крім того, точне розташування компонента ви можете задати, якщо виділіть його, і будете натискати клавіші курсора, утримуючи натиснутою клавішу <Ctrl>. Тепер спробуйте ще одна властивість компонента Label - властивість Font (шрифт). Знайдіть це властивість в інспекторі об'єктів, і виберіть його. Праворуч з'явиться кнопочка з трьома точками, натисніть її. Відкриється стандартне вікно вибору шрифту. Тут ви можете вибрати назву шрифту, його розміри, зображення (наприклад, жирний курсив) і колір тексту. Поекспериментуйте з розміром компонента, його становищем і шрифтом. Майже всі компоненти, з якими нам доведеться мати справу, мають ці властивості, так що надалі вам буде легше освоювати новий компонент.

Знову збережете і натисніть кнопку Run (або <F9>). Переконайтеся, що напис з'явився на формі, після чого закрийте програму (але не Delphi) і поверніться до форми.

*Спробуємо третій*, трохи більш складний спосіб. Поки що ми створювали програму, не написав жодного рядка коду. Ми займалися тільки дизайном, всі інші труднощі Delphi взяла на себе. Тепер спробуємо вивести це ж повідомлення, як тільки користувач натисне кнопку на формі.

Для початку потрібно встановити на форму кнопку. Цей компонент також знаходиться на вкладці Standard панелі компонентів, і виглядає як кнопка з написом «ОК». При наведенні миші вискакує підказка «Button». Клацнувши по компоненту, клацніть потім з того місця на формі, де ви хотіли б бачити цю кнопку. Змінимо напис на кнопці. Переконайтеся, що кнопка виділена, і знайдіть в інспекторі об'єктів її властивість Caption. Замініть напис «Button1» на «Натисни мене!». Якщо напис уміщається на кнопку, ви можете розтягнути кнопку миші, або використовувати для цього клавіші керування курсором з натиснутою кнопкою <Shift>. Далі нам потрібно створити обробник натискання на кнопку. Обробник являє собою процедуру, в якій ми будемо писати наш код. Цей код буде виконуватися програмою всякий раз, коли користувач натисне на цю кнопку. Щоб створити цей зворотній виклик, слід двічі натиснути на кнопку на формі. Ви відразу потрапляєте в редактор коду і бачите, що процедура вже створена, курсор блимає в тому місці, де ми повинні ввести свій код. Поки що не будемо розбиратися, що тут до чого, а просто впишемо рядок:

```
ShowMessage('Hello, world!');
```

Повний текст процедури вийде такий:

```
Procedure TForm1.Button1Click(Sender: TObject);
Begin
  Showmessage('Hello, world!');
End;
```

Якщо у вас так і вийшло, збережете, відкомпілюйте його і запустити на виконання. При натисканні на кнопку буде з'являтися зазначена напис. Ми створили повноцінну програму, що виводить напис «Hello, world!» трьома різними способами, зазначивши при цьому лише один рядок вихідного коду! Отриманий файл hello.exe знаходиться в зазначеній вами папки

C:\ProgrammFiles\Borland\Delphi7\Projects\01

При збереженні проекту ви можете вказувати й інші папки, і проект буде збережений за вказаною адресою. Отриманий програмний файл hello.exe тепер ви можете поширювати.

**Змінні**

У будь-якій мові програмування доводиться використовувати змінні. При завантаженні програми, комп'ютер спочатку зчитує всі необхідні дані в оперативну пам'ять, після чого вже має можливість працювати з ними.

Кожна змінна має унікальне ім'я. Присвоюванням імен (ідентифікаторів) змінним займається програміст. Імена змінних в Delphi даються за певними правилами:

1. Ім'я змінної може містити будь-яку кількість англійських літери, цифри та символ підкреслення, інші символи неприпустимі.

2. Першим символом обов'язково повинна бути буква.

3. В Delphi немає різниці, які букви ви даєте змінним - великі або малі. Тобто, myperem, MyPerem, MYPEREM - це одна і та ж змінна.

Намагайтеся давати змінним осмислені імена і поєднуйте великі літери з маленькими для поділу на слова. Хороші приклади - MinZarplata або Glav_Param.

**Типи змінних**

Кожна змінна має свій тип. Тип змінної обов'язково потрібно вказувати, тому що різні типи змінних займають різний розмір. Створення змінної складається з двох етапів:

1. Оголошення змінної (вказуємо ім'я і тип змінної). Змінна оголошується в спеціальному розділі var (пізніше ми познайомимося з цим розділом).

2. Присвоєння змінній значення.

Оголошення змінної виглядає так:

```
var
Peremennaya1: Real;
Peremennaya2, Peremennaya3: Integer;
```

Як видно з прикладу, спочатку вказується ім'я змінної, потім, після двокрапки вказується тип змінної. Якщо потрібно оголосити кілька змінних одного типу, їх імена розділяються комами. У наведеному прикладі ми оголосили одну дійсну змінну типу Real і дві цілі змінні типу Integer.

Присвоювати значення змінних можна неодноразово. Змінна тому й називається так, що її значення в процесі роботи програми може змінюватися. Оператор присвоєння значення виглядає так:

:=

Приклади присвоєння значень змінних:

```
A := 10;
B := 20.35;
C := 'Це рядок';
D := True;
A := 3+5-1;
```

Механізм присвоювання значення працює наступним чином: спочатку розраховується значення правої частині команди, тобто, після знаку«:=». Потім результат цього значення записується у змінну. В останньому рядку прикладу ми використовували вираз «3+5-1». Спочатку отримує результат, у нашому випадку він дорівнює 7. Потім цей результат записується до змінної.

Надалі, ім'я змінної можна використовувати в різних виразах, наприклад:

```
A1 := 3;
A2 := A1 + 7;
A1 := A1 + 1;
```

У першому рядку ми записали в змінну число 3. Другий рядок містить вираз, результатом якого буде число 10. А ось третій рядок цікавіше. Як ви вважаєте, що бо записано в змінну A1? Якщо ваша відповідь 4, ви абсолютно праві: спочатку розраховується результат правої частині команди, де у змінній A1 ще старе значення, потім він записується в цю змінну, змінюючи її значення.

Таблиця 1.1 – Основні типи змінних:

| **Назва типу** | **Опис** | **Пояснення** |
|---|---|---|
| **Integer** | Ціле число | Змінна може вміщувати тільки цілі числа, як зі знаком так і без. |
| **Real** | Дійсне число | Змінна може приймати в якості значення цілих та дробових чисел, як зі знаком так і без. |
| **String** | Рядок | Змінна може зберігати любі символи та набори символів. В змінну String можно записати до 2 Гб символів. |
| **Boolean** | Логічний тип | Булева змінна, може бути або False (Брехня), або True (Істина). |

Насправді, типів змінних значно більше, і в міру ускладнення програм ми будемо вивчати ці типи. В таблиці представлені тільки основні типи.

**Рядок**

З цим типом змінних доводиться працювати досить часто. У прикладі вище ми вказували рядок ‘рядок’, а на минулій лекції мали справу з рядком ‘Hello, world!’. Ви вже помітили, що всі рядки повинні бути укладені в одинарні лапки. Розмір рядка практично не обмежений. Ви можете записати в рядок будь-який текстовий файл, все залежить від того, скільки місця є у Вас на диску, і який ОЗП у вас встановлений.

Вивчати новий тип зручніше відразу на практиці, тому запускайте Delphi. Автоматично повинен створити новий проект. Напишемо простеньку програму, в якій попрацюємо з рядком. Всі компоненти, які нам для цього знадобляться, знаходяться на вкладці Standard Панелі компонентів.

Помістіть на форму, один за іншим, наступні компоненти: Label, Edit і Button. З Label і Button ви вже знайомі з минулого програмі - Label це проста напис, а Button кнопка на формі. Компонент Edit являє собою поле для введення користувачем значення типу рядок.

Розтягніть компонент Edit, зробивши його приблизно удвічі довшим. Виділіть компонент Label (одним натисканням миші!), і в його властивості Caption замість «Label1» впишіть «Як твоє ім'я?».

Тепер виділіть компонент Edit, і видаліть «Edit1» з властивості Text, залишивши там порожнє поле. Властивість Text цього компонента містить той текст, який в даний момент знаходиться в полі вводу. Взагалі, з властивостями компонента можна поводитися, як із змінними. В більшість властивостей можна заносити значення не тільки в Інспекторі об'єктів, у момент розробки форми, але і під час виконання програми.

Виділіть кнопку, і в її властивості Caption напишіть «Натисни мене!». Для краси, перемістіть її в центр форми.

Змініть розмір форми, щоб на ній не було багато вільного місця. А також у властивості Caption форми напишіть «Привітання». Нагадаємо, щоб виділити форму, потрібно один раз клацнути мишею з будь-якого вільного місця форми. Якщо ви все зробили правильно, у вас повинна вийшов така форма:

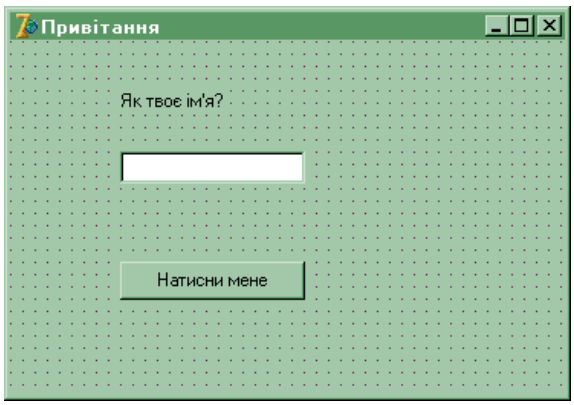



Рисунок 2.2 – Зовнішній вигляд форми

Тепер створимо обробник подій для кнопки. Для цього двічі клацніть по кнопці, яку ви розмістили на формі. Відразу ж ви потрапляєте в Редактор коду, і курсор блимає між рядками begin і end. Begin - це початок процедури, після нього крапка з комою не ставиться. End кінець процедури, після цього оператора крапка з комою обов'язкова. Якщо ви придивитеся, то в останньому рядку редактора коду побачите end з точкою - це кінець програми. Змінні вказуються (описуються) перед початком процедури. Отже, рядок begin

потрібно буде опустити, а перед нею вписати розділ змінних var, і вказати змінну s. Процедура повинна виглядати так:

```
procedure TForm1.Button1Click (Sender: TObject);
 var
s: String;
 begin
s: = 'Привіт,' + Edit1.Text + '!';
ShowMessage (s); end;
```

Зверніть увагу, що коли ви поставите крапку після назви компонента Edit1, вийде список доступних властивостей, методів і подій. Коли ми впишемо першу літеру «T», список сортується - в ньому залишаться тільки команди на літеру «T», причому Delphi оцінює контекст, і залишає у списку тільки ті команди, які в цьому контексті можуть бути використані. У нашому випадку, це рядок. Властивість Text, який залишиться в списку, може бути використано в якості рядки:

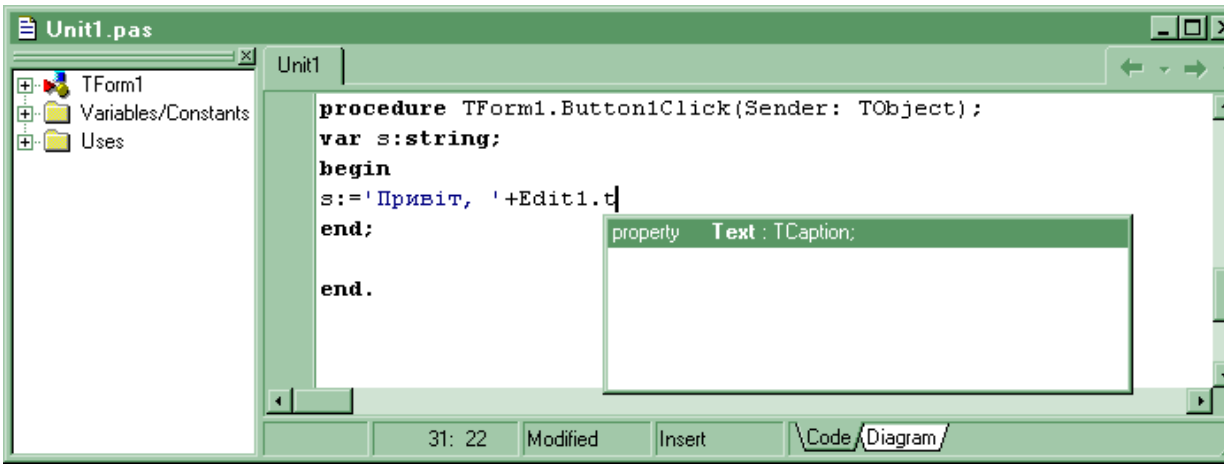



Рисунок 2.3 – Вибір властивості зі списку-підказки.

Тепер немає потреби вводити «Text» самим – цю властивість виділено в списку, і нам достатньо натиснути <Enter>, щоб вставити її в код. На майбутнє запам'ятайте: завжди року ви ставите крапку після назви компонента, дочекайтеся такого списку. Справа не тільки в тому, що так легше вводити код. Якщо список так і не з'явився, значить, ви допустили помилку. Можливо, у назві компонента, або навіть у попередній рядку. Тепер збережіть проект в

окрему папку, модуль, як зазвичай, назвіть Main, а проект можете назвати, наприклад, Privet. Після збереження зберіть його і подивіться, яке повідомлення буде виходити при різних рядках, які введе користувач у полі вводу Edit1.

Справедливості заради варто відзначити, що в даному прикладі зовсім необов'язково було використовувати змінну - ми це зробили лише в навчальних цілях. Такого ж результату можна досягти, якщо використовувати вираз

'Привіт, ' + Edit1.Text + '!'

прямо в команді *ShowMessage*:

```
procedure TForm1.Button1Click(Sender: TObject); begin
ShowMessage('Привет, ' + Edit1.Text + '!'); end;
```

## 2.2 Компоненти Edit, Label, Memo. Процедури й функції користувача

Кожен користувач, що працює з Windows, хоч раз та використав простий текстовий редактор Блокнот. Зараз ми створимо трохи спрощену версію Блокнота, яка дозволяє вводити текст, зберігати його на диск, і завантажувати з диска. Для цього нам потрібно створити новий проект. Якщо зараз у вас вже відкрито якийсь проект, виберіть команду меню «File - Close All (Файл – Закрити всі)», потім «File - New - Application (Файл - Нове - Додаток)».

У нас є нова форма, і поки більше нічого немає. Відразу змінимо деякі властивості форми. У властивості Caption впишіть «Мій блокнот» (природно, без лапок). Властивість Name змінимо на «fMain».

*Порада:* щоб легше орієнтуватися в назвах модулів і форм, краще відразу виробити правила:

1. Перед назвою форми будемо вказувати маленьку букву f, щоб показати, що це саме форма.

2. Імена форм будемо підбирати інформативні, щоб було зрозуміло, з якою формою ми маємо справу.

3. Модулі форм будемо називати також, але без букви f.

4. Головну форму завжди будемо називати fMain, а модуль, відповідно, Main.

Ці рекомендації не є обов'язковими, але надалі, коли підуть проекти з безліччю форм, вони допоможуть вам орієнтуватися в назвах. Можете виробити власні правила.

Збережемо проект в окрему папку. Модуль називаємо Main, а проектом даємо ім'я MyNotebook.

Тепер познайомимося з новим потужним компонентом Memo, який знаходиться на вкладці Standard і призначений для введення користувачем багаторядкового тексту.

Встановіть цей компонент на форму, у верхню ліву частину форми, і потягніть його за формою, залишивши внизу небагато місця. Вниз встановіть, одну поряд з іншого, три кнопки Button. У вас повинно вийти щось на зразок цього:

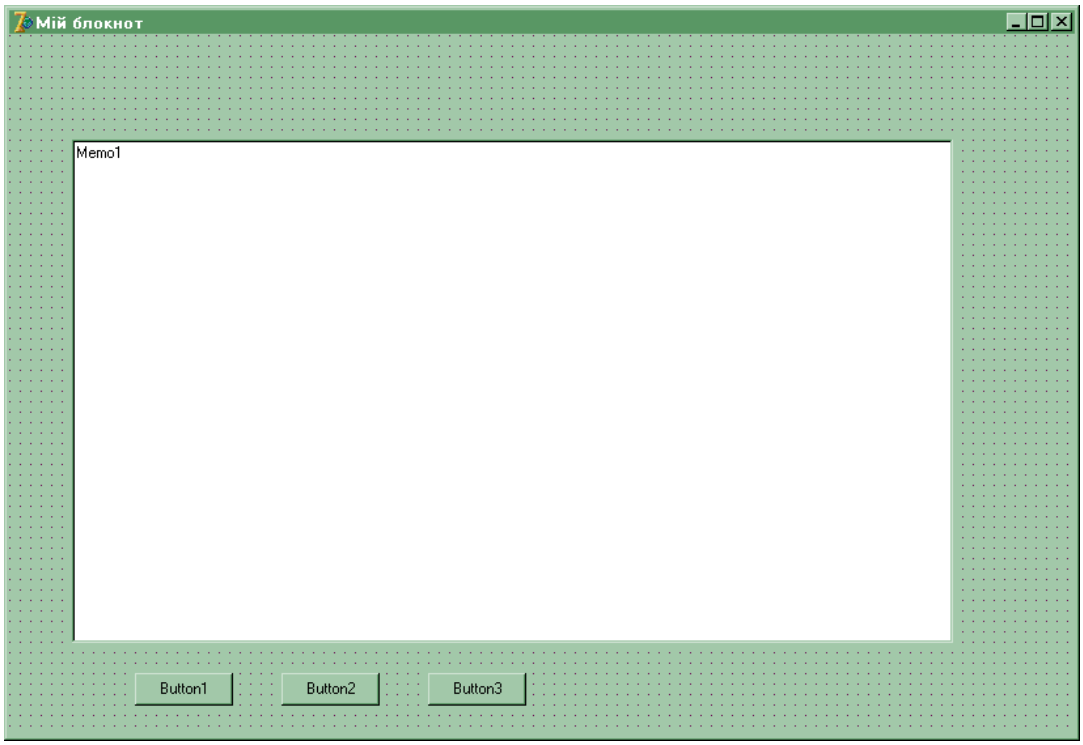



Рисунок 2.4 – Заготівка для редактора текстів

Тепер виділимо першу кнопку, і властивості Caption цієї кнопки напишемо «Зберегти». На другий кнопці напишемо «Завантажити», на третьому - «Очистити».

Виділимо компонент Memo, який являє собою велике біле поле. У компонента є одна цікава властивість Lines, яке містить рядки тексту, набраного в компоненті. Докладніше з цією властивістю ми познайомимося пізніше, а поки виділимо його в Інспекторі об'єктів, натискаємо на кнопку з трьома точками праворуч від властивості і тим самим відкриємо редактор тексту. Тут можна набрати текст, який буде виведений в компонент «за замовчуванням», при кожному запуску програми. Нам просто потрібно, щоб тексту ніякого не було, тому видаліть всі рядки, які там є і натисніть «ОК». Компонент Memo очистився.

Це ще не все. Нам потрібно, щоб при введенні текст автоматично переносився на інший рядок, а користувач мав можливість його перегортати. Знайдіть властивість ScrollBars (компонент Memo повинен бути вибраним), це властивість відповідає за наявність смуг прокрутки. Виберіть значення ssVertical, щоб з'явилася вертикальна смуга прокрутки.

З інтерфейсом покінчено, залишилося ввести код, який буде виконуватися програмою. Двічі натиснемо на першу кнопку, створимо обробник подій для кнопки «Зберегти». У місці, де блимає курсор, впишемо тільки один рядок:

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

Метод SaveToFile() властивості Lines компонента Memo зберігає весь текст у вказаний файл. Якщо ви не вказуєте шлях до файлу за замовчуванням файл буде створено там, звідки була запущена програма. Для другої кнопки напишемо два рядки:

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

Розглянемо ці рядки. Що, якщо користувач натисне цю кнопку до того, як що-небудь збереже в файл? Файлу то ще немає! Перший рядок як раз виконує перевірку на існування файлу. Якщо файлу немає, то другий рядок виконуватися не буде. Якщо він є, тоді другий рядок вважає текст з файлу в

компонент Memo. З умовними операторами ми будемо знайомити пізніше, тоді сенс першого рядка буде зрозуміліше.

Для третьої кнопки код ще простіше:

Memo1.Clear;

Ця команда очищає компонент Memo від тексту. Ось, власне, і вся програма. Збережіть її і зберіть, перевірте, як вона працює. Оцініть легкість програмування - для створення повноцінного редактора текстів ми написали всього 4 рядки коду!

Поекспериментуйте з властивістю Font (Шрифт) компонента Memo, подивіться, яким чином буде змінюватися шрифт тексту.

**Цілі і дійсні типи. Процедури та функції. Цілі числа**

Програмістам суцільно і поруч доводиться використовувати дані цілого типу. Ціле число - це число, яке не має коми. Число може бути беззнаковим (додатнім), і зі знаком мінус (від'ємним). Приклади:

*1*
*-12*
*1234567*

У минулій лекції ми згадали лише тип Integer, як основний тип цілих чисел. Дійсно, цей тип доводиться використовувати найчастіше, однак існують і інші типи цілих чисел. В таблиці 2.1 представлені цілі типи:

Таблиця 2.1. Цілі типи даних

| **Тип** | **Діапазон можливих значень** | **Розмір пам'яті під** | **Примітка** |
|---|---|---|---|
| ***Integer*** | -2147483648 .. | 4 байта | Знакове |
| ***Cardinal*** | 0 .. 4294967295 | 4 байта | Без знаку |
| ***Shortint*** | -128 .. 127 | 1 байт | Знакове |
| ***Smallint*** | -32768 .. 32767 | 2 байта | Знакове |
| ***Longint*** | -2147483648 .. | 4 байта | Знакове |
| ***Int64*** | $-2^{63}2^{63}-1$ | 8 байт | Знакове |
| ***Byte*** | 0 .. 255 | 1 байт | Без знаку |
| ***Word*** | 0 .. 65535 | 2 байта | Без знаку |

**Дійсні числа**

Дійсні числа – це числа з комою, після якої йдуть десяткові значення. Ще говорять, що вони мають плаваючу точку (запам'ятайте це визначення, воно буде часто зустрічатися). Деякі початківці програмісти вважають, що краще такий тип змінних використовувати завжди, навіть при обробці цілих чисел. Це велика помилка! Операції над числами з плаваючою точкою віднімають у процесора набагато більше часу, і вимагають більше пам'яті. Комп'ютер сприймає дійсне число, як два цілих, і робить подвійну роботу при обробці чисел до коми, і після неї. Однак іноді буває необхідно використовувати саме такий тип даних. Наприклад, якщо потрібно поділити одне ціле на інше. Добре, якщо це буде «4/2», результат теж буде цілим - 2. А якщо «4/3»? Тоді результатом буде 1,3333... і вже тут без дійсного числа не обійтися! Адже ми заздалегідь не знаємо, які числа буде ділити користувач, тому краще відразу мати на увазі, що результат може бути не цілим числом.

Як і цілі, дійсні числа мають кілька типів. У таблиці 3.2 вони представлені:

Таблиця 2.2. Дійсні типи даних

| **Тип** | **Діапазон возможных значень** | **Значущих цифр максимально** | **Розмір в байтах** |
|---|---|---|---|
| ***Real48*** | $2.9 * 10^{-39} .. 1.7 * 10^{38}$ | 11-12 | 6 |
| ***Real*** | $5.0 * 10^{-324} .. 1.7 * 10^{308}$ | 15-16 | 8 |
| ***Single*** | $1.5 * 10^{-45} .. 3.4 * 10^{38}$ | 7-8 | 4 |
| ***Double*** | $5.0 * 10^{-324} .. 1.7 * 10^{308}$ | 15-16 | 8 |
| ***Extended*** | $3.6 * 10^{-4951} .. 1.1 * 10^{4932}$ | 19-20 | 10 |
| ***Comp*** | $-2^{63}+1 .. 2^{63}-1$ | 19-20 | 8 |
| ***Currency*** | -922337203685477.5808 .. 922337203685477.5807 | 19-20 | 8 |

Третій стовпчик таблиці вказує кількість максимально значущих цифр. Цифри, які виходять за цю межу, будуть ігноруватися. Тут важливо пам'ятати, що дійсні числа не рівні цілим. Тобто, число 3,0 не буде дорівнювати 3. Щоб порівняти обидва ці числа, доведеться округлити дійсне число.

Вивчимо цілі і дійсні типи на практиці. Для цього створимо просту програму, яка ділить одне ціле число на інше. Результат буде виводитися, як дійсне число.

Відкрийте Delphi, створіть новий проект. На форму потрібно помістити три компоненти Label, три компоненти Edit і одну кнопку, щоб вийшла така картина:

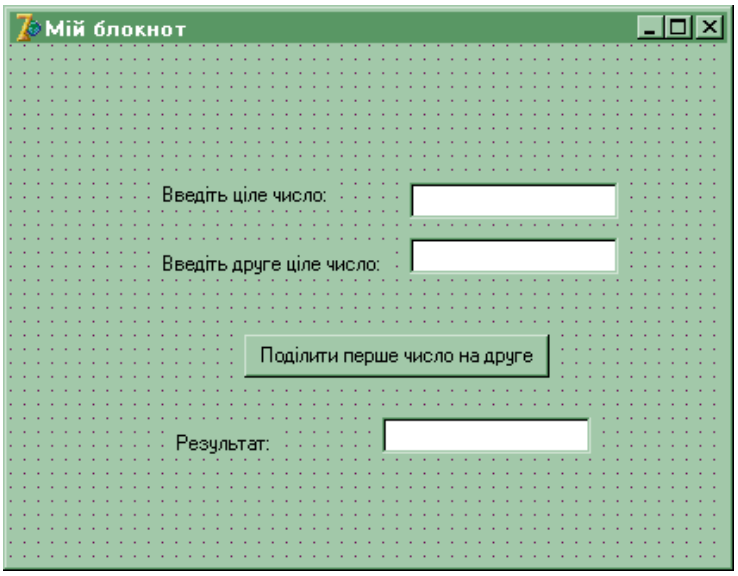



Рисунок 2.5 – Зовнішній вигляд форми

*Порада:* щоб виконати однакову операцію над декількома компонентами відразу, їх можна виділити один за іншим, утримуючи клавішу <Shift>. Наприклад, якщо таким чином виділити всі три компоненти Edit, то потім можна разом очистити їх властивість Text.

Збережіть проект під назвою MyCalc. Потім двічі клацніть по кнопці, щоб створити обробник натискання на кнопку.

Перед begin процедури слід створити розділ var, і оголосити там три змінних:

```
var
```

Perem1, Perem2 : Integer; Perem3 : Double;

Потім повернемося в тіло процедури (між командами begin і end), і присвоїмо цілим змінним введені користувачем значення. Тут потрібно зрозуміти одну важливу річ. Користувач буде вводити значення компоненти Edit, і там вони будуть зберігатися у властивості Text у вигляді текстового типу даних. Рядок не можна буде присвоїти змінній будь-якого іншого типу даних, присвоєння Perem1 := Edit1.Text; помилкове присвоєння – несумісність типів даних буде помилковим. Різниця досить істотна: навіть якщо користувач вводить, здавалося б, ціле число, наприклад, 123, то комп'ютер бачить рядок символів, а зовсім не число: ‘123’

Вихід - перетворити один тип даних в інший, в нашому випадку, рядок в цілий тип. Перетворенням типів доводиться займатися суцільно і поруч, у міру вивчення матеріалу ми будемо знайомитися з різними способами перетворення. Перетворити рядок в цілий тип можна за допомогою функції StrToInt(‘String’).

В якості параметра (в дужках) вказується рядок. Функція перетворює її в ціле число і поверне його як результат. Приклади використання функції (ці приклади не потрібно вводити в редактор коду):

```
var
s: String; i: Integer;
begin
s := '1234';
i := StrToInt(s); //параметр - символьна змінна
i := StrToInt('123456'); //параметр – рядок
i := StrToInt(Edit1.Text); //параметр - властивість Text
//компонента Edit, що має текстовий тип
end;
```

Як видно з прикладу, є маса можливостей передати функцію рядок. У першому випадку перетворення ми передаємо строкову змінну s, в якій зберігається рядок ‘1234’. Функція перетворює цей рядок в ціле число, і в результаті в змінну і потрапить вже число 1234.

У другому випадку ми передаємо безпосередньо рядок ‘123456’, а в змінну і потрапляє перетворене з цього рядка число. У третьому випадку ми в якості параметра передаємо той текст, який користувач ввів у полі вводу Edit1. Тут слід зробити застереження. Функція спрацює правильно, якщо користувач ввів туди дійсно ціле число. Інакше виникне помилка. Користувач - особистість непередбачувана, тому программист в таких випадках перед перетворенням типів завжди робить перевірку – чи дійсно в полі є тільки цифри від 0 до 9? Чи немає там випадково букви або комою? Таку перевірку ми навчимося робити пізніше. Поки що доведеться самим стежити, щоб в цих полях введення були тільки цілі числа.

Повернемося до програми. Відразу після begin присвоюємо цілим змінним значення, які ввів користувач:

```
Perem1 :=StrToInt(Edit1.Text);
Perem2 := StrToInt(Edit2.Text);
```

У третю, дійсну змінну, ми повинні записати результат ділення першого числа на друге. Тут може критися ще один «підводний камінь» – що, якщо у друге поле користувач ввів число 0? Відомо, що на нуль ділити не можна. Якщо ж ми спробуємо це зробити, то комп'ютер, в кращому разі, зависне. Тут знову доведеться робити перевірку на правильність введених даних, ставити, як кажуть, «захист від дурнів». Докладніше про такі перевірки ми поговоримо на наступних лекціях, коли вивчимо умовні конструкції. А поки що просто наберіть цей код після попередніх двох рядків присвоєння:

```
    {захист від дурнів:}
    IfPerem2 = 0 then begin //якщо це нуль, то:
    ShowMessage('на нуль ділити не можна!'); //виводить повідомлення
    Edit3.Text := '0'; //як результат записуємо нуль
    end
else
    begin //інакше:
    Perem3 := Perem1 / Perem2; //ділимо
```

```
Edit3.Text := FloatToStr(Perem3); //перетворимо дійсний тип у рядок
//і записуємо результат
end;
```

Тут слід звернути увагу на передостанній рядок. Функція FloatToStr() в якості параметра приймає дійсне число, і повертає це ж число у вигляді рядка. Наприклад, в результаті перетворення:

```
s := FloatToStr(123.45);
```

змінної s буде присвоєно рядок ‘123.45’, яку потім вже можна буде вивести користувачеві в якості результату.

У нашому прикладі ми результат ділення двох цілих чисел перетворимо в рядок і виводимо його в поле Edit3. Заради справедливості слід зауважити, що в якості параметра можна передавати не тільки значення, але й вирази. Наприклад, якщо записати:

```
Edit3.Text := FloatToStr(Perem1 / Perem2);
```

то потреба у використанні дійсної змінної Perem3 відпадає. Спробуйте, як працюють обидва варіанти.

**Процедури**

Іноді буває необхідно виконувати частину коду неодноразово. Цей самий код виносять в окрему підпрограму – процедуру. *Процедура* – підпрограма, яка виконує якісь дії, і яку можна викликати з іншого місця програми. Після виконання процедури виконання програми продовжується з того місця, звідки вона була викликана. Процедура живе самостійним життям, і в будь-який момент її можна викликати, щоб виконати якісь дії. Щоб процедуру можна було викликати з програми, її необхідно оголосити вище того місця, де ми будемо її викликати. Синтаксис процедури такий:

```
procedureNameProc(Param : Тип);
var //оголошення змінних(необов'язково)
begin
//тіло процедури
end;
```

Викликати таку процедуру можна, просто вказавши її ім'я. Перевіримо це на практиці. Повернемося до нашого проекту, і вище процедури обробки кнопки створимо таку процедуру:

```
procedure Soobshenie;
begin
ShowMessage('Помилка! На нуль ділити не можна!');
end;
```

У цій процедурі ми не використовуємо змінних, тому розділ var відсутній. Все, що робить наша процедура – виводить повідомлення про те, що на нуль ділити не можна. Зверніть увагу, що якщо немає вхідних параметрів, то дужки вказувати необов’язково. Тепер знову перейдемо до процедури обробки натискання кнопки, і замість виводу повідомлення, що на нуль ділити не можна, проведемо виклик процедури:

```
Soobshenie;
```

Тепер збережіть проект, зберіть його і подивіться, що вийшло. Не забудьте, що вводити в Edit1 і Edit2 можна тільки цифри від 0 до 9. Тепер про параметри. Параметри – це вхідні дані. Тобто, ми можемо викликати процедуру і задати їй потрібні дані. Параметри процедури записуються в круглих дужках з зазначенням типу. Якщо параметрів немає, дужки можна не ставити. Приклад процедури з параметрами:

```
procedure Primer(a,b: Integer);
begin
a*b;
 end;
```

Зверніть увагу, що обов'язково потрібно вказувати тип параметрів. Тепер ми можемо викликати цю процедуру, вказавши їй, які цифри потрібно помножити. Приклади виклику процедури:

```
Primer(10, 20); //передаємо цілі числа
Primer(a, 100); //передаємо змінну a з цілим числом, і ціле число
Primer(c, d); //передаємо дві змінних з цілим числом
```

Відразу слід сказати про області видимості змінних. Бувають змінні глобальні й локальні. Глобальні змінні видно у всій програмі, ми їх будемо використовувати пізніше. А локальні змінні створюються всередині процедури, в розділі var, і видно тільки в цій процедурі. Локальні змінні створюються в пам'яті в той час, коли процедура починає роботу, і знищуються, коли процедура закінчила роботу. Таким чином, ми можемо зробити дві або більше процедур, і вказати в них змінні з однаковим ім'ям. Це будуть різні змінні, і вони не будуть заважати один одному.

**Функції**

Функції – це такі ж підпрограми, як і процедури. Відміну від функцій процедур в тому, що вони не просто виконують якісь дії і розрахунки, але і можуть повертати результат певного типу. Оскільки вони повертають результат, необхідно вказати тип цього результату. Синтаксис функції такий:

```
function NameFunc(Param:Тип) :<Тип_значення,що буде повернено>;
var //оголошення змінних (необов'язково)
  begin
//тіло процедури
  Result := результат обчислень;
  end;
```

Тут слід звернути увагу на два моменти: після імені функції і параметрів у круглих дужках, після двокрапки, вказується тип повертає значення. Крім того, у кожної функції за умовчанням є мінлива Result, яка має той же тип, що і тип повертає значення. Цю змінну спеціально оголошувати не потрібно, вона вже готова до роботи. На відміну від інших мов, в Delphi цієї змінної можна привласнювати значення неодноразово. Результатом буде останнє присвоєне значення.

Є ще один спосіб повернути функції результат обчислень: використовувати змінну з таким же ім'ям, як і ім'я функції. Цю змінну теж оголошувати не потрібно. У нашому прикладі, рядок

```
Result := результат обчислень;
```

буде повністю ідентичною рядку

NameFunc := результат обчислень;

Який зі способів використовувати - вирішуйте самі, обидва способи правильні. Знову повернемося до нашої програми, і для закріплення знань додамо в неї функцію. Функція також повинна бути описана вище того місця, де ми будемо її викликати. Можете створити її між нашою процедурою і процедурою натискання на кнопку.

```
function Delenie(a,b : Integer): Real;
begin
Result :=a/b;
end;
```

Тепер замінимо той рядок, де в третю змінну записується результат ділення перших двох, на виклик функції і передачу їй цих двох чисел:

```
Perem3 := Delenie(Perem1, Perem2);
```

Звичайно, ці приклади примітивні. Реально функції і процедури виконують набагато більш важливі речі, ніж поділ одного на інше. З часом наші програми будуть містити безліч таких функцій і процедур.

**Події**

В Delphi подія означає, що якийсь компонент, якому ми призначили подія, змінився. Подія - це процедура, якій передається управління у випадку, якщо відбулися запрограмовані зміни.

Події можуть бути різними – зміна тексту в полі Edit, натискання кнопки миші або клавіші, або просто миша опинилася над компонентом.

Давайте будемо наш приклад, введемо в нього подія. Виділіть компонент Edit1. Зараз ми поставимо йому подія OnChange, яке відбувається кожного разу при зміні тексту в цьому компоненті. Давайте уявимо собі користувача, що працює з нашою програмою. Йому буде приємно, якщо він почне змінювати текст в першому полі, а інші поля автоматично очистяться, щоб бути готовими для нових розрахунків!

Виділимо компонент Edit1.Перейдемо в інспекторі об'єктів на вкладку Events (події).

Двічі клацнемо за події OnChange (Зміна).

Створиться процедура обробки події, і ми потрапимо в редактор коду. Там ми впишемо два рядки:

```
Edit2.Clear;
Edit3.Clear;
```

Тепер вставимо ще одну «захист від дурнів». Адже третє поле потрібно тільки для результату? А раптом користувач почне там вводити дані? Нічого Страшного не відбудеться, у розрахунках це поле не бере, але все одно неприємно, коли твою програму використовують неправильно. Виділіть компонент Edit3.

На вкладці Properties (властивості) знайдіть властивість ReadOnly (лише для читання), і замість False (брехня), поставте True (істина). Все, тепер користувач не зможе вводити дані в це поле, тільки програма зможе виводити в нього результат.

Збережіть, виконайте Run і переконаєтеся в цьому.

### 2.3 Компонент Діаграмма (TChart)

Дуже потужний і багатий можливостями компонент. Він дозволяє будувати двох- і тривимірні діаграми на основі різних даних, є спадкоємцем класу TPanel і успадковує всі властивості панелі.

Створити діаграму можна двома способами: візуально за допомогою Майстра (без програмування) і безпосередньо засобами Паскаля.

Початок роботи. Майстер запускається командою File> New> Business> - TeeChart Wizard (Файл> Створити> Ділові> Майстер діаграм), після чого розробнику треба виконан ¬ нить ряд уточнень. Спочатку вибирається джерело даних. Нехай він не розташований у файлі, а генерується програмою - перемикач Non Database Chart (Не на основі бази даних). Потім вибирається

зовнішній вигляд діаграми. Вона може бути двовимірної або тривимірної що визначається перемикачем 2D/3D.

На наступному етапі роботи Майстри прапорець Show Legend (Відображати легенду) визначаючи наявність легенди - додаткової панелі, на якій вказується відповідність кольорів частин діаграми зазначеним значенням. Прапорець Show Marks включає невеликі жовті підказки у кожної з частин діаграми.

На цьому створення діаграми закінчується. Після клацання на кнопці Finish (Готово) в проектувальник форм з'явиться нова форма, на якій буде розташовано об'єкт Chartl. Він заповнений якимось набором випадково згенерованих значень

Налаштування діаграми виконується за допомогою редактора, який викликається подвійним клацанням на об'єкті Chartl.

Параметри відображення діаграми у вікні визначаються на вкладці Chart (Діаграма), що складається у свою чергу з набору додаткових панелей.

Панель Series (Ряд даних) дуже важлива. Вона дозволяє об'єднувати декілька діаграм на одному графіку за допомогою кнопки Add (Додати). При цьому над значеннями рядів даних можна виконувати різні операції, зада ¬ ваемие на вкладці Functions (Функції): додавання (Add), віднімання (Subtract), множення (Multiply), ділення (Divide), взяття найбільшого (High), найменшого (Low ) або середнього (Average) значення

Панель General (Загальні) містить елементи керування для:

• експорту зображення у файл - кнопка Export (Експортувати,);

• установки (у відсотках) зсуву кордонів зображення по відношенню до кордонів об'єкта - поля Margins (Поля);

• масштабування - панель Zoom (Масштаб);

• прокрутки - панель Allow Scroll (Дозволити прокрутку).

Засоби панелі Axis (Oci) відповідають за все, що стосується визначення коорди-кімнатної осей, їх масштабу, заголовків, кроку пунктирною сітки і так далі.

Панель Titles (Заголовки) містить засоби для. оформлення заголовка.

Панель Legend (Легенда) використовується при оформленні зовнішнього вигляду і вмісту легенди.

Засоби панелі Panel (Панель) описують форму і візуальне уявлення панелі-основи, на якій розташована діаграма.

Панель Pages (Сторінки) служить для розділення діаграми на сторінки. Збільшуючи кількість точок на сторінці за допомогою поля Points per Page (Точки на сторінку), можна підібрати оптимальне співвідношення між наочністю діаграм і розумним числом сторінок.

Панель Walls (Межі) дозволяє задати колір і розміри меж діаграми.

Панель 3D описує просторове уявлення тривимірних діаграм. За допомогою декількох движків проектовану діаграму можна обертати і масштабувати.

На вкладці Series (Ряди даних) в редакторі задаються конкретні параметри оформлення кожного ряду даних (кожного графіка, доданого за допомогою вкладки Chart). Вибір поточного ряду даних проводиться за допомогою списку Area (Область).

Тут найбільш важлива панель Data Source (Джерело даних). З її допомогою можна задати для ряду випадкові значення (Random Values), відмовитися від генерації значень (No Data) або сформувати значення поточного ряду даних як результат застосування деякої функції (список, що розкривається Function) до значень обраних рядів даних. Вибір рядів даних – занесення в список Selected Series (Вибрані ряди) здійснюється за допомогою кнопки>.

Програмна робота з діаграмами. Розглянемо приклад створення тривимірної діаграми і заповнення її значеннями безпосередньо з програми.

На формі Forml розмістимо компонент TChart і викличемо редактор. Це можна зробити також з контекстного меню об'єкта вибором пункту Edit Chart (Змінити діаграму).

На панелі Series (Ряд даних) вкладки Chart (Діаграма) клацніть на кнопці Add (Додати) і виберіть підходяще тривимірне представлення, наприклад Point (Точковий). На формі з'явиться діаграма, заповнена випадковими даними. На панелі Titles (Заголовки) треба вказати відповідну назву діаграми та закрити редактор.

Діаграма зв'язується з програмним кодом дуже просто. Більшість налаштувань, що мають відношення до оформлення діаграми, формуються в редакторі, а в програмі (в розділі класу TForml, де розташовуються створювані в Проектування-щіке елементи управління) повинен з'явитися новий об'єкт - мінлива Seriesl типу TPointSeries. Вона описує послідовність значень, які будуть відображатися на діаграмі. Всю іншу роботу система Delphi 7 бере на себе – дуже зручний і простий підхід.

Розглянемо основні властивості і методи класу TPointSeries (він є спадником базового класу TChartSeries, який служить основою для всіх класів, що описують вміст конкретних типів діаграм). Розробнику потрібні такі можливості, як додавання і видалення точки, зміна деякого значення, очищення всіх точок, отримання загального числа точок і доступ до їх поточним значенням.

Нехай є діаграма типу Point (точкове уявлення) на якій повинні розташовуватися умовні значення результатів двох експериментів («Експеримент А» і «Експеримент Б»). Ці значення вводяться за допомогою двох текстових полів, для їх редагування використовується клацання миші на точці діаграми. Потрібні також можливості видалення точки і очищення поточного графіка.

Так як потрібно виводити результати двох експериментів (два ряди значень), треба додати до поточної діаграмі ще один ряд. У редакторі діаграми на панелі Chart> Series (Діаграма> Ряд даних) клацніть на кнопці Add (Додати) і виберіть вид представлення Point (Точковий).

Щоб підпис «Експеримент ...» під крайньою лівою точкою діаграми цілком містилася на панелі Chartl, можна трохи зрушити ліву границю області

діаграми вправо. Для цього на панелі редактора Chart> General (Діаграма> Загальні) можна задати значення 5% в лівому полі на панелі Margins (Поля).

Додавання нової точки до серії виконується за допомогою методу Add, заголовок якого виглядає наступним чином.

```
function AddXY (Const AXValue, AYValue: Double;
Const AXLabel: String;
AColor: TColor): Longint;
```

Додана точка задається параметрами AXValue і AYValue. Параметри AXLabel і AColor - необов'язкові. Перший описує довільну назву групи, до якої буде належати точка, другий - колір цієї групи. У нашому випадку виберемо червоний колір для точок експерименту А, а цвіт точок експерименту Б зробимо синім. Функція повертає позицію (номер) нової точки у властивості XValues або YValues (масиви значень) залежно від того, за яким вимірюванню додається на діаграму точка.

Оброблювач клацання на кнопці Експеримент А (Buttonl) запишеться наступним чином.

```
procedure TForml.ButtonlClick (Sender: TObject);
begin
Seriesl.AddXY (StrToFloat (Editl.Text),StrToFloat (Edit2.Text), 'Експеримент А', clRed);
end;
```

Оброблювач клацання на кнопці Експеримент Б (Button2) буде виглядати так.

```
procedure TForm2.ButtonlClick (Sender: TObject);
begin
Series2.AddXY (StrToFloat (Editl.Text),StrToFloat (Edit2.Text), 'Експеримент Б', clBlue);
end;
```

Тепер можна запустити програму, ввести в поля значення і додати "результати експерименту» на діаграму.

Щоб видалити раніше введену точку або змінити її значення, треба попередньо визначити її номер в масиві Values. Для цього по кожному ряду даних (об'єкти Seriesl і Series2) формується обробник події onclickPointer. Його заголовок виглядає наступним чином.

procedure SeriesClickPointer (Sender: TCustomSeries; Valuelndex: Longint; X, Y: Integer);

Найбільш важливий параметр Valuelndex містить номер найближчої точки ряду, біля якої на діаграмі був виконаний клацання. X і Y - це координати точки клацання.

Помістимо на форму новий елемент - прапорець Режим видалення (назвемо його DeleteBox). Коли він встановлений, обирані точки будуть видалятися (після уточнюючого запиту). В іншому випадку поточна точка буде коригуватися у відповідності зі значеннями, вказаними в полях введення.

Видалення елемента з ряду даних здійснюється за допомогою методу Delete, що має єдиний параметр - номер елемента. Зміна поточного значення і положення на діаграмі виконується простим зміною вмісту відповідних елементів масивів ValueX і ValueY. Щоб зроблені зміни відобразилися на діаграмі, треба викликати метод Repaint (нечитабельний) для відповідного ряду даних.

Щоб не дублювати однаковий текст обробників клацання на двох рядах точок, додамо в клас TForml метод, який буде отримувати в якості параметрів номер ряду даних та індекс точки, після чого виконувати всі необхідні дії. Введемо в частину public класу TForml заголовок такої процедури.

procedure SerieClick (SNum: Integer; Index: Longint);

Тепер досить встановити на ім'я даного методу покажчик миші і вибрати в контекстному меню пункт Complete class at cursor (Завершити реалізацію класу).

У частині реалізації модуля відразу з'явиться порожня реалізація даної процедури. У неї треба додати перевірку стану прапорця DeleteBox і в

залежності від цього стану виконати або видалення точки, або коригування її значення (перемальовування діаграми здійсниться автоматично).

```
procedure TForml.SerieClick (SNum: Integer; Index: Integer);
begin
if DeleteBox.Checked then
begin
if SNum = 1 then Seriesl.Delete (Index)
else Series2.Delete (Index)
end else
begin
if SNum = 1 then
begin
Seriesl.XValues [Index]: = StrToFloat (Editl.Text);
Seriesl.YValues [Index]: = StrToFloat (Edit2.Text);
Seriesl.Repaint;
end else
begin
Series2.XValues[Index]: = StrToFloat (Editl.Text);
Series2.YValues[Index]: = StrToFloat (Edit2.Text);
Series2.Repaint;
end
end
end;
```

Тоді обробники клацання на точках кожного ряду даних запишуться таким чином.

```
procedure TForml.SerieslClickPointer (Sender: TCustomSeries;
Valuelndex, X, Y: Integer);
begin
SerieClick (1, Valuelndex);
end;
```

```
procedure TForml.Series2ClickPointer (Sender: TCustomSeries;
Valuelndex, X, Y: Integer);
begin
SerieClick (2, Valuelndex);
end;
```

Тепер програма дозволяє за допомогою клацань видаляти зайві точки і коректувати положення точок, введених раніше.

Для видалення всіх значень в ряду даних служить метод Clear.

```
Seriesl.Clear;
```

### 2.4 Головне меню

Будь-яка більш-менш серйозна програма має власне меню. Прийшла пора познайомитися з цим компонентом. Для прикладу знову завантажимо наш редактор текстів. Виділіть всі кнопки на формі і видаліть їх. Також видаліть і панель, на якій ці кнопки були. Потім нам потрібно видалити всі процедури обробки цих кнопок, але тут потрібно проявити обережність - не можна просто взяти і видалити процедуру. Кожна згенерований процедура прописана в коді і вище. Щоб без помилок видалити всі непотрібні останні процедури, слід просто видалити з них той код, який ми писали самі, залишивши «порожню» процедуру - ім'я процедури і рядки begin .. end:

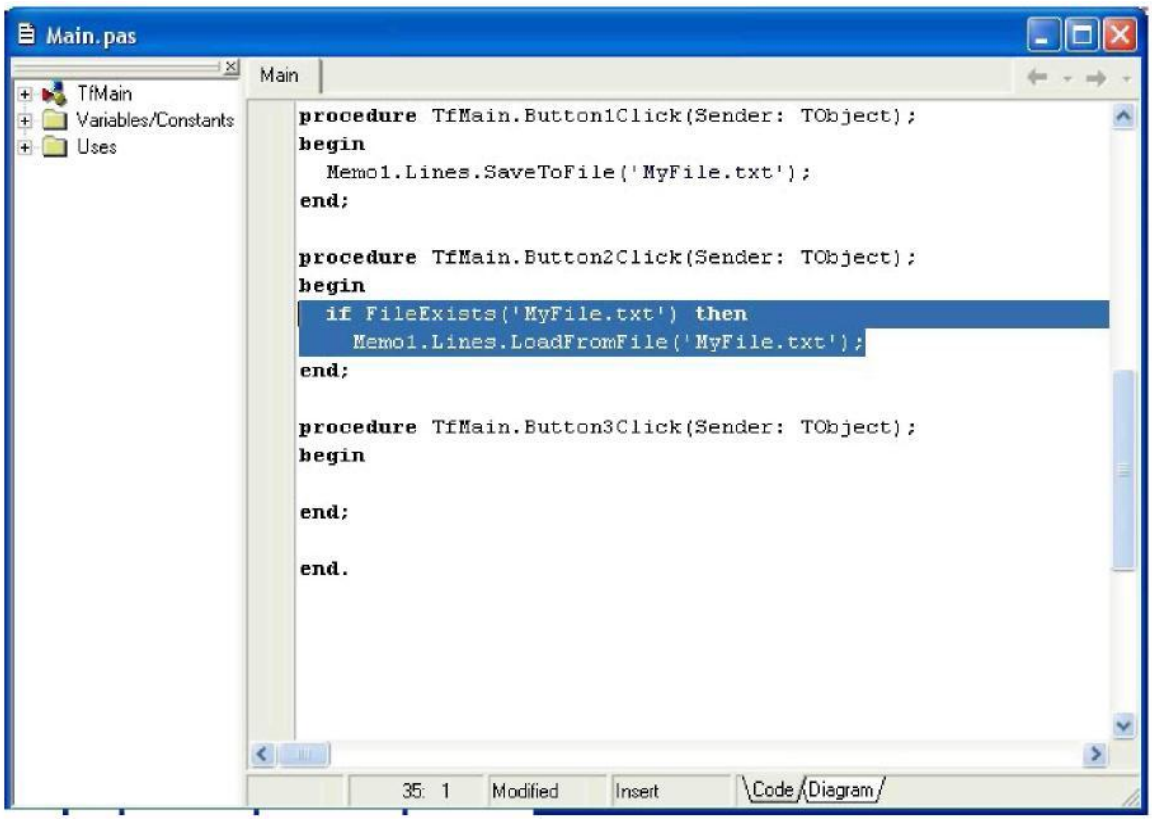



Рисунок 2.6 – Видалення непотрібних процедур

Після того, як ви збережете, всі порожні процедури будуть автоматично видалені. Це стосується останніх процедур - якщо після такої порожній процедури буде присутній діюча процедура або функція, вони обидві залишаться в коді.

Таким чином, у вас повинна вийти форма, на якій розташований тільки компонент Memo, і більше нічого. У редакторі коду не повинно залишитися жодного процедури. Загалом, тільки заготівля програми.

Виділіть компонент Memo, і переконайтеся, що в його властивості Align встановлено значення alClient, тобто, Memo розтягнуто на всю форму. На вкладці Standard знайдіть компонент MainMenu (головне меню), і встановіть його на будь-яке місце форми, прямо на компонент Memo. Компонент MainMenu не візуальний, тобто, користувач все одно не буде його бачити.

Будемо створювати меню. Двічі клацніть по MainMenu, щоб викликати редактор меню. Коли редактор відкриється, ви побачите, що перший пункт меню виділений синім кольором. Нехай виділення залишається, перейдіть на

властивість Caption і введіть текст «Файл». Натиснувши <Enter>, ви сформуєте у меню команду «Файл», а виділення переміститься направо, до наступної команди. Інший пункт меню ми поки робити не будемо, клацніть мишею трохи нижче команди «Файл», щоб виділити пункт нижче. Виходить, що ми зробили пункт меню «Файл», і зараз робимо підменю цього пункту. Нам потрібні такі команди:

«Зберегти»

«Завантажити»

«Очистити»

«-»

«Вихід»

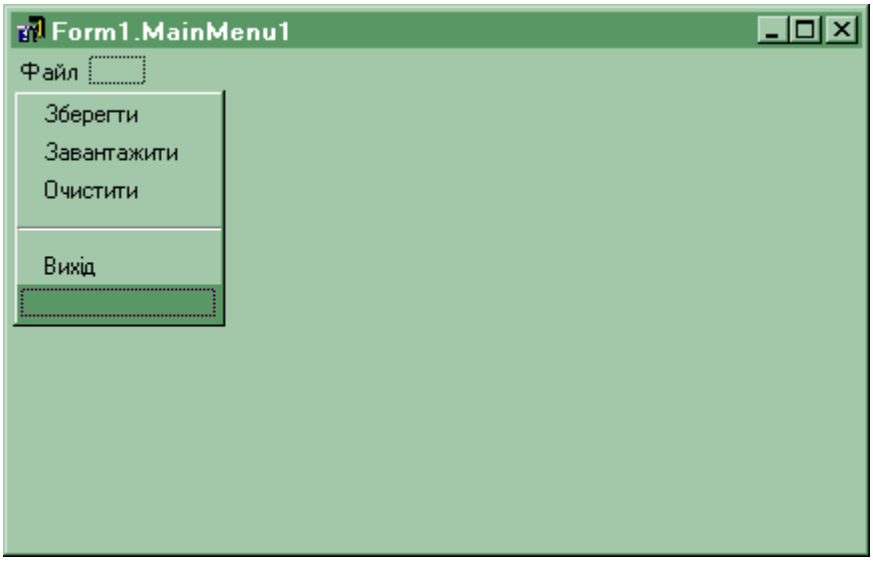



Рисунок 2.7 – Зовнішній вигляд головного меню редактора

Передостання команда, знак «-» (мінус), формує в меню розділову смугу. Як тільки ви закриєте редактор меню, рядок з головним меню зараз же з'явиться над компонентом Memo. Клацніть один раз по слову «Файл», і відкриється підменю. Клацніть по команді «Зберегти», і буде створена процедура обробки цієї команди. Команда «Зберегти» по -, як і раніше виглядає, як

Memo1.Lines.SaveToFile('MyFile.txt');

Всі інші команди ви вже знаєте, введіть їх аналогічним чином. Команда «Вихід» виглядає так:

Close; //вихід із програми

**Спливаюче меню**

Спливаюче меню викликається, коли користувач клацає правою кнопкою миші по об'єкту - формі або будь-якого іншого компоненту. Знайдіть на вкладці Standard компонент PopupMenu (спливаюче меню), і також встановіть його поверх компонента Memo. Редактор цього меню викликається таким же чином, як і редактор головного меню. У спливаючому меню зазвичай використовують тільки один головний пункт (хоча можливість зробити подпунткы є). Іншими словами, не потрібно робити пункти меню (Файл, Правка, Вигляд і т.д.) і їх підпункти (Файл -> Створити Файл -> Завантажити і т.д.).

Створіть наступні команди:

«Зберегти»

«Завантажити»

«Очистити»

«-»

«Вихід»

Щоб створити обробник подій для команди, двічі клацнувши по ній в редакторі спливаючого меню. Самі команди точно такі ж, як і у головного меню. Напишіть код для всіх зазначених пунктів контекстного меню.

Примітка: у подальших лекціях ми будемо вивчати компонент ActionList, який дозволяє використовувати одну команду для однойменних пунктів головного і спливаючого меню, а також панелі інструментів. Поки що нам доведеться дублювати команди головного і спливаючого меню.

Тепер спливаюче меню потрібно прив'язати до формі, саме по собі воно працювати не буде. Для цього нам потрібно виділити форму, що є непростим завданням - компонент Memo розтягнутий на все вікно, і немає можливості клацнути по вільного місця форми. Форму найпростіше виділити у вікні Дерева об'єктів (Object TreeView). Якщо у вас в даний момент це вікно закрите, відкрити його можна командою меню View -> Object TreeView, або гарячими

клавішами <Shift+Alt+F11>. У цьому вікні можна легко виділити будь-який компонент, у тому числі і форму.

Отже, виділіть форму. У вікні Інспектора об'єктів позначаться властивості форми. Нас цікавить властивість PopupMenu. Воно має вигляд списку, в якому ми можемо вибрати те або інше спливаюче меню. Оскільки таке меню у нас тільки одна, його і вибираємо. Тепер можна зберегти проект, скомпілювати його і запустити на виконання. Клацніть правою кнопкою миші на будь-якому місці форми призведе до виклику контекстного меню.

Спливаюче меню також називають контекстними - справа в тому, що багато компоненти мають властивість PopupMenu - редактор Memo, панелі і багато інші компоненти. Можна встановити кілька випадаючих меню з різними командами, і прив'язати до різних компонентів свої власні PopupMenu. Тоді клацніть правою кнопкою над одним компонентом призведе до виклику одного спливаючого меню, над іншим - іншого.

### 2.5 Масиви і константи

#### Масиви

Масив – це формальне об'єднання декількох однотипних змінних в одну. Тобто, масив розглядається як одна змінна, він має одне ім'я, але насправді там є кілька змінних одного типу. Масиви, як і змінні, оголошуються в розділі змінних var, і мають такий синтаксис:

Ім'я змінної : array [діапазон значень] of Тип змінних в масиві;

Приклад:

```
var
a : Integer; //змінна цілого типуЪ : array [1..10] ofInteger; //масив цілого типу
begin
```

Зверніть увагу, що діапазон значень в масиві вказується довільно, між першою і останньою цифрою немає пробілів, але обов'язкові дві точки. Приклад діапазону може бути таким: [5..7], які цифри призначати - вирішувати Вам. Зазвичай у практиці доводиться починати діапазон з 1 або 0, в залежності від ситуації.

Звернення до окремого розділу масиву таке:

```
a: = 5;
b[1] := a;
b[2] := b[1]+2;
```

Тобто, відразу після імені масиву ставляться квадратні дужки, усередині яких вказують індекс - номер елемента в масиві. Подібний масив можна сприймати як таблицю з одним рядком і декількома комірками:

| b[1] | b[2] | b[3] | b[4] | b[5] | b[6] | b[7] | b[8] | b[9] | b[10] |
|---|---|---|---|---|---|---|---|---|---|

Рисунок 2.8 – Подання масиву у вигляді таблиці

Оскільки в наведеному прикладі оголошено масив цілих чисел типу Integer, кожній комірці (елементу) масиву буде виділено за 4 байти пам'яті, кожен такий елемент відповідає звичайній змінної такого ж типу. Подібні масиви називаються одновимірними. З багатовимірними і динамічними масивами ми познайомимося пізніше.

Як вже згадувалося, масиви можуть бути будь-якого типу:

```
var
   a: array [0..3] ofString;
   b: array [1..5] of Real;
   c: array [10..20] of Boolean;
```

**Обробка масиву**

Масиви зручно використовувати там, де доводиться обробляти велику кількість однотипних даних. Візьмемо гіпотетичний приклад: потрібно перевести дані від 1 до 100 американських миль у звичні нам кілометри. Оскільки ми точно знаємо, що у нас буде 100 елементів, можна скористатися масивом, а для його обробки найзручніше використовувати цикл for:

```
var
i : Byte; //лічильник для for
```

```
a : Array [1..100] of Real; //масив для отриманих даних
begin
for i := 1 to 100 do
 a[i] := i * 1,609;
```

Що ми маємо в даному прикладі? При першому проході циклу for лічильник і має значення 1, отже, в рядку

```
a[i] := i * 1,609;
```

ми звертаємося до першого (a[1]) елементу масиву, і присвоюємо йому значення і, помножене на 1,609 кілометрів. Отже, у елемент масиву a[1], який відповідає одній милі, потрапляє значення 1,609 кілометрів. При другому проході циклу і вже дорівнює 2, значить, у елемент a[2] піде значення 2 * 1,609, тобто 3,218. І так далі, до кінця масиву. Надалі, звернувшись до потрібного елемента масиву, ми зможемо дізнатися, скільки кілометрів буде одно ця кількість миль:

```
ShowMessage('23 милі = ' + FloatToStr(a[23]) + ' кілометра');
```

З наведеного вище прикладу ви можете почерпнути корисний прийом програмування: як індексу масиву можна вказувати не тільки ціле число, але і лічильник циклу, або значення будь-який змінної цілого типу. Однак при програмуванні циклу слід дотримуватися обережності: якщо вказати індекс, який не входить в діапазон значень масиву, станеться помилка. Так, якби ми спробували виконати код

```
for i := 0 to 100 do
a[i]:= i * 1,609;
```

то отримали б помилку, так як вже при першому проході ми звернулися б до елемента a[0], якого в нашому прикладі не існує.

**Константи**

Константи - це такі ж, як змінні, комірки оперативної пам'яті, значення яких не можна змінити. Тобто, одного разу ви вказали значення константи, і надалі звертаєтеся до неї, як до звичайної змінної, але присвоїти їй інше значення вже не зможете. Константи зручно використовувати, коли ви будете

неодноразово звертатися до одного і того ж значення. Тип константи вказувати не потрібно, компілятор сам підбирає для неї відповідний тип. Константи описуються в розділі const, який завжди повинен описуватися перед розділом var:

```
procedure MyProc;
const
pi = 3.14;
MessageError = 'Текст повідомлення про помилку';
var
a : Integer; b : String;
 begin
... end;
```

Константи, особливо глобальні, зручно використовувати там, де ви працюєте з якимось певним значенням. Якщо надалі це значення зміниться, немає необхідності скрупульозно змінювати код у всій програмі, достатньо змінити значення константи. Наприклад, ви виробляєте розрахунок окладу, ґрунтуючись на сумі, що відповідає мінімальній заробітній платі. У такому разі не гріх оголосити константу MinZarPlat та присвоїти їй поточне значення мінімальної зарплати.

У процесі розрахунку вам доводиться множити це значення на кількість мінімальних зарплат. Припустімо, хтось Сидоров має оклад, рівний 5-ти мінімальним зарплатам. У момент розрахунку ви множите 5 не на суму мінімальної зарплати, а на константу MinZarPlat, яка містить потрібне значення. А якщо через місяць або рік сума мінімальної зарплати зміниться, то вам залишиться тільки замінити значення вашої константи на нове, не змінюючи при цьому іншого коду. Зручно, чи не так?

Пройдений матеріал на практиці.

Створимо проект, що використовує всі нові можливості. Створіть новий проект, перейменуйте форму в fMain, а у властивості Caption вкажіть «Випадкові числа». Збережіть в нову папку. Проект назвіть RandomNum.

Встановіть на форму Memo, видаліть з нього весь текст. Нижче кнопку з назвою «Генерувати».

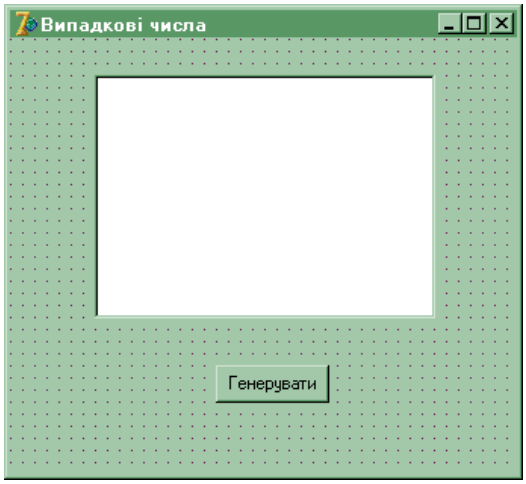



Рисунок 2.9 – Остаточний вигляд форми

У обробнику натискання на кнопку впишемо такий текст:
procedureTfMain.Button1Click(Sender: TObject);

constMaxValue = 1000; //записуємо максимальний розмір випадкових чисел

var

a : array [1..100] ofInteger; //масив цілих чисел з 100 елементів

i : integer; //лічильник для for

s : String;

begin

//спочатку очистимо Memo:

Memo1.Clear; //Заповнюємо випадковими числами від 0 до MaxValue масив a:

fori := 1 to 100 do

a[i] := Random(MaxValue); //дані з масиву додаємо спочатку в

//строкову змінну, потім в Memo:

s :=’’;

for i := 1 to 100 do

s := s + IntToStr(a[i]) + ', ';

Memo1.Lines.Add(s);

```
end;
```

В результаті виконання цього коду ми отримаємо рядок s, в якій через кому будуть перераховані 100 випадкових чисел від 0 до 1000, яку потім додамо в компонент Memo.

**Оператор циклу repeat...until**

Ми вже знаємо умовний цикл while, вивчимо ще один. Синтаксис циклу нескладний:repeat<тіло циклу>until<умова>;

Якщо цикл while мав передумову, то цикл repeat має постумову, яке виконується після тіла циклу. Є ще відміна – цикл while виконується до тих пір, поки умова вірна, а цикл repeat буде виконуватися, поки умова помилкова. У циклі while, якщо умова спочатку помилкова, тіло циклу ніколи не буде виконуватися. Repeat виконає тіло циклу принаймні один раз. Потім оператор перевіряє умову, і якщо воно помилкова, виконує тіло циклу ще раз, і так до тих пір, поки <умова> не поверне істину. Ще одне зауваження: зарезервовані слова repeat...until працюють як дужки begin...end, тому в тілі циклу може бути скільки завгодно інструкцій, які вже не потрібно укладати в begin...end. Приклад:

```
i := 3;
repeat
    i := i + 1;
ShowMessage('i = ' + IntToStr(i));
until i > 10;
```

**Символьні типи даних**

Символ – це один знак. Будь-яка - буква, цифра, арифметичний знак або пробіл, знак пунктуації або підкреслення. А також спеціальні символи – перехід на новий рядок, BackSpace, знак долара або відсоток. Тип «символ» Delphi позначається Char:

```
var
    c : Char;
begin
```

```
  c := 'B';
ShowMessage('Вы ввели ' + c);
c := #13;
ShowMessage('Переход на новую' + c + 'строку'); …
```

Ми вже говорили, що символи беруться з таблиці символів ANSI або UNICODE. Більшість символів використовуються користувачами, деякі символи є службовими. Зверніть увагу, що велика літера «А» і маленька «а» - це різні символи! Також різними символами є латинська «з» і російська «з», хоча вони схожі, як дві краплі води.

Нульовий символ не використовується, він зарезервований як повний нуль. Програмісти знайшли гідне застосування цього символу, використовуючи його в подіях введення тексту, коли потрібно заборонити користувачу введення будь-яких символів. Службові символи ми не можемо побачити в текстовому полі. Службові символи, це <Esc>, <Enter>, <Tab> та інші. Кожен символ обробляється комп'ютером як число від 0 до 255, таким чином, слово «ПРИВІТ» в пам'яті машини буде виглядати як набір цифр: «207 208 200 194 197 210».

**Функції роботи з символами**

У практиці часто доводиться обробляти окремі символи. Змінним символьного типу Char можна привласнювати значення таким чином:

```
var
c,b : Char;
begin
c := 'a'; b := c;
```

Оскільки для комп'ютера символ - число, то символьні дані можна порівнювати між собою. При цьому більшим буде той символ, число якого в таблиці символів більше. Наприклад, ‘я’ буде більше, ніж ‘а’:

```
if b > c then ShowMessage('Істина!') else ShowMessage('Брехня!');
```

При роботі з символьними змінними часто використовують функції(Chr) і Ord(). Функція(Chr) приймає в якості параметра число, і повертає символ, який

відповідає цьому числу в таблиці ANSI: function Chr ( X: Byte ): Char; Функція Ord() здійснює прямо протилежне дію, вона приймає в якості параметра символ, і повертає число, під яким цей символ зберігається в таблиці ANSI: function Ord ( C: Char ): Byte;

Символьні змінні можна використовувати з цими функціями:

```
a := Chr(200);
i := Ord(a);
i := Ord('f');
```

У першому рядку в змінну a ми записали літеру «I», якому в таблиці символів відповідає номер 200. У другу, цілу змінну, ми записали цифру 200, так як символ з цим номером був записаний в змінну a, яку ми передали в якості параметра. Нарешті, у третьому рядку ми цілу змінну записали цифру 102, цей номер відповідає символу «f».

**Діалоги**

Що таке діалоги в Delphi? Це невізуальні, тобто, невидимі користувачеві компоненти, які виконують стандартні для Windows діалоги користувача і програми. Наприклад, діалог відкриття або збереження файлу, діалог вибору шрифту або кольору, і т.п. Будь-яка серйозна програма містить такі діалоги. Робота з ними нескладна, кожен діалог містить буквально за два властивості, які нам будуть потрібні. Вивчимо роботу діалогів на прикладі.

Відкриваємо новий проект. Відразу ж форму перейменовуємо в fMain, у властивості Caption пишемо «Мій блокнот». Збережіть проект під назвою Editor. Це буде більш професійна версія редактора текстів, який ми вже робили. Вважайте, що це друга версія програми, що і напишете у вікні fAbout.

Встановіть на форму компонент Memo, видаліть весь текст з його властивості Lines. Поверх Memo встановіть MainMenu і PopupMenu. Перейдіть на вкладку Dialogs на панелі компонентів, а також поверх Memo встановіть OpenDialog, SaveDialog, FontDialog і ColorDialog. Всі ці компоненти не візуальні, користувач їх бачити не зможе. У Memo відразу ж вкажіть

властивість Align=alClient, а властивість ScrollBar=ssVertical. Рекомендую для кращого вигляду вибрати шрифт (властивість Font) Times New Roman, розмір = 12. На всіх комп'ютерах з російської Windows є цей шрифт, однак якщо ви будете писати програму для продажу в Інтернет, краще залишити шрифт за замовчуванням, це гарантує, що він буде читатися на всіх комп'ютерах в світі.

У всіх діалогах є один метод, який нас цікавить – Execution. Це метод логічного типу, він повертає True, якщо діалог з користувачем стався успішно (наприклад, користувач вибрав відкривається файл), і False в іншому випадку (наприклад, користувач відмовився від вибору файлу). У зв'язку з цим, діалоги зазвичай застосовують разом з перевіркою:

```
ifOpenDialog1.Execute then ...
```

У діалогах, пов'язаних з файлами (OpenDialog - відкрити файл, і SaveDialog - зберегти файл), є властивість FileName, яке повертає рядок - адресу та назву виділеного файлу. Ось, власне, і все, що нам потрібно від цих діалогів!

Діалог FontDialog у властивості Font возвращает вибраний шрифт. Це властивість непростий, воно має тип TFont, і привласнити його можна тільки іншому шрифту, наприклад:

```
Memo1.Font := FontDialog1.Font;
```

Точно також, діалог ColorDialog возвращает властивість Color - колір, що має тип TColor, і це властивість можна присвоїти тільки об'єкту, що має такий же тип:

```
Memo1.Color := ColorDialog1.Color;
```

Продовжимо наш проект. Відкрийте редактор головного меню. Створіть розділ «Файл» і підрозділи «Відкрити», «Зберегти», «Закрити», «-» і «Вихід». Створіть розділ «Настройки» і підрозділи «Вибрати шрифт» і «Вибрати колір». Створіть розділ «Довідка» і підрозділ «Про програму».

Тепер відкриваємо редактор PopupMenu і вписуємо розділи «Відкрити», «Зберегти», «Закрити», «-» і «Вихід». Відразу ж на формі виділяємо саму форму (це можна зробити у вікні Object - TreeView), і у властивості PopupMenu

вибираємо наше меню. Тепер це меню відкриється, якщо користувач клацне правою кнопкою з будь-якого місця на формі.

Тепер нам потрібно налаштувати фільтри в діалогах OpenDialog (Відкрити) і SaveDialog (Зберегти). Фільтри дозволяють відображати в цих діалогах тільки потрібні формати файлів, і для цього вибору використовується маска. Приміром, маска *.* буде відображати файли всіх типів! Двічі клацнувши по властивості Filter діалогу OpenDialog, відкрийте редактор фільтрів. У першій колонці напишіть

«Текстові документи», другий - «*.txt». У рядку введіть «Всі файли», а у другій колонці - «*.*». Теж саме зробіть для діалогу SaveDialog:

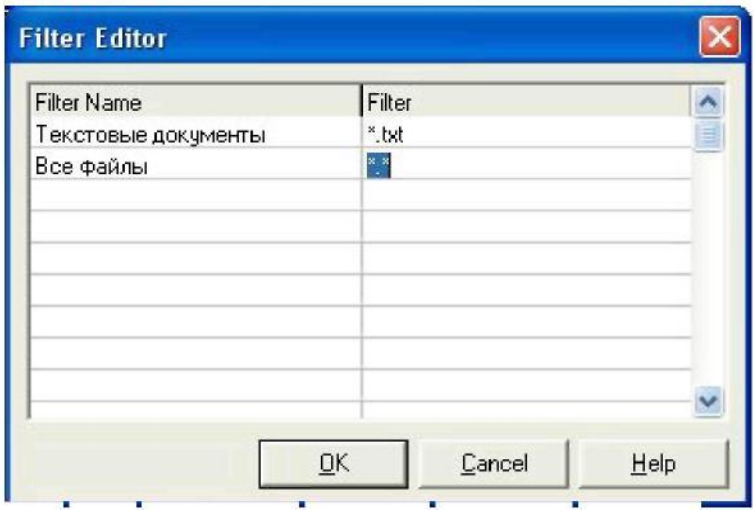



Рисунок 2.10 – Налаштування фільтра файлових діалогів

Тепер, відкриваючи файл, ви побачите тільки ці два типи файлів. Подумаємо про те, що програма повинна знати - чи змінився текст у Memo. Адже користувач може закрити програму і не зберегти текст, а потім буде лаяти програміста за те, що він цього не передбачив. При цьому маємо на увазі, що у нас є багато команд меню, значить, буде багато процедур. А щоб дати програмі знати, чи змінився текст, найрозумніше створити змінну логічного типу - змінився текст, присвоюємо їй True, інакше False. Щоб з цієї змінної можна було працювати з усіх процедур, вона повинна бути глобальною. Робимо глобальну змінну перед словом implementation:

```
izmen : Boolean; //чи змінився текст у Memo
```

Події onChange компонента Memo присвойте рядок:

```
izmen := True;
```

Як тільки зміниться текст у Memo, мінлива тут же буде видавати істину.

Тепер ще один момент - програма повинна знати ім'я та адресу відкритого файлу. Якщо імені файлу немає, то програма буде виводити діалогове вікно, а якщо ми файл вже відкривали і ім'я є, то програма просто буде перезаписувати той файл без виведення діалогового вікна. Стало бути, робимо ще одну глобальну змінну:

```
myfile : String; //Адресу та ім'я відкритого файлу
```

Я спеціально не беру широко поширені слова «File» або «FileName», так як вони можуть бути зарезервованими або в компонентах можуть бути властивості з такими іменами, в результаті вийде конфлікт назв.

Метикуємо далі. Відкрити файл можна буде командою меню «Файл - Відкрити», або командою PopupMenu «Відкрити». Стало бути, нам доведеться двічі писати один і той же код? А якщо він буде великим і складним? Можна, звичайно, і скопіювати його, компілятор це витримає, і програма буде працювати нормально. А як же оптимізація коду? Два однакових коду в програмі будуть займати в два рази більше місця в пам'яті і на диску! Для цього ми маємо користувальницькі функції і процедури.

Згадуємо - функцію або процедуру ми повинні описати ВИЩЕ того місця, де будемо її використовувати, значить, першу нашу процедуру ми повинні описати в розділі implementation, прямо під рядком {$R *.dfm}.

Зараз ви дізнаєтеся ще щось новеньке. У таких ось користувальницьких процедури і функції ви не можете безпосередньо звертатися до компонентів форми, оскільки Ваші процедури і функції самій формі не належать. Якщо Ви введете

```
Memo1.
```

то компілятор відразу видасть помилку. Звертатися до компоненту потрібно, вказавши спочатку ім'я форми, на якій він знаходиться:

```
fMain.Memo1.
```

Однак такий код буде не дуже зручний - кожен раз доведеться звертатися до компонентів через форму. Зайвий набір коду, надто довгий текст. Вихід є - функція with (c). Ця функція має вигляд:

```
with fMain do begin
... end;
```

де fMain - ім'я форми з потрібними нам компонентами. Тепер між дужок begin...end цієї конструкції ми можемо звертатися до компонентів форми напряму. Пишемо загальну для всіх процедуру відкриття файлу:

```
{Процедура відкриття файла} procedureOtkrivaem; begin
withfMaindobegin //робити разом з формою
ifOpenDialog1.Executethenbegin //якщо діалог виконаний
//присвоюємо змінної myfile адресу та назву виділеного файлу:
myfile := OpenDialog1.FileName;
//читаємо цей файл у Memo: Memo1.Lines.LoadFromFile(myfile);
izmen := False; //файл тільки відкритий, змін ще немає
end; //endif; //withend;
```

Тепер створюємо обробник подій для команди меню «Файл - Відкрити». Там викликаємо нашу процедуру:

```
Otkrivaem;
```

Теж саме робимо для команди PopupMenu «Відкрити».

Далі - складніше. Відкрити файл просто, але при збереженні нам доведеться враховувати багато речей:

1. Користувач може зберігати новий файл, який він тільки що набрав. Тобто, мінлива myfile порожня і не містить ім'я файлу. У такому випадку доведеться виводити діалогове SaveDialog1, щоб користувач вказав ім'я файлу. Коли він вкаже ім'я, привласнити його змінної myfile і зберегти Memo у вказаний файл.

2. Користувач може зберігати новий файл, як у попередньому прикладі. Ми виведемо діалог, але він його не завершить - натисне кнопку «скасувати» або червоний хрестик у верхній частині вікна праворуч. Значить, зберігати нічого не потрібно, адже ми не знаємо, куди зберігати! Але доведеться його попередити, що файл не збережений!

3. Користувач може відкрити файл і змінити його, а потім зберігати. У змінній myfile це ім'я вже є, діалог не потрібна, просто перезаписуємо цей файл.
4. Користувач може зберігати файл, текст якого не змінювався. У такому випадку, просто ігноруємо його команду!

Бачите? Користуватися діалогами нескладно, але як багато доводиться враховувати під час перевірок дій користувача! Потрібно враховувати кожну дрібницю, інакше вам скажуть, що ваша програма повна «багів» - дрібних несправностей. Отже, пишемо процедуру збереження, відразу під процедурою відкриття:

```
{Процедура збереження файлу}
procedure Sohranyaem;
begin
with fMain do begin
//якщо змін не було, виходимо з процедури,
//нічого не роблячи:
if not izmen then Exit;
//Якщо файл вже відкривався, і в змінній myfile
//є його адреса та ім'я, просто перезаписуємо цей файл:
if myfile<> '' then
begin
   Memo1.Lines.SaveToFile(myfile);
izmen := False;
Exit; //виходимо після збереження
end;
```

```
    {Файл новий, змінна myfile ще порожня. Далі є два варіанти: користувач вибере або вкаже файл у діалозі, або не зробить цього}
    //якщо вибрав файл:
    If SaveDialog1.Execute then begin
    //прописуємо адресу та ім'я файлу змінну:
    myfile := SaveDialog1.FileName;
    //якщо немає розширення *.txt то додаємо його:
    if copy(myfile, length(myfile)-4, 4) <> '.txt' then
    myfile := myfile + '.txt';
    //зберігаємо Memo у вказаний файл:
    Memo1.Lines.SaveToFile(myfile);
//файл збережено, змін немає:
    izmen := False; end
    if //
    //якщо не вибрав файл:
    else ShowMessage('Ви не вказали ім'я файлу, файл не збережений!');
    end; //with end;
```

Наведений вище код має досить детальні коментарі, так що все повинно бути зрозуміло. Нове, що ви могла побачити - директива Exit. Ця директива достроково завершує роботу процедури (або функції). Тобто, якщо виконана умова і відпрацьований потрібний код, Exit змушує процедуру завершити роботу. Інший код, який є в цій процедурі, не виконується.

Тепер ми можемо створити обробник головного меню «Файл - Зберегти», і там прописати виклик цієї процедури:

Sohranyaem;

Теж саме робимо для команди PopupMenu «Зберегти».

Далі йде команда головного і спливаючого меню «Вихід». Тут все просто, в обох випадках пишемо команду

```
Close;
```

Далі йде розділ «Настройки». Створюємо обробник «Вибрати шрифт», там все просто:

```
ifFontDialog1.Executethen
Memo1.Font := FontDialog1.Font;
```

Точно також і з підрозділом «Вибрати колір»:

```
ifColorDialog1.Executethen
Memo1.Color := ColorDialog1.Color;
```

Далі повернемося в розділ «Файл». Тут у нас залишився підрозділ «Закрити». Створюємо для нього обробник, в який запишемо:

{якщо файл не збережений, попереджаємо користувача про це. Якщо він бажає зберегти, то викликаємо процедуру збереження:} `if izmen then`

```
if Application.MessageBox('Файлзмінено. Зберегти?', 'Увага!', MB_YESNO+MB_ICONQUESTION) = IDYES then Sohranyaem;
//тепер закриваємо поточний файл: Memo1.Clear; //очищаємо Мемо myfile := ''; //немає імені поточного файлу izmen := False; //немає змін
```

Тут не так багато тексту, і він цілком зрозумілий, тому не будемо для нього робити окрему процедуру. Просто скопіюйте текст, створіть обробник подій для «Закрити» PopupMenu та вставте цей же текст туди.

Приблизно теж саме нам доведеться зробити і для події форми onClose - адже ми заздалегідь не можемо знати, яким чином користувач закриє форму, через пункт головного меню «Файл - Закрити», через команду контекстного меню «Закрити», натисне чи він червоний хрестик нагорі або натисне гарячі клавіші <Alt + F4>! Тому для перевірки - чи є в програмі незбережений текст, ми використовуємо подія onClose, що трапляється при спробі закрити форму, тобто в даному випадку всю програму. Тут код буде таким же, як і в попередньому прикладі, але раз ми закриваємо всю програму, нам не потрібно виконувати блок закриття файлу. Просто перевіримо, чи є зміни, і якщо вони є,

виведемо запит - чи зберегти їх. Якщо користувач скажуть «Так», значить, викличемо процедуру збереження:{якщо файл не збережений, попереджаємо користувача про це. Якщо він

бажає зберегти, то викликаємо процедуру збереження:} ifizmenthen

ifApplication.MessageBox('Файл змінено. Зберегти?', 'Увага!', MB_YESNO+MB_ICONQUESTION) = IDYES then Sohranyaem;

Тепер введемо ще одну команду для обох меню - «Очистити». Вона буде призначена на випадок, якщо користувач введе абракадабру, а потім вирішить очистити текст. Я навмисно залишив цю дію на останок, щоб ви могли засвоїти, що в процесі роботи над програмою меню може змінюватися!

Відкрийте редактор MainMenu, в розділі «Файл» в самому низу додайте підрозділ «Очистити». Потім перетягніть мишкою його на місце лінії розділів, при цьому лінія опуститься вниз, а новий підрозділ встане на її Створіть для команди «Очистити» оброблювач, і впишіть туди текст:

```
{MainMenu - Файл - Очистити}
procedureTfMain.N17Click(Sender: TObject);
begin
//очищаємо Memo: Memo1.Clear;
//якщо відкритого файлу немає, то немає і змін: if myfile = '' then izmen := false //інакше текст змінено: elseizmen := true; end;
```

Зробіть те ж саме для спливаючого меню - створіть розділ «Очистити», а в його обробник скопіюйте той же текст.

Залишилося створити модальне вікно fAbout, де ви вкажете, що це текстовий редактор, другої версії. Зробіть це самостійно і прив'яжіть виклик цієї форми до команди головного меню «Довідка Про програму».

Все, зберігайте, компилируйте і перевіряйте, як працює програма! У нас вийшов нескладний, але цілком професійний редактор текстів. Одне зауваження: текст ми зберігаємо, але вибраний шрифт і колір вікна ви не

можете зберегти, про це ми будемо говорити на інших лекціях, коли навчимося зберігати параметри.

**Робота з текстовими файлами**

***TStringList***

Практично будь-яка програма взаємодіє з якими-або типами файлів. Delphi дозволяє нам працювати з файлами різних типів. Текстовий файл - одна з цих різновидів. У цих файлах інформація розташована не суцільним блоком даних, а у вигляді рядків, які закінчуються символом кінця рядка і переведення каретки. Можна працювати з текстовими файлами загальним способом, тобто, побайтно зчитувати і записувати дані. Однак це не завжди зручно. Розглянемо приклад:

*Привіт! Як життя?*

Якщо ми вважаємо ці 2 рядки суцільним блоком, то отримаємо рядок:

*'Привіт!' + #13 + #10 + 'Як життя'*

Це дуже незручно, так як для обробки рядків нам кожен раз доведеться шукати ці два символи кінця рядка. А якщо рядків багато, наприклад 100? І якщо потрібно отримати доступ до 75-й рядку?

Тут на допомогу приходить об'єкт TStrings, який ми використовували при обробці Memo, і який є простим контейнером для зберігання рядків. Однак використовувати об'єкт TStrings безпосередньо не можна, так як цей об'єкт є шаблоном для інших об'єктів, що працюють з рядками. Згадайте, в компоненті Memo і Listbox ми теж не використовували цей об'єкт безпосередньо, ми використовували його «нащадків» - Lines і Items. Тому краще використовувати більш просунутий варіант цього об'єкта - TStringList. Цей варіант успадковує всі можливості об'єкта TStrings і додає ще нові. Це, мабуть, найпростіший спосіб роботи з текстовими файлами.

Щоб скористатися цим об'єктом, потрібно спочатку визначити змінну цього типу, потім запустити її, а в кінці закрити. У простому варіанті це буде виглядати так:

```
var
  f : TStringList;  //об'явили змінну - об'єкт
begin
f := TStringList.Create();  //ініціалізували її
f.Free;  //звільнили
 end;
```

Тут ми створюємо об'єкт і тут же його знищуємо, не працюючи з ним. Коли ми оголосили змінну типу об'єкт TStringList, ми вказали компілятору, скільки оперативної пам'яті буде потрібно виділити в подальшому для цієї змінної. Коли ми проініциализировали об'єкт методом Create, ми виділили під нього оперативну пам'ять. Надалі, з цим об'єктом можна працювати, записувати в нього текст, читати текст, користуватися властивостями, методами і подіями цього об'єкта. Метод Free руйнує об'єкт, знищує його в оперативній пам'яті. Після цього до змінної типу f TStringList звертатися вже не можна. Все, що повинен був робити цей об'єкт, повинно бути описано до методу Free.

Читання та запис з таким об'єктом відбувається знайомим нам способом. Припустимо, нам потрібно прочитати текстовий файл і змінити в ньому 10-ий рядок:

```
var
  f : TStringList;
begin
  f := TStringList.Create();
  //читаємо текст з файла:
  f.LoadFromFile('c:\myfile.txt');
  //змінюємо текст 10-го рядка (рядки починаються з 0):
  f[9] := 'Новий рядок';
  //зберігаємо змінений текст знов у файл:
f.SaveToFile('c:\myfile.txt'); f.Free; end;
```

Зверніть увагу, що ми звернулися до 9-го індексу, так як нумерація індексів починається з нуля, і f[9] містить 10-й рядок.

Властивість Count цього об'єкта возвращает загальна кількість рядків, починаючи з першої. Давайте напишемо приклад, в якому ми будемо серед всіх рядків шукати рядок «Привіт, Київ!». Якщо такий рядок буде знайдений, то програма виведе номер шуканого рядка, інакше виведе повідомлення, що такого рядка немає.

Спочатку скористайтесь власним редактором текстів і створіть текстовий файл, куди запишіть 10-15 рядків. Один з рядків зробіть «Привіт, Київ!».

Потім робимо новий додаток, встановлюємо кнопку «Виконати» і діалог OpenDialog. Потім створюємо обробник подій для кнопки, куди записуємо наступний код:

```
procedure TForm1.Button1Click(Sender: TObject); var
  f : TStringList; i : Integer;
begin
  //якщо нічого не відкрили, виходимо з процедури:
   if not OpenDialog1.Execute then Exit;
  //читаємо з файла:
  f := TStringList.Create();
  f.LoadFromFile(OpenDialog1.FileName);
  //шукаємо рядок:
  for i := 0 to f.Count-1 do begin
    if f[i] = 'Привет, Москва!' then begin
  ShowMessage('Рядок знайдений під № ' +
IntToStr(i+1));
   Break; end; //if end; //for
 //закрываємо файл: f.Free; end;
```

Тут код досить прозорий, проте ще раз звертаємо вашу увагу на те, що індекси рядків починаються з нуля, а властивість Count возвращает загальна кількість рядків. Тобто, якщо в тексті лише 2 рядки і Count поверне цифру 2, то індекси цих рядків будуть 0 і 1, тому в циклі ми використовували діапазон від 0 до Count-1, інакше б помилка переповнення циклу.

Отже, крім того, що ми вже розглянули, цей об'єкт має й інші відомі вам за Memo методи:

**Add** - додати новий рядок у кінець:

*f.Add('Новий рядок');*

**Clear** - очистити об'єкт (не буде тексту):

*f.Clear;*

**Insert** - вставити рядок. Є 2 параметра - індекс вставляється рядка, і сама рядок:

*f.Insert(3, 'Нова 4-я рядок, наступні рядки зрушитися вниз');*

**Delete** - видалити рядок. Є параметр - індекс видаляється рядка.

*f.Delete(f.Count-1);* //видалили останній рядок

**Загальні принципи роботи з файлами**

У попередніх прикладах ми зберігали текст у файл і прочитували текст з файлу за допомогою властивостей:

```
Memo1.Lines.SaveToFile()

Memo1.Lines.LoadFromFile()
```

Цей метод зручний і гарний, однак він не дозволяє нам контролювати весь процес запису у файл і читання з файлу. Тобто, ці функції весь процес читання і запису виконують приховано від нас.

Однак буває, коли програміст відчуває необхідність контролю цих процесів, крім того, не для всіх типів ці функції доступні. Наприклад, у нас є текст посеред змінної. Ми вже говорили, що символьна мінлива може зберігати майже необмежену кількість символів, тобто в змінну ми можемо записати весь текстовий файл, включаючи і символи переходу на інший рядок і повернення

каретки (#13 #10). З функціями SaveToFile() і LoadFromFile() можуть працювати дані, які мають тип TStrings, а рядки не можуть викликати їх. Тому доводиться робити запису у файл безпосередньо, а також безпосередньо їх зчитувати.

Для роботи з файлами багато програмістів воліють використовувати функції WinAPI. Незважаючи на грізне звучання, нічого особливо складного в цих функціях немає, тим не менш, такі функції мають один великий недолік. Корпорація MicroSoft постійно вносить якісь зміни у свої операційні системи. Так, у перших версіях Windows, для читання файлів використовувалася функція WinAPI _lread. Потім з'явилася функція ReadFile і Microsoft стала рекомендувати використовувати в програмуванні саме її. А потім з'явилася функція ReadFileEx, яка дозволяє працювати з файлами великих розмірів. Після кожної зміни цих функцій дуже часто доводиться переробляти всю програму, щоб вона могла працювати з новою операційною системою. А це забирає багато часу і створює додаткові незручності і для програмістів, і для користувачів.

Тому в Delphi рекомендується використовувати вбудований спеціалізований об'єкт TFileStream. Якщо Microsoft введе якісь нововведення, Boland врахує їх в об'єкті, і нам залишиться лише перекомпілювати нашу програму, не змінюючи її коду. Крім того, використання TFileStream набагато простіше, ніж функції WinAPI. TFileStream - це об'єкт, і як кожен об'єкт, його потрібно спочатку оголосити, а потім запустити.

Насамперед необхідно створити змінну типу TFileStream :

```
var
  f : TFileStream;
```

Таким чином, ми оголосили змінну типу об'єкт TFileStream, і надалі можемо працювати з цієї змінної, як з об'єктом. Тобто, вказувати ім'я цієї змінної, а після точки вибирати властивості, методи і події цього об'єкта. Однак оголосити змінну мало, потрібно ще проініціалізувати її.

```
f := TFileStream.Create(параметри);
```

У методу Create об'єкта TFileStream може бути три параметри, причому третій параметр необов'язковий, його можна не вказувати. Розберемося з цими параметрами.

Ім'я файлу - цей параметр - проста рядок, яка може містити тільки ім'я файлу, або повне ім'я файлу, включаючи і адресу.

Режим відкриття. Тут можна вказати один з наступних параметрів:

- **fmCreate** - Створити файл з вказаним у першому параметрі ім'ям. Якщо файл вже існує, він відкриється в режимі запису.
- **fmOpenRead** - Відкрити файл тільки для читання. Якщо файл не існує, станеться помилка, тому перш потрібно виконати перевірку на існування файлу. Запис у файл у цьому режимі неможлива.
- **fmOpenWrite** - Відкрити файл для запису. При цьому поточний вміст файлу знищується, і файл перезаписується.
- **fmOpenReadWrite** - Відкрити файл для редагування, тобто, і читання і запису.
- Права, з якими буде відкритий файл. Цей параметр необов'язковий, його можна не вказувати. Він має наступні можливі значення:
- **fmShareCompat** - Інші програми теж мають право працювати з відкритим файлом.
- **fmShareExclusive** - Інші програми не зможуть відкрити файл.
- **fmShareDenyWrite** - Інші програми зможуть відкрити файл тільки для читання, записати в нього дані вони не зможуть.
- **fmShareDenyRead** - Інші програми зможуть відкрити файл тільки для запису, для читання вони не зможуть його відкрити.
- **fmShareDenyNone** - не заважати іншим програмам працювати з файлом.

Приклад:

```
f:=TFileStream.Create('C:\MyFile.txt', fmOpenReadWrite,
fmShareExclusive);
```

Для чого потрібні права доступу до файлу? Наприклад, текстовий файл може бути відкритий стандартною програмою «Блокнот», цей же ми можемо відкрити файл з нашого власного текстового редактора. Редактор менеджера файлів FAR також може відкрити цей текстовий файл. І програма MSWord теж може відкрити його! Тепер припустимо, що наша програма робить запис у файл. У цей же самий час якась інша програма також зберігає зміни в цьому файлі. Наша програма зробила зміни і закрила файл, а інша програма навіть не підозрює про ці зміни, і просто перезаписує файл зі своїми змінами. Таким чином, наші зміни просто губляться!

Теж саме може статися, якщо ваша програма виконана в мережевому варіанті. Наприклад, є якийсь спеціальний файл, у який співробітники відділу записують свої зміни. Цей файл може перебувати на якомусь мережевому диску, і доступ до цього файлу має кожен співробітник. І у кожного співробітника встановлена ваша програма. Таким чином, якщо одночасно двоє співробітників будуть проводити зміни у файлі, то дані одного з них загубляться, так як програма іншого перезапише файл без урахування цих змін. Якщо ви пишете однокористувацький програму, і доступ до файлу буде мати тільки вона, то про третій параметр можете зовсім забути.

Після того, як ви створили об'єкт, проініціалізовали його і попрацювали з ним, файл потрібно закрити і звільнити пам'ять, займану цим об'єктом. Для цього досить викликати метод:

```
f.Free;
```

Тепер залишилося розібратися зі структурою файлу, і методами читання з нього, і запису в нього. Почнемо зі структури.

Коли ви тільки відкрили файл, позиція курсора встановлюється на початок, і будь-яка спроба читання або запису буде застосована до цієї позиції

курсору. Щоб прочитати або записати в іншу позицію, потрібно пересунути курсор. Для цього використовують метод Seek, який має два параметри:

- Число, що показує кількість байт (символів), на які потрібно перемістити курсор.
- Звідки потрібно рухатися. Тут може бути три варіанти:
    - SoFromBeginning - рухатися на вказану кількість байт від початку файлу.
    - SoFromCurrent - рухатися від поточної позиції курсору.
    - SoFromEnd - рухатися на вказану кількість байт від кінця файлу до його початку.

Отже, щоб перемістити курсор від початку файлу на 10 байт, потрібно виконати команду:

```
f.Seek(10, soFromBeginning);
```

Метод Seek - це функція, вона завжди повертає кількість байт зміщення від початку файлу. Цією властивістю можна скористатися, щоб дізнатися загальна кількість байт у файлі:

```
Размер Файла := f.Seek(0, soFromEnd);
```

Правда, якщо нам потрібно просто подивитися розмір файлу, то цей трюк не зовсім придатний: нам доведеться відкрити файл, переміститися в його кінець, і потім закрити його. Як дізнатися розмір файлу більш підходящим способом, дізнаємося пізніше.

Для читання файлу з потрібно використовувати метод Read, у якого теж є два параметри:

- Змінна, в яку буде записаний прочитаний текст.
- Кількість байт, які варто прочитати.

Розглянемо приклад читання файлу з 10 символів, починаючи з 15-й позиції:

```
var
```

```
   f: TFileStream;  //об'являємо змінну
buf: array[0..10] ofChar;//буфер для зберігання даних
 begin
   //відкриваємо файл filename.txt для читання й
запису:

f:=TFileStream.Create('c:\filename.txt',fmOpenReadWrit
e); //зміщуємось на 15 символів вперед:
    f.Seek(15, soFromBeginning);
   //читаємо в буфер 10 символів з встановленої
позиції: f.Read(buf, 10);
Memo1.Lines.Add(buf);
 //скопірували ці 10 символов в Memo
//знищуємо об'єкт и тим самим закрываємо файл:
 f.Free; end;
```

Метод Read повертає кількість реально прочитаних байт. Воно повинно бути дорівнює тій кількості байт, які ми вказали при виклику цього методу. Є дві причини, за якими ця кількість може бути не одно вказаною:

1. При читанні було досягнуто кінець файлу, і функція прочитала менше байт, ніж планувалося.
2. Помилка на диску або будь-яка інша проблема.

Для запису в файл використовується метод Write. Є два параметра і у нього:

1. Змінна, вміст якої потрібно записати.
2. Число байт для запису.

Користуватися методом запису можна точно також, як і методом читання.

Примітка: після читання або запису позиція курсора зміщується на кількість прочитаних або записаних байт. Тобто, позиція курсора встановлюється після прочитаного або записаного блоку.

**Панель інструментів**

Що таке панель інструментів? Ми вже знаємо, що у багатьох програмах є панелі, на яких встановлені кнопочки з малюнками. Іноді кнопочки містять текст, частіше містять тільки зображення. При наведенні покажчика миші на таку кнопочку, через деякий час, виходить підказка - що це за кнопка. Як правило, кнопки панелі інструментів дублюють команди головного і (або) спливаючого меню, полегшуючи користувачеві роботу з програмою. Є два способи організувати панель інструментів, розглянемо обидва.

***Кнопка SpeedButton***

Створіть новий додаток. Встановіть на форму панель, очистіть у неї властивість Caption, властивість Align встановіть alTop, щоб вона зайняла весь верх форми. Властивість Height встановіть 24.

Тепер перейдіть на вкладку Additional і знайдіть кнопку SpeedButton. Встановіть її на панель. Щоб притиснути її до самого лівому краю, властивість Left встановіть у 0. Тепер погляньте на властивість Height, воно дорівнює 22. У панелі це властивість одно 24. Отже, щоб кнопка була посередині панелі, потрібно властивість Top встановити в 1. Тоді зверху і знизу кнопки вийде за 1 пікселя.

Тепер розглянемо, що ця кнопка вміє робити. Насамперед, вона відрізняється від інших кнопок тим, що не має фокусу вводу. Що це означає? При роботі програми один з компонентів має фокусу вводу, він виділений. Якщо це компонент для введення тексту (Edit, Memo), то користувач відразу може вводити текст. Якщо це кнопка, то користувач може натиснути <Enter>, що буде рівносильне натискання на кнопку миші. Крім того, клавішею <Tab> можна перемістити фокус вводу від одного компонента до іншого, порядок виділення компонентів визначається їх властивістю TabOrder. А кнопка SpeedButton фокусу вводу не має, її не можна виділити клавішею <Tab>, а якщо ви клацніть по ній мишею, то фокус вводу повернеться до того компоненту, в якому був до цього.

Властивість Glyph у цієї кнопки працює також, як у BitBtn і дозволяє завантажити на кнопку зображення. Відкрийте діалогове вікно цього властивості і виберіть картинку «DoorOpen» з стандартною колекції Delphi.

Двічі клацніть по кнопці SpeedButton і напишіть там вихід з програми:

```
Close;
```

Збережете, зберіть його, і подивіться, як кнопка працює.

У всіх сучасних додатках такі кнопки зазвичай виглядають більш плоскими. Щоб прибрати опуклість кнопки, змініть властивість Flat на True.

Тепер познайомимося ще з одним властивістю, яка має більшість компонентів. Це система підказок Hint, підказка виходить, коли користувач наведе курсор миші на компонент. У властивості Hint кнопки напишіть «Вийти з програми». Однак цього замало, потрібно ще дозволити підказці виходити на екран. За це відповідає властивість ShowHint. Але увага! Якщо ми встановимо це властивість у True, то тільки ця кнопка зможе виводити підказку, для інших кнопок його теж доведеться виставляти в True. Щоб цього не робити, можна встановити властивість ShowHint в True у батьківського компонента - панелі або самої форми. Тоді всі компоненти, розташовані на цьому об'єкті, будуть мати True в цій властивості. Давайте встановимо це властивість True у форми. Тепер і панель, і кнопка теж мають True в ShowHint. І будь-яка наступна кнопка, яку ми кинемо на панель поруч з першої також матимуть можливість виводити підказку. Зберіть проект і подивіться, як він працює.

Однак це ще не межа можливостей кнопки SpeedButton. Її ще можна групувати. Встановимо поруч ще дві такі кнопки. Зробіть так, щоб вони були впритул один до одного, але трохи віддалік від першої кнопки. На першу накладіть малюнок «led2on», на другу - «led2off» з колекції зображень Delphi:

Ці кнопки розташовані поруч, і мають схожі картинки. Таким чином, користувач вже бачить, що вони відносяться до якого-то одній властивості. Виділіть обидві кнопки (утримуючи <Shift>і встановіть у них властивість GroupIndex в одиницю. Цим самим ми згрупували їх в одну індексний групу. Тепер у будь-який з них встановіть властивість Down (натиснута) у True.

Збережете, зберіть його, і подивіться, як працює індексний група. Натисніть іншу клавішу, перша відіжметься. І навпаки. Саме таким чином ви можете, наприклад, вибрати накреслення шрифту або вирівнювання абзацу у MSWord.

Ви можете встановлювати скільки завгодно груп, і кожній їм у властивості GroupIndex свою цифру. Нуль означає, що кнопка не належить ні до якої групи. У кнопки з груповим індексом 0 ви просто не зможете перевести властивість Down у True.

Така панель інструментів виглядає привабливіше, ніж просто панель з кнопками, не чи правда? Проте є і більш професійний спосіб організації панелі інструментів.

***Переставні панелі інструментів***

Якщо вам в додатку потрібно лише пару-трійку кнопок на панелі інструментів, то простіше скористатися першим варіантом. Але якщо ви робите серйозне додаток і бажаєте, щоб у нього була професійна панель інструментів, то краще скористатися переміщуваною панеллю, як програм з MSOffice. Такі панелі можна рухати, можна відривати їх від форми і робити з них окреме вікно. Втім, давайте відразу ж напишемо приклад програми з такою панеллю.

Створіть новий додаток, встановіть на форму компонент ControlBar з вкладки Additional. Це простий компонент, працює як панель, але він дозволяє панелей інструментів переміщатися всередині себе за бажанням користувача. Властивість Align встановіть Top, а властивість AutoSize - у True. Тоді компонент буде автоматично розтягуватися або звужуватися, коли ви будете рухати панелі інструментів всередині нього.

Тепер перейдіть на вкладку Win32, і знайдіть там компонент ToolBar. Це і є панель інструментів, встановіть її поверх ControlBar. Як бачите, ControlBar одразу прийняв потрібну висоту. Якщо ви знімете виділення з цього компонента, то побачите, що він має оббирання зверху. Зазвичай на програмах такий волани немає, тому знімемо її. Властивість EdgeBorders компонента розкривається, і показує, які волани є. Встановіть ebTop в False. Тепер наша

панель інструментів має професійний вигляд, тільки кнопочок їй бракує. Бажано і тут встановити властивість AutoSize в True.

Почнемо додавати кнопки. Клацніть правою кнопкою миші на панелі інструментів і виберіть команду NewButton. З'явилася нова кнопка. Команда NewSeparator у цьому меню створює роздільники між кнопками. Видалити кнопку або розділювач просто - виділяєте її та натисніть <>Delete.

Давайте створимо таку ж панель, як у минулому прикладі. Після першої кнопки вставте сепаратор, потім ще дві кнопки. Знову ж таки, ці кнопки опуклі, а в сучасних додатках вони виглядають більш плоскими. Виділіть саму панель, і властивість Flat змініть на True, тоді всі кнопки на панелі будуть виглядати плоскими.

Тепер встановимо на форму компонент Imagelist і завантажимо ті ж три картинки: opendoor.bmp, led2on.bmp і led2off.bmp.

Тепер перейдіть до панелі інструментів, і у властивості Images виберіть наш Imagelist. Зображення зі списку автоматично завантажилися в кнопки. Якщо вам не сподобалося розподіл цих картинок, можете змінити їх, змінюючи властивість ImageIndex. Перша картинка там має індекс 0, друга - 1, і так далі. Таким чином, ви можете присвоювати кнопок різні картинки зі списку.

Давайте розглянемо деякі корисні властивості кнопки на панелі інструментів. Сама кнопка називається ToolButton, але такого компонента на панелі інструментів немає, її можна завантажити, натиснувши на панелі інструментів правою кнопкою миші і вибравши команду NewButton. Кнопка по своїм властивостям схожа з кнопкою SpeedButton. Розглянемо їх детальніше.

**AllowAllUp**. Якщо встановлено у True, то кнопка синхронізує свій стан з іншими кнопками групи - у будь-який момент може бути натиснута тільки одна кнопка групи. Це властивість працює тільки в тому випадку, якщо властивість Grouped (угрупування) у кнопки також встановлено у True. Виділіть два останніх кнопки і встановіть у нього обидва ці властивості в True.

**Caption**. Містить напис на кнопці, яка буде виходити разом з зображенням, якщо у самої панелі інструментів властивість ShowCaptions встановити в True.

У першій кнопки в цій властивості напишіть «Вихід», у другий - «Активна», а в третій «Неактивна». Що там буде активно чи ні, не має значення, зараз ми просто вчимося працювати з панеллю інструментів. Тепер виділіть саму панель інструментів, і встановіть властивість ShowCaptions в True. Як бачите, кнопки стали великими, і разом з зображенням на них виходить також і текст. У деяких програмах ви можете зустріти такі панелі інструментів. Знову поверніть це властивість в False. Щоб повернути кнопок початковий розмір, виділіть першу кнопку і змініть її розміри. Встановіть властивості Height і Width рівним 23.

**Down**. Як і раніше, це властивість відповідає за стан кнопки - натиснута вона чи ні. Щоб повторити попередній приклад, встановіть у першій сгруппированной кнопки Down у True.

**ImageIndex**. З цією властивістю ми вже зрозуміли, воно відповідає за картинку, яка зображена на кнопці.

**Style**. Стиль кнопки. Мабуть, найцікавіше властивість. Якщо ви скомпилируете приклад, то побачите, що при натисканні на кнопку, вона повертається в віджате стан. А як бути, якщо нам, як у минулому прикладі, потрібно, щоб завжди була натиснута тільки одна кнопка з групи? Ми вже згрупували дві останні кнопки і вказали їм синхронізувати стан з іншими кнопками групи. Тепер знову виділіть їх, а у властивості Style виберіть tbsCheck. Цей стиль дозволяє кнопці залишатися в натиснутому стані. Щоб подолати її, потрібно буде знову клацнути по кнопці. Якщо ж кнопки згруповані, як у нашому прикладі, то натискання на іншу кнопку буде домагатися своєї першу.

Сама панель інструментів також має ряд цікавих властивостей, які потрібно знати. Виділіть її, і подивіться на Інспектор Об'єктів. Такі властивості, як AlignWidth, Height і ми розглядати не будемо, оскільки знайомі з ними по інших компонентів.

**AutoSize**. Якщо Так, то панель автоматично вирівнює висоту, враховуючи висоту кнопок.

**ButtonHeight**. Визначає висоту кнопок, створюваних на цій панелі.

**ButtonWidth**. Визначає ширину кнопок на панелі. Щоб кнопки були квадратними, залишайте ці властивості рівними один одному.

**Caption**. Назва панелі інструментів, яке буде видно, якщо зняти панель інструментів з місця і зробити з неї окреме вікно. Вкажіть у цьому властивості «Файл».

**Flat**. Якщо одно True, то кнопки виглядають сучасно, без опуклостей.

**List**. Працює, якщо ShowCaprions встановлений в True. Якщо одно True, то зображення пригорнеться до лівої межі, а текст - до правого. Інакше зображення буде зверху, а текст - знизу кнопки.

**ShowCaprions**. Дозволяє або забороняє показ тексту на кнопках.

Крім того, ми знаємо, що кнопки можуть мати різні стани, і бути активними або неактивними (це залежить від властивості Enabled). Звичайна кнопка в неактивному стані має малюнок сірого тону. Кнопки мають три варіанти зображення - звичайне, неактивна, і коли над кнопкою розташовується вказівник миші. В один контейнер Imagelist не вдається завантажити зображення різних станів кнопок. Якщо ви бажаєте використовувати всі три стани, то вам доведеться встановлювати три контейнери Imagelist для зображень. У кожен контейнер ви додаєте картинку свого стану, важливо, щоб ці картинки мали однаковий індекс, тобто, щоб вони відповідали один одному по черговості в списку. Далі, ви встановлюєте:

**Images**. Тут ви вказуєте контейнер з звичайним зображенням кнопки, у нас це ImageList1. DisabledImages. Тут ви вказуєте контейнер з зображеннями недоступних кнопок. HotImages. Тут вказується контейнер з зображеннями кнопок у момент, коли вказівник миші знаходиться над ними.

Зазвичай такі премудрості не потрібні, достатньо вказувати тільки звичайне зображення кнопок. Але знати про такі можливості компонентів потрібно.

**DragKind**. Мабуть, найбільш цікава властивість панелі інструментів. Воно може мати два варіанти - dkDrag (за замовчуванням), і dkDock. Якщо встановлено dkDrag панель можна буде переміщати тільки всередині ControlBar. Для цього потрібно покажчиком миші вхопитися за вертикальну

риску в лівій частині панелі і переміщати її. А ось якщо встановити в цій властивості dkDock панель інструментів можна буде зняти з ControlBar, встановити її всередині вікна програми або навіть за її межами. Спробуйте це зробити.

Тепер ще один цікавий приклад панелі інструментів. Кнопки на панелі можна пов'язувати з головним або спливаючим меню! Тобто, якщо у вас є головне меню, і там є команда «Вихід», яку ви вже запрограмували, то немає потреби писати цей код кнопки, досить пов'язати її з компонентом головного меню.

У нас немає коду обробника події кнопки «Вихід». Якщо є - видаліть. Встановіть на форму головне меню. Вкажіть там розділ «Файл» і підрозділи «Відкрити», «-» і «Вихід». Зв'яжіть меню з нашим Imagelist, і пункту «Вихід» присвойте картинку opendoor.bmp. Тепер створіть обробник подій для цього пункту і пропишіть там

```
Close;
```

Збережете, зберіть його, і подивіться, як він працює. Тепер виберіть кнопку «Вихід» на панелі інструментів, і у властивості MenuItem виберіть пункт меню, з яким потрібно асоціювати кнопку (у мене це N4). Якщо ви все зробили правильно, то при роботі програми, якщо користувач вибере команду «Вихід» в меню або натисне кнопку «Вихід» на панелі інструментів, виконається один і той же код.

Точно також кнопки можна пов'язати з спливаючим меню, за це відповідає властивість PopupMenu, в якому ви можете вибрати спливаюче меню. Властивість DropDownMenu пов'язує кнопку з допоміжним меню, якщо воно є.

## СПИСОК ЛІТЕРАТУРИ


1. Кандзюба С.П. Громов В.Н. Delphi 5. Базы данных и приложения. Лекции и упражнения. – К.: ДиаСофт, 2001. - 592 с.
2. Архангельский А.Я. Разработка прикладных программ для WINDOWS в Delphi 5. – М.: Бином, 1999. – 256 с.
3. Delphi: Специальный справочник. – СПб.: Питер, 2001. – 688 с.
4. Шпак Ю.А. Разработка приложений в Delphi 2005/2006 – К.: «МК-Пресс», Киев, 2007 – 544.
5. Архангельский А.Я. Програмирование в Delphi 5. – М.: Бином, 2000. – 1072 с.
6. Delphi 7/ А. Хоменко, В. Гофман, Е. Мещеряков, В.Никифоров, Под ред. А. Хоменко. – СПб.: БХВ – Петербург, 2003. – 1216 с.
7. Кафаров В.В., Мешалюн В.П., Перов В.Л. Математические основы автоматизированного проектирования химических производств. – М.: Химия, 1979. – 320 с.
8. Бояринов А.И., Кафаров В.В. Методы оптимизации в химической тенологии. - М.: Химия, 1985. – 576 с.
9. Кафаров В.В. Методы кибернетики в химии и химической технологии. - М.: Химия, 1985 – 448 с.
10. Петренко А.И., Семенков О.И. Основы построений автоматизированого проектирования. – Киев. Вища школа, 1985. – 94 с.
11. Бузя А.П., Кононюк А.Е., Куценко Г.П. и др. Справочник по САПР. – К.: Техника, 1988. – 375 с.
12. Лукач Ю.Е., Воронин Л.Г., Ружинская Л.И. и др. Автоматизированное проектирование валковых машин для переработки полимеров. – М.: Теюижр, 1988. – 209 с.
13. Тревский Б.А. Методические указания по применению вычислительной техники по курсу “Машины и аппараты химических производств” и алгоритмы расчета фильтров. – К.: КПИ, 1981. – 27 с.

14. Радченко Л.Б. Методические указания по применению вычислительной техники по курсу “Процессы и аппараты химических производств”. Раздел “Алгоритм расчета пленочного теплообменника”. – К.: КПИ, 1982. – 15 с.
15. Тананайко Ю.М. Методические указания по применению вычислительной техники по курсу “Машины и аппараты химических производств” (Алгоритм расчета пленочного аппарата. Абсорбер насадочного типа). – К.: КПИ, 1981. – 19 с.
16. Казачинская Н.В., Сидоренко С.В. Методические указания по применению вычислительной техники по курсу “Процессы и аппараты химических производств” (Алгоритм расчета пластинчатого теплообменника). – К.: КПИ, 1983. – 24 с.
17. Марчевский В.Н. Методические указания по применению вычислительной техники по курсу “Машины и аппараты химических производств” (Алгоритм расчета аппаратов для сушки дисперсных материалов в псевдоожиженном слое). – К.: КПИ, 1981. – 16 с.
18. Тананайко Ю.М. Методические указания по применению вычислительной техники по курсу “Процессы и аппараты химических производств” (Алгоритм расчета насадочной ректификационной коллоны). – К.: КПИ, 1983. – 23 с.
19. Гаевский Б.А. Методические указания по применению вычислительной техники по курсу “Машины и аппараты химических производств” (Алгоритм теплового расчета контактной сушилки). – К.: КПИ, 1984. – 20 с.
20. Корниенко Я.Н., Яременко В.И., Коннов В.А. Методические указания по применению вычислительной техники по курсу “Машины и аппараты химических производств” (Алгоритм расчета аппаратов для сушки пастообразных пордуктов). – К.: КПИ, 1988. – 24 с.
21. Тананайко Ю.М., Воронцов В.Г., Шевчук Ю.В. Методические указания по применению вычислительной техники по курсу “Машины и аппараты химических производств” (Алгоритм технологического, конструктивного и

энергетического расчета роторно-пленочного аппарата в качестве испарителя). – К.: КПИ, 1988. – 48 с.

22. Лукач Ю.Е., Доброногова С.И., Ружинская Л.И. Методические указания по применению вычислительной техники по курсу “Машины и аппараты химических производств” (Алгоритм расчета устройств для термообработки изделий из термопластов). – К.: КПИ, 1984. – 84 с.