Міністерство освіти і науки України
Полтавський національний педагогічний університет імені В.Г. Короленка

Кривцова Олена Павлівна

навчальний посібник

Інформатика.
Основи програмування у середовищі Microsoft Visual C++ Express

м. Полтава – 2018

УДК 519.682(075)

ББК 32.973.26-018.1 я 73

К 82

Кривцова О.П. Інформатика. Основи програмування у середовищі Microsoft Visual C++Express [Електронний ресурс] : навч. посіб. / О.П. Кривцова. – Полтава: ПНПУ імені В.Г. Короленка, 2018. – 161с.

Навчальний посібник «Інформатика. Основи програмування у середовищі Microsoft Visual C++ Express» висвітлює основні прийоми програмування мовою C++. Розглянуто опис типів даних, оголошення змінних, організація розгалужень та циклів, опис та використання масивів, покажчиків, функцій та описано основи роботи у візуальному середовищі програмування.

У першій частині посібника викладено теоретичні основи програмування мовою C++ та основи роботи у середовищі Microsoft Visual C++ Express. На простих прикладах показано особливості розв'язування задач з програмування. Друга частина містить завдання до лабораторних робіт з прикладами розробки додатків у середовищі Microsoft Visual C++Express.

Рецензенти:

Ємець О.О. - доктор фіз.-мат. наук, завідувач кафедри математичного моделювання та соціальної інформатики Полтавського університету економіки і торгівлі

Барболіна Т.М. - кандидат фіз.-мат. наук, доцент, завідувач кафедри математичного аналізу та інформатики Полтавського національного педагогічного університету імені В.Г. Короленка

Рекомендований Вченою радою Полтавського національного педагогічного університету імені В.Г. Короленка (протокол №16 від 27.06.2018р.)

ЗМІСТ

ПЕРЕДМОВА 5

ЛЕКЦІЙНИЙ КУРС 6

ТЕМА 1. ОСНОВИ АЛГОРИТМІЗАЦІЇ ПРОЦЕСІВ ОБРОБЛЕННЯ ІНФОРМАЦІЇ 6

Алгоритм і його властивості 6
Схеми алгоритмів 7
Графічне зображення різних видів обчислювальних процесів 9

ТЕМА 2. ІНСТРМЕНТАЛЬНІ МОВИ ТА СИСТЕМИ ПРОГРАМУВАННЯ 12

Покоління та класифікація мов програмування 12
Компоненти інструментальних систем 13

ТЕМА 3. МОВА ПРОГРАМУВАННЯ С++ 15

Історія виникнення 15
Основні етапи розробки програм мовою С++ 16
Елементи мови С++ 17

ТЕМА 4. БАЗОВІ ТИПИ ДАНИХ 20

Перетворення типів 22

ТЕМА 5. ОСНОВНІ ОПЕРАЦІЇ МОВИ С++ 23

Арифметичні операції 23
Операції присвоювання 24
Операції відношення 25
Логічні операції 25
Порозрядні операції (побітові операції) 25
Операція слідування 26
Умовна операція 26

ТЕМА 6. СТРУКТУРА ПРОГРАМИ МОВОЮ С++ 27

Функції введення та виведення 28

ТЕМА 7. ПРОГРАМУВАННЯ РОЗГАЛУЖЕНЬ У МОВІ С++ 30

Оператор IF 30
Оператор IF-ELSE 30
Оператор вибору SWITCH 31

ТЕМА 8. ЦИКЛИ В МОВІ С++ 32

Оператор циклу з передумовою 32
Оператор циклу з постумовою DO...WHILE 33
Оператор циклу FOR 33
Оператор розриву 34
Оператор продовження 34

ТЕМА 9. ФУНКЦІЇ 36

ТЕМА 10. МАСИВИ В МОВІ С++ 39

Одновимірні масиви 39
Багатовимірні масиви 40
Покажчики 42
Динамічне виділення масивів 45

ТЕМА 11. РЯДКИ ТА ОПЕРАЦІЇ З НИМИ 47

Масиви символів 47
Стандартний клас String 48

ТЕМА 12. СТРУКТУРИ 50

ТЕМА 13. ГОЛОВНІ ПРИНЦИПИ ОБ'ЄКТНО - ОРІЄНТОВАНОГО СТИЛЮ ПРОГРАМУВАННЯ 52

Головні елементи об'єктно-орієнтованого програмування 52
Технологія візуального програмування 53

ТЕМА 14. ВІЗУАЛЬНЕ СЕРЕДОВИЩЕ РОЗРОБКИ ПРОГРАМ МОВОЮ С++ 54

Середовище Microsoft Visual C++ Express 54
Етапи розробки додатків 57

ТЕМА 15. ІНСТРУМЕНТИ ВІЗУАЛЬНОЇ РОЗРОБКИ ДОДАТКІВ 60

Особливості роботи з формами 60
Характеристика основних компонентів середовища 61
Компонент Button 62
Компонент Label 63
Компонент TextBox 64
Компоненти CheckBox і RadioButton 67
Компонент ComboBox 69
Компонент ListBox 70

ТЕМА 16. РОЗРОБКА ГРАФІЧНОГО ІНТЕРФЕЙСУ КОРИСТУВАЧА 72

Компонент MenuStrip 72
Компонент ContextMenuStrip 74
Компонент ToolStrip 74
Стандартні діалогові вікна 75
Компонент OpenFileDialog 76
Компонент SaveFileDialog 76

ТЕМА 17. ГРАФІКА У MICROSOFT VISUAL C++ EXPRESS 78

Компонент PictureBox 78
Виведення графічних зображень 79
Олівці і пензлі 80
Графічні примітиви 83
Лінія 85
Ламана лінія 85
Прямокутник 86
Точка 87
Багатокутник 88
Еліпс і коло 89
Дуга 90
Сектор 90
Текст 91
Метод базової точки 92
Анімація 92

Лабораторна робота №1. ОЗНАЙОМЛЕННЯ З ОСНОВАМИ МОВИ С++. КОМПІЛЯЦІЯ ТА ЗАПУСК ПРОГРАМИ НА ВИКОНАННЯ 94

Лабораторна робота №2. ПРОГРАМУВАННЯ РОЗГАЛУЖЕНЬ У МОВІ С++ 102

Лабораторна робота №3. РОЗРОБКА ПРОГРАМ МОВОЮ ПРОГРАМУВАННЯ С++ З ВИКОРИСТАННЯМ ЦИКЛУ «З ПЕРЕДУМОВОЮ» ТА «З ПІСЛЯУМОВОЮ» 107

Лабораторна робота №4. РОЗРОБКА ПРОГРАМ, ЩО МІСТЯТЬ ЦИКЛ FOR 111

Лабораторна робота №5. ФУНКЦІЇ У МОВІ С++ 115

Лабораторна робота №6. ОБРОБКА ОДНОВИМІРНИХ МАСИВІВ У МОВІ С++ 117

Лабораторна робота №7. ОБРОБКА БАГАТОВИМІРНИХ МАСИВІВ У МОВІ С++ 121

Лабораторна робота №8. ПРЕДСТАВЛЕННЯ СИМВОЛЬНИХ ДАНИХ У МОВІ С++ 125

Лабораторна робота №9 СТРУКТУРИ 130

Лабораторна робота №10. ОЗНАЙОМЛЕННЯ З ВІЗУАЛЬНИМ СЕРЕДОВИЩЕМ VISUAL C++. КОМПІЛЯЦІЯ ТА ЗАПУСК ПРОГРАМИ НА ВИКОНАННЯ. 133

Лабораторна робота №11. ІНСТРУМЕНТИ ВІЗУАЛЬНОЇ РОЗРОБКИ ДОДАТКІВ. 137

Лабораторна робота №12 РОЗРОБКА ДОДАТКІВ З ВИКОРИСТАННЯМ ЕЛЕМЕНТІВ УПРАВЛІННЯ. 147

Лабораторна робота №13 РОЗРОБКА ІНТЕРФЕЙСУ КОРИСТУВАЧА ПРИКЛАДНИХ ДОДАТКІВ 150

Лабораторна робота №14 ОСОБЛИВОСТІ РОБОТИ З ГРАФІЧНИМИ ПРИМІТИВАМИ 155

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 161

ПЕРЕДМОВА

Важливим етапом підготовки майбутніх вчителів інформатики є формування знань вмінь та навичок програмування у межах існуючих інструментальних систем. Для цього студентам пропонується вивчення основ програмування мовою С++ та знайомство з концептуальними аспектами об‘єктно-орієнтованого програмування.

С++ - універсальна мова програмування, яка надає гнучкі і ефективні засоби розв’язування задач та розробки програмних продуктів. У С++ збережені можливості мови Сі по роботі з основними об'єктами апаратного забезпечення.

Навчальний посібник призначений головним чином для того, щоб допомогти майбутнім учителям вивчити мову програмування і застосовувати її при розробці власних проектів. У посібнику дано ґрунтовний опис мови С++, багато прикладів і фрагментів програм. Основна увага приділена основам роботи у візуальному середовищі програмування Microsoft Visual C++ Express.

Теоретична частина є курсом лекцій з мови програмування С++. Лабораторні роботи згруповані по темах і охоплюють частину базового курсу «Інформатика»: від оголошення змінних і програм з лінійною структурою до розробки прикладних додатків у візуальному середовищі програмування.

Перша частина посібника призначена, для ознайомлення з теоретичними основами мови С++ та дозволяє навчитися розробляти ефективні додатки. У якості середовища розробки використано Microsoft Visual C++ Express. Друга частина містить блок лабораторних робіт з основ програмування мовою С++.

Даний посібник складається з 17 тем і 14 лабораторних робіт і може бути використаний при вивченні дисциплін «Інформатика» студентами педагогічних університетів, а також для самоосвіти.

ЛЕКЦІЙНИЙ КУРС

ТЕМА 1. ОСНОВИ АЛГОРИТМІЗАЦІЇ ПРОЦЕСІВ ОБРОБЛЕННЯ ІНФОРМАЦІЇ

Алгоритм і його властивості

Одним із базових понять інформатики й ОТ є поняття алгоритму як деякого правила перетворення інформації. Він указує, які операції оброблення даних і в якій послідовності необхідно виконати, щоб одержати розв'язок задачі.

Слово "алгоритм" виникло від імені узбецького математика Хорезмі (по-арабськи Ал-Хорезмі), який у IX ст. до н.е. розробив правила чотирьох арифметичних дій над числами в десятковій системі числення. Сукупність цих правил в Європі стали називати "алгоризм", що пізніше перетворилось в слово "алгоритм" і стало збірною назвою окремих правил визначеного виду (і не тільки в математиці). Протягом значного часу його використовували тільки математики, визначаючи правила розв'язування різних задач. У 30-х роках XX століття поняття алгоритму стало об'єктом математичного вивчення (раніше ним тільки користувались), а з появою ЕОМ воно стало широко відомим. Стало зрозумілим, що розробка алгоритму є необхідним етапом розв'язування задачі на ЕОМ. Сьогодні слово "алгоритм" вийшло за рамки математики. Його стали використовувати в різних областях, розуміючи під ним точно сформульоване правило для досягнення необхідного результату. Формування наукового поняття алгоритму не закінчено і в даний час.

Приклади побутових алгоритмів: інструкція на телефон-автомат; рецепт застосування ліків; рецепт годування дитини; рецепт приготування кави і т.п.

Життя кожної людини - це реалізація алгоритмів різного роду (побутових, наукових, технічних, лікувальних тощо) Так що ж таке алгоритм?

Кожен алгоритм повинен мати деякі вхідні дані, які поступово при його реалізації перетворюються в шукані результати (для алгоритмів додавання a + b = c: a, b - вхідні дані, c - шуканий результат).

Алгоритм – точне розпорядження, що визначає обчислювальний процес, який веде від початкових даних, що змінюються, до шуканого результату.

Алгоритм має такі основні властивості:

детермінованість (визначеність) - однозначність результату обчислювального процесу при заданих початкових даних;

дискретність - розчленованість обчислювального процесу на окремі елементарні кроки, можливість виконання яких не викликає сумніву;

масовість - забезпечення розв'язання будь-якої задачі з класу однотипних;

результативність - забезпечення одержання результату через кінцеве число кроків.

формальність. Будь-який виконавець, здатний сприймати і виконувати вказівки алгоритму (навіть не розуміючи його змісту), діючи за алгоритмом, може виконати поставлене завдання. Ця властивість має особливе значення для автоматизації виконання алгоритмів.

Схеми алгоритмів

Для представлення алгоритмів використовують такі способи, як словесний опис послідовності обчислень, аналітичний (у вигляді формул), графічний (у вигляді схем і діаграм), псевдокод, запис алгоритмічною мовою.

Прикладом словесного опису алгоритму є кулінарний рецепт, що вказує певну послідовність дій.

Запис алгоритму алгоритмічною мовою потребує точного дотримання правил цієї мови, оскільки він має бути зрозумілим не тільки людині, а й комп'ютеру.

Псевдокод займає проміжне місце між словесним описом алгоритму і його записом алгоритмічною мовою. У цьому способі вживаються конструкції, близькі до алгоритмічної мови, але не вимагається повного дотримання всіх її правил, оскільки він призначений для сприйняття людиною.

Велике поширення дістав графічний спосіб представлення алгоритму у вигляді схем.

Схема алгоритму — графічне зображення його структури, в якому кожний етап процесу перероблення даних подається у вигляді різних геометричних фігур **(**символів).

Ці фігури з'єднуються між собою лініями потоку, які для кожного етапу вказують можливих наступників. Усередині фігури дається опис відповідного етапу, якщо він не є занадто громіздким. У противному разі такий опис наводиться в додатку до схеми, а замість нього у відповідній фігурі записується номер або яке-небудь позначення цього етапу. Біля фігури можуть бути деякі зауваження, наприклад такі, що показують, в якому випадку вибір наступника буде робитися відповідно до лінії потоку.

Символам присвоюють порядкові номери, які проставляються в розриві лінії контуру в лівій частині верхнього боку зображення символу. Лінії потоку проводять паралельно лініям зовнішньої рамки схеми. Напрямок лінії потоку зверху вниз і зліва направо прийнято за основний і, якщо вони не мають зламів, стрілками їх можна не позначати. В інших випадках їхній напрямок обов'язково позначають стрілкою. Лінію потоку, як правило, підводять до середини символу.

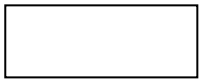
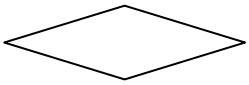
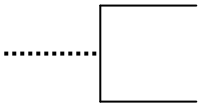
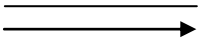
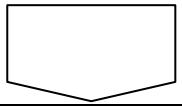
Відстань між паралельними лініями потоку має бути не меншою від 3 мм, між іншими символами - не меншою від 5 мм. Лінію потоку можна обривати, використовуючи на місці обриву з'єднувачі, якщо схему виконано на двох і більше аркушах, або якщо символи, які з'єднуються, розташовано на значній відстані один від одного.

Запис усередині символу або поруч із ним потрібно виконувати машинописом з одним інтервалом або креслярським шрифтом.

Перевагою схем є те, що з їх допомогою можна наочно зобразити структуру алгоритму в цілому, відобразивши його логічну суть (показати розгалуження шляхів розв'язання задачі залежно від виконання деякої умови, відобразити багаторазове повторення окремих етапів обчислювального процесу). Особливо це важливо для задач економічного характеру і задач управління. Вони містять велику кількість операцій порівняння, логічних, арифметичних й інших операцій, і тому відразу важко встановити їх послідовність у процесі розв'язування задачі.

Графічне зображення алгоритму у вигляді схем полегшує складання програми для розв'язання задачі на комп'ютері (*У таб.1 наведено елементи, що найчастіше використовуються в схемах алгоритмів.*).

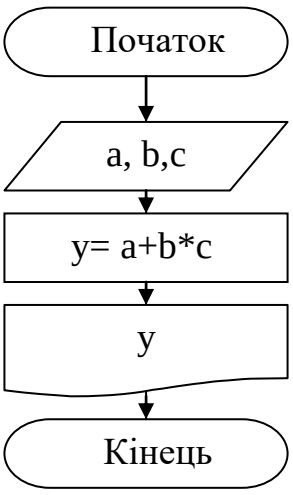
Таблиця 1. Елементи блок-схеми

Процес		Виконання дії або групи дій, завдяки яким змінюється значення, форма подання або розташування даних
Розв‘язування		Вибір напрямку виконання алгоритму залежно від деяких змінних умов
Введення - виведення		Перетворення даних у форму, придатну для оброблення (введення) або відображення здобутих результатів (виведення)
З‘єднувач		Зазначення зв‘язку між перерваними лініями потоку, що зв‘язують символи
Пуск-зупинка		Початок, кінець переривання процесу оброблення даних або виконання програми
Коментар		Зв‘язок між елементом схеми і пояснення
Лінія потоку		Зазначення послідовності зв‘язків між символами
Міжсторінковий з‘єднувач		Зазначення зв‘язку між роз‘єднаними частинами схем алгоритмів і програм, розташованих на різних аркушах

Графічне зображення різних видів обчислювальних процесів

Обчислювальні процеси, що виконуються за заданим алгоритмом, поділяють на три основні види**: лінійні; розгалужені; циклічні.**

Вони, як правило, є окремими частинами обчислювального процесу, тоді як загальний обчислювальний процес має складнішу (комбіновану) структуру.



Графічне зображення лінійних обчислювальних процесів

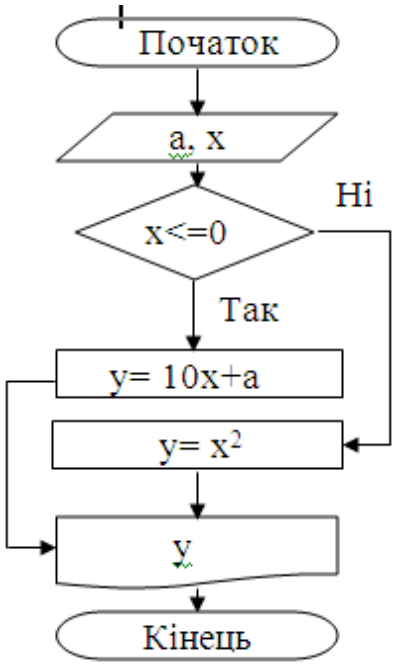
У лінійному обчислювальному процесі всі операції виконуються послідовно у порядку їх запису.

Типовим прикладом такого процесу є стандартна обчислювальна схема, що складається з трьох етапів:

- введення початкових даних;
- обчислення за формулами;
- виведення результату.

Приклад *Процес обчислення виразу* ***Y= a+b*c***

Графічне зображення розгалужених обчислювальних процесів



Обчислювальний процес називається **розгалуженим,** якщо для здобуття кінцевого результату передбачається вибір одного з кількох можливих напрямів обчислень (гілок) залежно від результату перевірки деякої умови.

Розгалужений обчислювальний процес, що складається з двох гілок, називається *простим,* а з більшої кількості гілок - *складним.* Напрям обчислень вибирається перевіркою, внаслідок якої можливі два виходи: «Так» - умову виконано; «Ні» - умову не виконано.

Приклад *розгалуженого алгоритму, що описує процес обчислення виразу*

$$y = \begin{cases} 10x + a, \text{ якщо } _ x \le 0 \\ x^2, \text{ якщо } _ x > 0 \end{cases}$$

Графічне зображення циклічних обчислювальних процесів

Для більшості обчислювальних процесів характерною є повторюваність дій.

Циклом називається послідовність дій, що багаторазово повторюється, а обчислювальний процес, який містить цикл, має назву циклічного.

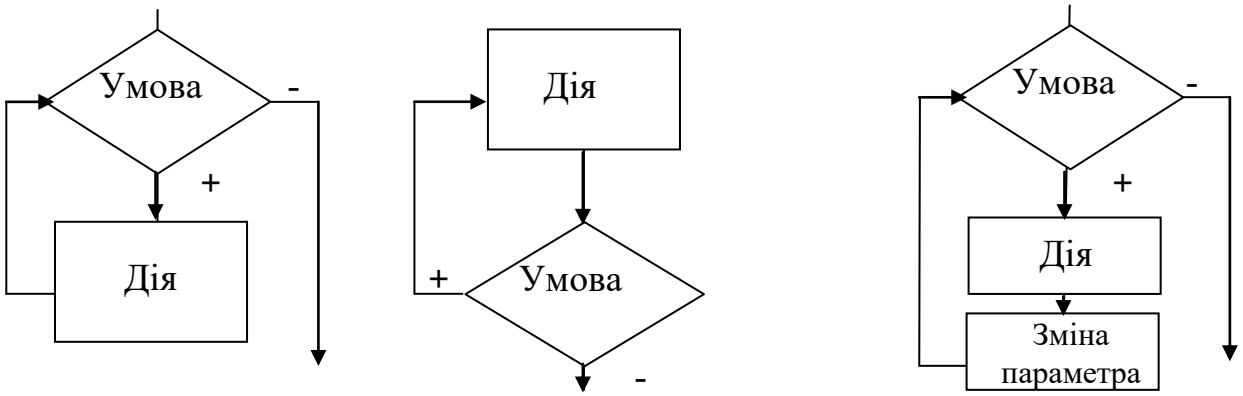
Керування повторенням циклу здійснюється за допомогою змінної, яка називається параметром циклу. Спочатку цьому параметру присвоюється деяке початкове значення. Потім цикл виконується зі Зміною параметра при кожному повторенні від початкового до кінцевого значень на величину, що називається кроком циклу.

Крок циклу може бути позитивним або негативним. Залежно від цього параметр циклу зростає або зменшується. Цикл припиняється, якщо параметр циклу має значення, що лежить поза межами діапазону між початковим і кінцевим значеннями.

Розрізняють три види циклів. Перші два види циклів використовуються тоді, коли заздалегідь невідома кількість повторень.

<u>У циклі з передумовою</u> спочатку перевіряється умова (звідси і назва — цикл з передумовою) і, якщо умова виконується, то здійснюється дія. Потім знову перевіряється умова і т. д. Виконання циклу припиняється, коли умова перестає виконуватися. Для цього необхідно, щоб дія в циклі впливала на зміну умови. Інакше відбудеться "зациклювання" — нескінченне виконання циклу. Воно є типовою помилкою в разі використання циклів.

з передумовою; **з післяумовою;** **з параметром.**



<u>Цикл з післяумовою</u> виконується аналогічно, але умова перевіряється після виконання дії (тому цикл і називається з післяумовою). Дія в циклі з постумовою виконується завжди хоча б один раз, а з передумовою може не виконуватися жодного разу, якщо з самого початку умова хибна.

<u>Цикл із параметром</u> будується на підставі одного з перших двох видів циклів.

ТЕМА 2. ІНСТРМЕНТАЛЬНІ МОВИ ТА СИСТЕМИ ПРОГРАМУВАННЯ

Покоління та класифікація мов програмування

Мови програмування прийнято поділяти на п'ять поколінь. У перше покоління входять мови, які створено на початку 50-х років, коли з'явилися перші комп'ютери. Першою мовою був **асемблер,** створений за принципом «одна інструкція - одна команда».

Друге покоління мов програмування з'явилося у кінці 50-х на початку 60-х років, коли було створено символьний асемблер, у якому з'явилося поняття змінної. Він став першою повноцінною мовою програмування.

Третє покоління мов програмування з'явилося у 60-ті роки. У цей час виникли універсальні мови високого рівня, за допомогою яких стало можливим розв'язання задач у різних областях. Такі якості нових мов, як відносна простота, незалежність від конкретного комп'ютера та можливість використання складних синтаксичних конструкцій, полегшили роботу програмістів.

Починаючи з 70-х років почався період розвитку мов програмування четвертого рівня. Ці мови програмування призначені для реалізації складних проектів та орієнтовані на спеціалізовані області застосування, де позитивний результат досягається лише при використанні проблемно-орієнтованих мов які оперують конкретними поняттями вузької предметної області.

Народження мов п'ятого покоління припадає на 90-ті роки. До них відносяться також системи автоматичного створення прикладних програм за допомогою візуальних засобів розробки, без володіння мовами програмування.

Інструментальні мови поділяють на мови низького рівня (близькі до машинної мови) та мови високого рівня (близькі до мови людини). До мов високого рівня належать:

FORTRAN - перша компілюєма мова, створена Джимом Беку сем у 50-ті роки.

ALGOL - створена у 1960 році для заміну FORTRAN, але через складну структури не отримав широкого використання.

PASCAL - мову програмування створено у кінці 70-років Ніколасом Віртом.

BASIC - створено у 60-ті роки в якості навчальної мови, дуже легкий у вивченні.

C++ (Си++) – об'єктно-орієнтована мова програмування створена у 1980 році Бьярном Страуструпом.

JAVA (Ява) — мову було створено компанією SUN на початку 90-х років на основі Си++.

Компоненти інструментальних систем

Інструментальні системи служать для розроблення програм. У загальному випадку для. створення програми на довільній мові програмування необхідно мати наступні компоненти.

Текстовий редактор. Текст програми створюється за допомогою ключових слів та стандартних символів для запису операцій. Формування тексту можна здійснювати у будь-якому текстовому редакторі, але краще використовувати спеціалізовані редактори орієнтовані на конкретну мову програмування.

Транслятор. Кожна команда що виконується комп'ютером має вигляд послідовності нулів й одиниць. Писати програми машинною мовою незручно, а їх надійність низька. Тому програми розробляють мовою, зрозумілою людині (інструментальна мова), після чого спеціальною програмою (транслятором) текст програми перекладається на машинний код (транслюється).

Транслятори бувають двох типів: **інтерпретатори і компілятори**. Інтерпретатор читає один оператор програми, аналізує його в контексті вже працюючої програми і потім його виконує, після чого переходить до оброблення наступного оператора. Компілятор спочатку читає, аналізує та перекладає на машинний код усю програму, і тільки після завершення всієї трансляції ця програма виконується.

Оскільки при інтерпретації програма виконується частинами, на комп'ютерах з малою оперативною пам'яттю можна виконати, хоча і повільно, досить великі програми. Компілятори під час аналізу всієї програми оптимізують її. З цієї причини, а також завдяки тому, що програма читається в оперативну пам'ять відразу вся, при компіляції вона виконується швидше, ніж: при інтерпретації.

Бібліотека стандартних програм що поставляються разом з компілятором та реалізує стандартні функцій.

Компонувальник. Виконує зв'язування об'єктних модулів та машинного коду стандартних функцій, знаходячи їх у бібліотеках та формує на виході працездатний додаток.

Налагоджувач. У сучасних інтегрованих системах є компонент, який дозволяє аналізувати роботу програми під час її виконання. З його допомогою можна послідовно виконувати окремі оператори вихідного тексту по крокам, спостерігаючи при цьому, як змінюються значення різних змінних.

Засоби автоматизації програмування (дизайнери, майстри). Інтегроване середовище автоматично готує всю необхідну послідовність команд, виконує їх, отримує результат, повідомляє про можливі помилки.

ТЕМА 3. МОВА ПРОГРАМУВАННЯ С++

Історія виникнення

У 1949 році у Філадельфії під керівництвом Джона Мочлі був створений „Стислий код" – перший примітивний інтерпретатор мови програмування.

У 1972 році 31-літній фахівець системного програмування фірми Bell Labs Денніс Рітчі розробив мову програмування Сі. У 1984 році французький математик Філіп Кан заснував фірму Borland International де розробляється діалект мови Сі.

Мова Сі була розроблена як мова для програмування в операційній системі UNIX. У 1983 році Американський Інститут Стандартів (ANSI) сформував Технічний комітет для створення стандарту мови Сі. На сьогодні мова С++, що з'явилася як послідовник мови Сі, підпорядковується більшості вимог стандарту.

За своїм змістом Сі – є мовою функцій. Програмування здійснюється шляхом опису функцій і звертання до бібліотечних функцій.

На початку 80-х років Бьярн Страуструп розробив мову програмування С++ побудовану на базі мови Сі що включала об'єктно-орієнтовані конструкції, такі як класи, похідні класи, віртуальні функції. У програмах написаних мовою С++ з'явилася можливість створювати власні типи даних на базі існуючих.

Назву С++ запропонував Рик Маскити у 1983 році, спочатку мова називалася (Сі з класами).

Об'єктно-орієнтоване програмування є технологією, яка базується на класифікації та абстракції об'єктів. Мова С++ реалізує три основні концепції об'єктно-орієнтованого програмування: і**нкапсуляція, наслідування, поліморфізм.**

Інкапсуляція – це об'єднання даних та функцій які їх обробляють в одному об'єкті. Метою інкапсуляції є автономність модулів, що дозволяє локалізувати наслідки зміни структур даних певного модуля.

Наслідування. Мова С++ володіє засобами, які дозволяють розробити ієрархію класів або типів об'єктів. Відповідний механізм називається наслідуванням. Наслідування дозволяє використовувати розроблені раніше класи, що забезпечує скорочення процесу розробки програмних модулів.

Поліморфізм – можна визначити, як властивість, яка дозволяє використовувати одне ім'я для позначення дій, загальних для споріднених класів. При цьому конкретизація дій, які виконуються відбувається в залежності від типу даних, що обробляються.

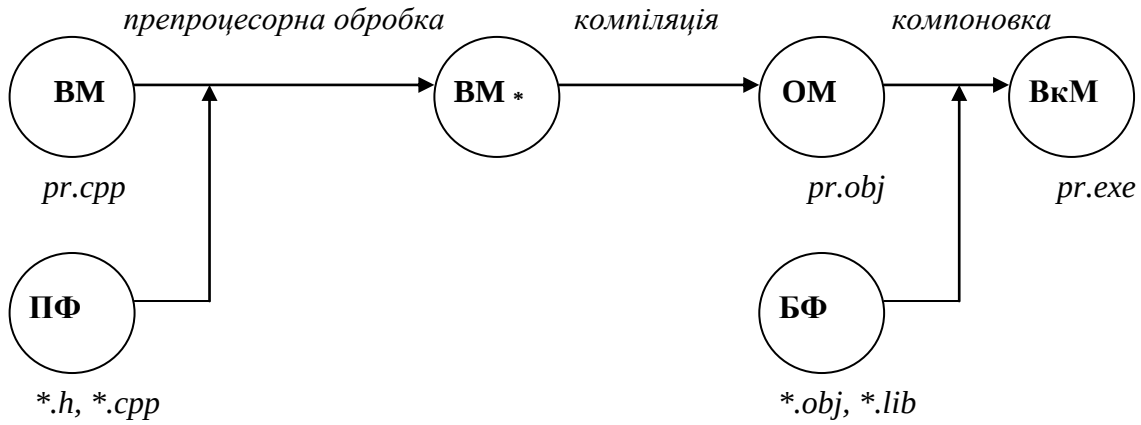
Основні етапи розробки програм мовою С++

Розробка програм мовою С++ ведеться за допомогою спеціальних комплексів програм і включає 4 етапи.

1. Написання та редагування вихідного тексту програми та збереження у вигляді вхідного файлу або модуля.
2. Компіляція програми та збереження у вигляді об'єктного файлу або модуля.
3. Побудова виконуючого файлу шляхом об'єднання отриманого об'єктного модуля програми з іншими об'єктними модулями стандартних та спеціальних бібліотек.
4. Відлагодження програми.

Інтегроване середовище програмування є програмою, яка має такі компоненти:

- вбудований редактор текстів;
- підсистему роботи з файлами;
- систему довідки
- вбудований відлагоджувач;
- підсистема керування компіляцією та редагуванням зв'язків.



Вихідний модуль (**ВМ**) – програми готується за допомогою текстового редактора та поміщається у файл з розширенням (***.cpp**). Після цього **ВМ** обробляється препроцесором та у випадку необхідності до вихідного тексту програми приєднується підключаємий файл (**ПФ**). В подальшому модернізований **ВМ*** обробляється компілятором. Відкомпільований об'єктний модуль (**ОМ**) поміщається у файл з розширенням (**pr.obj**). На наступному етапі **ОМ** обробляється компонувальником, який доповнює програму необхідними бібліотечними функціями з бібліотечних файлів (**БФ**). Отриманий модуль називається виконуючим файлом (ВкМ) та поміщається у файл з розширенням (**pr.exe**).

Препроцесор входить до складу системи програмування С++ та забезпечує обробку вхідного тексту програми перед її компіляцією.

Елементи мови С++

В алгоритмічних мовах програмування існують структурні елементи аналогічні будь якій мові (*російській, український, англійській*…).

Лексема – елементарна конструкція, яка є мінімальною одиницею мови, що має самостійний зміст.

Алфавіт мови включає :

- великі та малі літери латинського алфавіту;
- арабські цифри;
- пробільні символи (пробіл, символ табуляції, символ переходу на новий рядок);
- спеціальні символи , . ; : ? ! / \ | ~ () {} [] < > # % ^ & - + * =

Ідентифікатори. При записі ідентифікаторів:

- можуть використовуватися великі та маленькі літери латинського алфавіту, арабські цифри, символ підкреслювання;
- ідентифікатор не може починатися з цифри і не може містити пробілів;
- у якості ідентифікаторів не можуть використовуватися ключові слова;
- компілятор сприймає літери верхнього та нижнього регістрів, як різні літери. Тому можна створювати ідентифікатори, які співпадають орфографічно, але

відрізняються регістром літер. (Sum, sum, SUM, SuM, sUM – унікальні ідентифікатори);

Конструкції завершуються символом (**;**) виключенням є лише директиви пре процесора, які починаються символом (**#**), оператори та блоки визначення функції, які обмежуються фігурними дужками.

Константи. Існують чотири типи констант : цілі, дійсні, рядкові та символьні.

1. Цілі константи. Можуть бути десятковими, вісімковими, шіснадцятковими.

Десяткова – послідовність десяткових цифр від „0” до „9” яка не починається з нуля, якщо це число не нуль.

Вісімкові – починаються з символу „0” , після якого розміщуються вісімкові цифри від „0” до „7”.

Шіснадцяткові – починаються із символу „0X”, після яких розміщуються цифри від „0” до „F”.

2. Дійсні константи. Складаються з цілої частини, десяткової крапки, дробової частини, символу експонента, показника степеня. У записі констант можуть бути опущені ціла чи дробова частка.

3. Символьні константи. Один або декілька символів, які заключні в апострофи. Якщо константа складається з одного символу, то вона займає у пам’яті 1 байт (тип **char**). Двосимвольні константи займають у пам’яті 2 байти (тип **int**).

Послідовність символів, які починаються з символу зворотній шлеш „\” називаються керуючими або escape – послідовностями.

Таблиця 2. Escape-послідовності

Спеціальний символ	Шістнадцятковий код	Значення
\n	0A	перехід на наступний рядок
\r	0D	повернення каретки
\t	09	горизонтальна табуляція
\v	0B	вертикальна табуляція
\\	5C	символ \
\’	27	символ ‘
\”	22	символ “
\?	3F	символ ?

4. Рядкові константи. Записуються як послідовність символів, заключних в подвійні лапки.

Наприклад: „ рядкова константа **\n** ”

Для формування рядкових констант, які займають декілька рядків тексту програми використовується символ „\” зворотній слеш.

Наприклад: „ Довгий рядок можна розбити \
на частини”

Загальна форма визначення констант має вигляд:

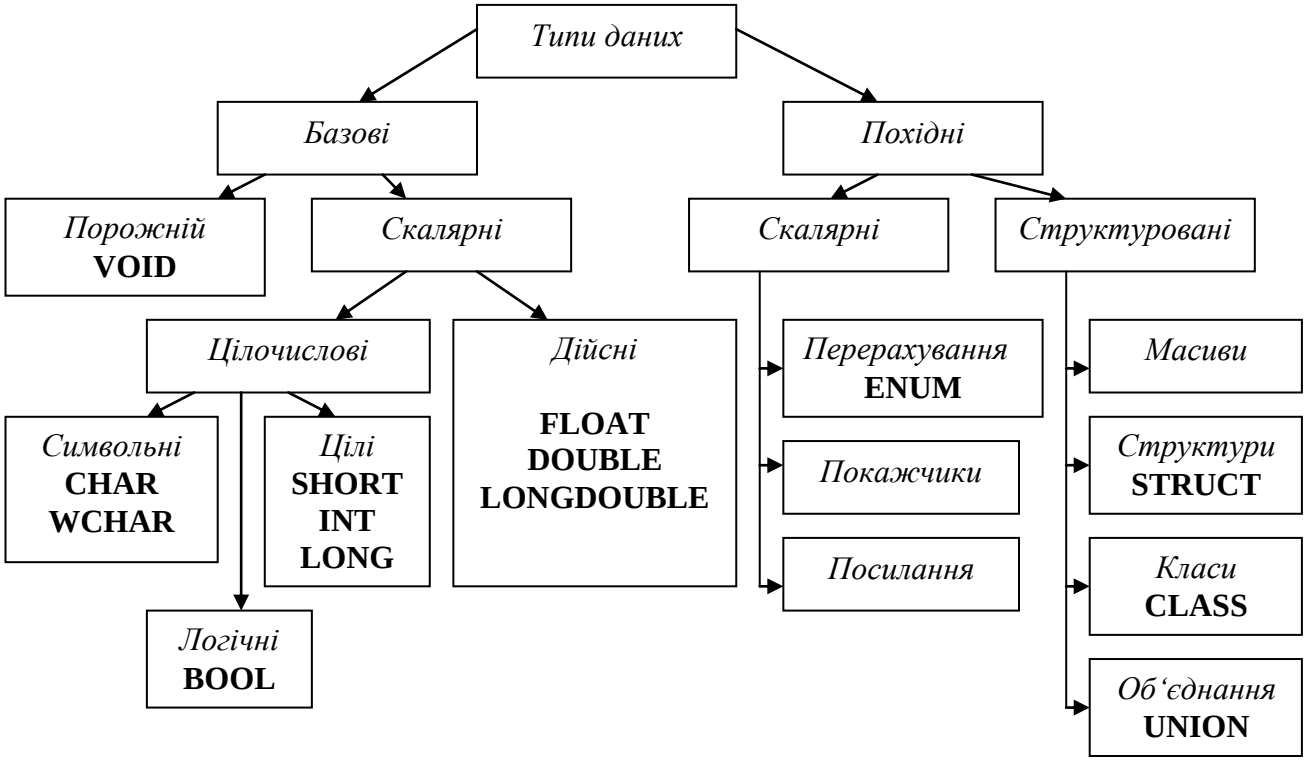
CONST <*тип*> <*ім’я*> = <*значення*>

Модифікатор **CONST** попереджує будь яке присвоєння даному об’єкту, а також інші дії, що можуть вплинути на зміну значення.

Коментар. Текст що міститься у дужках **/* текст */** ігнорується компілятором, тобто вважається коментарем до програми. При створенні однорядкового коментаря використовується символ **// текст.**

ТЕМА 4. БАЗОВІ ТИПИ ДАНИХ

Під типом даних розуміють сукупність допустимих значень даних та множину дозволених операцій над ними. Одночасно тип даних визначає розмір пам'яті, яку займають змінні або константи даного типу.



В мові С++ виділяють наступні категорії типів:

1. Базові типи
2. Похідні типи

До базових типів відносяться:

- Скалярні типи;
- Пустий тип (**void**).Не має значення та введений для опису функцій, які не повертають значення.

Скалярні типи поділяються на:

- Цілочислові (логічні, символьні, цілі).

char (символ**) –** може використовуватися для зберігання літер, цифр або будь якого символу. Значення об'єкту типу **char** є код символу. Тип **char** інтерпретується як однобайтове ціле з областю значень від -128...127.

int (ціле) – цілі числа в діапазоні від -32768...32767. Як різновид цілих чисел у деяких версіях компілятора існують **short** (коротке ціле) та **long** (довге ціле).

bool (логічний) – самостійний тип даних, який займає у пам'яті 1 байт, та використовується у логічних операціях. Може приймати значення нуль „0" (хибно, false) чи відмінне від нуля (істина, true).

- Дійсні.

float (дійсне число з плаваючою комою одинарної точності) – може представляти числа як у фіксованому форматі так і експоненціальному.

double (дійсне число з плаваючою комою подвійної точності) – має значно більший діапазон порівняно з типом **float.**

Похідні типи утворюються на основі базових та діляться на

- Скалярні (перерахування, вказівними, посилання);
- Структуровані (масиви, структури, об'єднання, класи).

Таблиця 3. Діапазон значень простих типів даних

Тип	Діапазон значень	Розмір (байт)
char	-128 … 127	1
short	-32768 ... 32767	2
int		2 або 4
long	-2,147,483,648 ... 2,147,483,647	4
unsigned char	0 ... 255	1
unsigned short	0 … 65535	2
unsigned		2 або 4
unsigned long	0 ... 4,294,967,295	4
float	$\pm(3.4\cdot10^{-38}$ ... $3.4\cdot10^{38})$	4
double	$\pm(1.7\cdot10^{-308}$ ... $1.7\cdot10^{308})$	8
long double	$\pm(3.4\cdot10^{-4932}$ ... $3.4\cdot10^{4932})$	10

У мові С++ використовується префіксний запис оголошення. При цьому на початку вказується тип змінної, а потім її ім'я. Змінні необхідно описати до моменту коли вони будуть використані у програмі. Ніяких ключових слів при цьому не пишеться. Змінну можна ініціалізувати безпосередньо у місті опису.

Наприклад:
```
int a = 12;
float c, d;
```

Цілочислові типи даних можуть використовуватися з модифікаторами **signed** (зі знаком) **usigned** (без знаку) не може набувати від'ємних значень за рахунок чого дещо розширюється діапазон можливих додатних значень типу

Наприклад: **char –** діапазон **(-128...127)**

usigned char – діапазон **(0...255)**

Перетворення типів

Компілятор Паскаля виконує автоматичне перетворення типів даних, цей стиль підтримує і С++. Якщо операнди мають різні типи, перед тим як виконати операцію молодший тип „підтягується " до старшого.

У С++ існує можливість примусового перетворення типів, щоб дозволити явно конвертувати значення одного типу до іншого. Загальний синтаксис перетворення тип має два варіанти:

1. **(новий тип) вираз;**
2. **новий тип (вираз).**

ТЕМА 5. ОСНОВНІ ОПЕРАЦІЇ МОВИ С++

Операції подібні вбудованим функціям програмування, що застосовуються до операндів. Операції, які мають два операнди називаються *бінарними*, якщо операнд один – *унарними*. Три операнди має лише одна операція мови С++ - „?” – *тернарна*.

Операнд – елемент операції. Операндами можуть бути константи, змінні, функції, вирази.

За призначенням операції можна поділити на:

1. арифметичні операції;
2. операції присвоювання;
3. операції відношення;
4. логічні операції;
5. порозрядні операції;
6. умовна операція „?”;
7. операція слідування (кома).

Арифметичні операції

До арифметичних операцій належать відомі бінарні операції додавання, віднімання, множення, ділення та знаходження залишку від ділення.

Таблиця 4. Бінарні арифметичні операції

Операція	Значення	Приклад
+	Додавання	a+b
–	Віднімання	a–b
*	Множення	a*b
/	Ділення	a/b
%	Залишок від ділення	a%6

До наведених арифметичних операцій діють наступні правила:

- бінарні операції додавання та віднімання застосовуються лише до цілих та дійсних чисел, а також до покажчиків;
- в операціях множення та ділення операнди можуть бути будь яких типів;
- операція „залишок від ділення” застосовується лише до цілих чисел;

В мові С++ визначені унарні арифметичні операції. Операція **інкременту** (++) збільшує операнд на одиницю, а операція **декременту** (- -) зменшує операнд на

одиницю. Ці операції виконуються швидше ніж операції віднімання чи додавання одиниці.

Таблиця 5. Унарні арифметичні операції

Операція	*Значення*	*Приклад*
+	Унарний плюс (підтвердження знака)	+5
–	Унарний мінус (зміна знака)	–x
++	Операція інкременту (збільшення на 1)	i++, ++i
––	Операція декременту (зменшення на 1)	j—, —j

Існує дві форми запису операцій інкременту та декременту: **префіксна** та **постфіксна**. Якщо операція інкременту розміщена перед змінною, це префіксна форма, якщо після змінної то постфіксна. У префіксній формі змінна спочатку збільшується на одиницю, а потім її нове значення використовується у виразі. При постфікс ній формі у виразі використовується поточне значення змінної, а потім відбувається збільшення цієї змінної на одиницю. Аналогічно відбувається з операцією декременту.

Операції присвоювання

У мові C++ знак „=” означає операцію присвоювання деякого значення змінній. Операція присвоювання повертає як результат присвоєне значення, тому допускається присвоєння виду a = (b = c = 1) + 1

Таблиця 6. Операції присвоювання

Операція	*Значення*
a = b	присвоювання значення *b* змінній *a*
a += b	додавання з присвоюванням. Означає a = a + b
a –= b	віднімання з присвоюванням. Означає a = a – b
a *= b	множення з присвоюванням. Означає a = a * b
a /= b	ділення з присвоюванням. Означає a = a / b
a %= b	залишок від ділення з присвоюванням. Означає a = a % b
a <<= b	зсув вліво з присвоюванням. Означає a = a << b
a >>= b	зсув вправо з присвоюванням. Означає a = a >> b
a &= b	порозрядне І з присвоюванням. Означає a = a & b
a |= b	порозрядне АБО з присвоюванням. Означає a = a | b
a ^= b	побітове додавання за МОД2 з присвоюванням, означає a = a ^ b

Операції відношення

Операції порівнянні використовуються здебільшого в умовних виразах. Кожна умовна операція перевіряється істина вона чи хибна. Результатом умовного виразу є цілочислове арифметичне значення. „Істинно” – це ненульова величина, а „хибно” – це нуль.

Таблиця 7. Операції порівняння

Операція	*Значення*
<	Менше
<=	менше або рівно
==	перевірка на рівність
>=	більше або рівно
>	Більше
!=	перевірка на нерівність

Логічні операції

Логічні операції використовуються здебільшого для „об’єднання” простих логічних виразів.

Наприклад: (a = = b&&c = = d)

Таблиця 8. Логічні операції

Операція	*Значення*
&&	логічне І (and)
||	логічне АБО (or)
!	логічне заперечення (not)

Порозрядні операції (побітові операції)

Порозрядні операції застосовуються лише до цілочислових операндів і „працюють” з їх війковими представленнями.

Таблиця 9. Порозрядні операції

Операція	*Значення*
~	порозрядне заперечення
&	побітова кон’юнкція (побітове І)
|	побітова диз’юнкція (побітове АБО)
^	побітове додавання за МОД2
<<	зсув вліво
>>	зсув вправо

- Порозрядне заперечення заміняє кожну 1 на 0, а 0 на 1.

Наприклад: ~(10011010) = = (01100101)

- Порозрядна кон'юнкція порівнює послідовно розряд за розрядом два операнди. Для кожного розряду результат рівний 1, якщо тільки два відповідні розряди операндів рівні 1, в інших випадках результат 0.

Наприклад: (10010011) & (00111101) = = (00010001)

- Порівняльна диз'юнкція порівнює послідовно розряд за розрядом два операнди. Для кожного розряду результат рівний 1, якщо хоча б один з відповідних розрядів рівні 1.

Наприклад: (10010011) | (00111101) = = (10111111)

- Побітове додаваня за МОД 2 порівнює послідовно розряд за розрядом два операнди. Для кожного розряду результат рівний 1, якщо один з двох відповідних розрядів (але не обидва одночасно) рівні 1.

Наприклад: (10010011) ^ (00111101) = = (10101110)

- Операція зсуву вліво (вправо) переміщує розряди першого операнду вліво (вправо) на число позицій, яке задано другим операндом. Позиції, що звільняються заповнюються нулями, а розряди, що зсуваються за ліву (праву) межу втрачаються.

Наприклад: (10001010) << 2 = = (00101000)

(10001010) >> 2 = = (00100010)

Операція слідування

Операція „кома" (,) називається операцією слідування, яка зв'язує два довільні вирази. Список виразів, розділених комою обчислюється зліва на право.

Наприклад:

```
a = 4;
b = a+5;        ⟶   a = 4 , b = a+5;
```

Умовна операція

Умовна операція (?) – єдина тернарна операція мови С++. Її синтаксис

Умова **?** *вираз_1* **:** *вираз_2*

Наприклад: max = (x > y)? x : y;

ТЕМА 6. СТРУКТУРА ПРОГРАМИ МОВОЮ С++

Усі програми написані мовою С++ повинні містити в собі хоча б одну функцію. Функція **main()** – вхідна точка будь якої програмної системи, причому немає різниці, де її розміщувати. Якщо вона відсутня завантажувач не зможе зібрати програму.

```
main ()          //main() – заголовок функції.
{                //Тіло функції записується у {… }. У тілі функції містяться команди.
   …….           // Команди одна від одної відокремлюють символом “;”
   return 0;     //return 0 – команда виходу з функції та повернення функції результату
                 //  (у описаному випадку значення – 0)
}
```

```
void main()      // Така функція називається функцією типу void.
{…}                 Вона не повертає в програму жодних значень.
                    Тому команду return писати не треба.
```

Функція починається з імені. Фігурні дужки {...} позначають блок або складений оператор. Оператор повернення значення **return** завершує виконання програми та повертає деяке ціле значення. Ненульове значення свідчить про помилку в програмі, нульове про її успішне завершення.

Більшість програм на С++ починаються з одного або декількох операторів **# include.** Оператор **# include** забезпечує підключення «допоміжних» файлів, які містять оператори С++.

Перш ніж приступити до компіляції програми, компілятор С++ запускає спеціальну програму, яка називається **препроцесором**.

Препроцесор – це програма, яка опрацьовує директиви. Директиви препроцесора – це команди компілятора відповідної мови програмування, які виконуються на початку компіляції програми. Директиви мови С++ починаються із символу #.

Директива #include означає, що до програми необхідно приєднати програмний код із зазначеного після неї файлу (header-файлу, модуля, бібліотеки).

Наприклад”

 # include <math.h> *// math.h є стандартною бібліотекою підключення математичних функцій (усі стандартні бібліотеки розміщені у папці INCLUDE середовища С++).*

 # include <iostream.h> *// під’єднує бібліотечний файл iostream.h. Саме в цьому файлі описана команди введення та виведення даних.*

Функції введення та виведення

Існують стандартні потоки введення інформації з клавіатури та виведення даних на екран.

У мові С++ стандартний потік введення пов’язаний з константою **cin**, а виведення **cout**. Для використання цих констант підключається заголовочний файл **iostream.h**

Для виведення інформації в стандартний потік використовується формат

cout << *вираз*

Для введення даних використовується формат запису

cin >> *змінна*

Наприклад: int age;
cout << „введіть ваш вік”;
cin >> age;

Одночасно (*через пробіл*) можна вводити декілька значень для різних змінних. Введення завершується натисканням клавіші **Enter**. Якщо введено значень більше ніж очікується програмою, частина введених даних залишається у вхідному буфері.

У випадку. Якщо необхідно ввести рядок символів, введення продовжується до першого символу пробіл чи натискання клавіші **Enter.** Якщо рядок складається з декількох слів через пробіл, то кожне наступне слово буде вводитися лише після того як у програмі зустрінеться наступний оператор введення.

Приклад програми визначення максимального з двох чисел

```
#include<iostream>      //директива включення файлу
#include<conio.h>
using namespace std;
void main( )
{
    int a,b,max;
    cout <<"\n введіть два числа \n";
    cin >> a>>b;
    max=(a>b)?a:b;
    cout << "\n max=" <<max;
    _getch();              // затримка екрану (при підключеному  файлі conio.h)
}
```

ТЕМА 7. ПРОГРАМУВАННЯ РОЗГАЛУЖЕНЬ У МОВІ С++

Мова С++ володіє набором конструкція, які дозволяють керувати порядком виконання окремо взятих гілок програми в залежності від результату перевірки певної умови. Для реалізації розгалуження може бути використано умовний оператор.

Оператор IF

Реалізує розгалуження програми в залежності від перевірки умови на істинність.

if (<*умова*>) <*оператор*>**;**

Умова містить операторі порівняння:

> більше;
< менше;
>= більше або дорівнює;
<= менше або дорівнює;
= = рівно;
!= не рівно.

У конструкціях мови С++ оператори можуть бути блочними. Блок починається з відкриваючої фігурної дужки „{” та закінчується закриваючою „}”. Весь вміст блоку розглядається компілятором як єдиний оператор.

Оператор IF-ELSE

Реалізує розгалуження програми в залежності від перевірки умови. Якщо умова набуває значення „істина”, то виконується оператор_1, якщо умова „хибна” – оператор_2.

```
if (<умова>) <оператор_1>;
else <оператор_2>;
```

або

```
if (<умова>)
{
<оператори_1>;
}
else
{
<оператори_2>;
}
```

Якщо умова складається з декількох частин то у мові С++ застосовуються логічні операції:

&& логічне І;
|| логічне АБО;
! логічне заперечення.

При використанні логічних операцій умова групується в дужки.

Приклад програми визначення максимального числа.

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
   int a, b, max;
   cout<<"\n Введіть 2 числа \n";
   cin>>a>>b;
   if(a>b) max=a;
   else max=b;
   cout<<"max= "<<max;
   _getch();
}
```

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
    int a, b, c, max;
    cout<<"\n Введіть 3 числа \n";
    cin>>a>>b>>c;
    if(a>b&&a>c) max=a;
    else
    if(b>a&&b>c) max=b;
    else max=c;
    cout<<"max= "<<max;
    _getch();
}
```

Оператор вибору SWITCH

Оператор вибору застосовується коли необхідно здійснити вибір одного варіанту із багатьох.

switch (<*вираз*>**)**
{
case <*константне значення 1*>**:** <*оператор 1*>**; break;**
case < *константне значення 2*>**:** <*оператор 2*>**; break;**
default>**:** <*оператор n*>**; break;**
}

Використання оператора починається з обчислення виразу, після чого управління передається одному з операторів, за умови, що значення константи співпадає зі значенням виразу.

Гілка **DEFAULT** спрацьовує, якщо значення виразу не відповідає жодній зі значень констант.

Обов'язковим є застосування оператора **BREAK** у кожному **CASE** – фрагменті.

ТЕМА 8. ЦИКЛИ В МОВІ С++

Оператор циклу з передумовою

Оператор **WHILE** використовується для організації циклічного обчислювального процесу.

Синтаксис:

while (<*логічний вираз*>) <*оператор*>;

Цикл закінчується у випадках:

1. умовний вираз у заголовку приймає нульове значення (хибне);
2. у тілі циклу досягнуто місця, де розташований оператор **BREAK**;
3. у тілі циклу виконаний оператор **RETURN**;

У перших двох випадках керування передається оператору, розташованому безпосередньо за циклом, у третьому випадку активна на той момент функція завершує свою роботу, повертаючи якесь значення.

Якщо у тілі циклу необхідно виконати більше ніж один оператор то їх заключають у фігурні дужки.

Приклад програми для виведення таблиці множення на довільне число.

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
   int  i=0, r, x;
   cout<<"x=";
   cin>>x;
   while (++i<=10)
   {
     r=x*i;
     cout<<"\n" <<x<<"*"<<i<<"="<<r;
   }
   _getch();
}
```

Приклад програми, яка обчислює кількість від'ємних чисел у послідовності, що вводиться з клавіатури.

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
   int  i=0, n, k=0, x;
   cout<<"n=";
   cin>>n;
   while (++i<=n)
   {
      cout<<"\n x=";
      cin>>x;
      if(x<0) k++;
   }
   cout<<"\n k=" <<k;
   _getch();
}
```

Оператор циклу з постумовою DO...WHILE

Оператор циклу з постумовою використовують для організації циклічного виконання оператору, який називається тілом циклу, до тих пір, поки умова не стане хибною.

Синтаксис:

do <*оператор*> **while (**<*логічний вираз*>**);**

Ситуації що призводять до виходу з циклу, аналогічні наведеним для циклу з передумовою. Характерним є те, що тіло циклу виконується хоча б один раз. На відміну від мови Паскаль, цикл з післяумовою в С++ виконується поки умова істинна і переходить до виконання наступного за циклом оператору, коли умовний вираз набуває нульового значення (хибно).

Приклад програми, яка обчислює суму послідовності парних чисел, що вводяться з клавіатури. Для завершення вводу послідовності ввести нуль.

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
   int s=0, x;
   do
   {
      cout<<"\n x=";
      cin>>x;
      if (x%2= = 0) s+=x;
   }
   while (x!=0);
   cout<<"\n s=" <<s;
   _getch();
}
```

Приклад програми, яка виводить таблицю значень функції y = 3x+2 в діапазоні (-2;2) з кроком 0,5.

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
   float  y, x= -2;
   do
   {
      y=3*x+2;
      cout<<"\n y (" <<x<< ") =" <<y;
      x+=0.5;
   }
   while (x<=2);
   _getch();
}
```

Оператор циклу FOR

Оператор **for** забезпечує циклічне повторення деякого оператора певне число раз. Повторення циклу здійснюється з використанням деякої змінної (лічильника),

яка змінюється при кожному виконанні тіла циклу. Повторення завершується, коли лічильник досягає заданого значення.

Синтаксис:

for ([*ініціалізація*]; [*перевірка_умови*]; [*нове _значення*]) *оператор*;

Перший вираз служить для ініціалізації лічильника, другий для перевірки кінця циклу, а третій – для зміни значення лічильника.

Приклад програми, яка обчислює суму перших N елементів ряду 1+1/2+1/3+.... (кількість вводиться під час роботи програми).

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
   double s = 0;
   int n;
   cout<<"\n n=";
   cin>>n;
   for (int i=1; i<=n; i++)
   s+=1.0/i;
   cout<<"\n s="<<s;
   _getch();
}
```

Приклад програми, яка виводить таблицю квадратів перших п'яти цілих додатних непарних чисел.

```
#include<conio.h>
#include<math.h>
#include<iostream>
using namespace std;
void main()
{
   double y, i;
   int k=0;
   for (i=1;++k<=5;i+=2)
   {
      y=pow(i,2);
      cout<<"\n "<<i<<"^2="<<y;
   }
   _getch();
}
```

Оператор розриву

Оператор розриву **BREAK** перериває виконання операторів: do, for, while, switch.

У операторі **switch** він використовується для завершення блоку **case**. У операторах циклу – для негайного завершення циклу, що не пов'язано з перевіркою звичайної умови завершення циклу. Коли оператор **BREAK** зустрічається всередині оператору циклу, то здійснюється негайний вихід з циклу та перехід до виконання оператору, що слідує за оператором циклу.

Оператор продовження

Оператор **CONTINUE** передає управління на наступну ітерацію в операторах циклу. Він може розміщуватися лише в тілі цих операторів. В операціях **do, while**

наступна ітерація починається з обчислення виразу умови, а для оператора **for** наступна ітерація починається з обчислення виразу зміни значення лічильника.

Приклад програми вгадування числа за 10 спроб.

```
#include<conio.h>
#include<iostream>
using namespace std;
void main()
{
   int i=0, r;
   while (++i<=10)
   {
      cout<<" \n вгадай число >";
      cin>>r;
      if(r ==15) break;
      cout<< " FALSE";
   }
   if(i!=11) cout <<" TRUE";
   _getch();
}
```

Приклад програми піднесення до квадрату всіх від'ємних чисел із введених з клавіатури (к-ть 5 чисел).

```
#include<conio.h>
#include<math.h>
#include<iostream>
using namespace std;
void main()
{
   double s,x;
   for (int i=5;i>=1;i--)
   {
      cout<<"\n x=";
      cin>>x;
      if(x >= 0) continue;
      s=pow(x,2);
      cout<<x<<"^2="<<s;
   }
   _getch();
}
```

ТЕМА 9. ФУНКЦІЇ

Розв'язання будь якої задачі передбачає розбиття її на дрібні задачі, які розв'язуються за допомогою стандартних процедур. Таким чином здійснюється функціональна декомпозиція. Ключовим елементом даної моделі для розв'язання задачі на мові С++ виступає функція.

Функції використовуються для спрощення процесу розробки програми у випадку, коли аналогічні перетворення над різними даними необхідно виконати декілька раз у різних частинах програми.

Кожна програма у своєму складі повинна містити **ГОЛОВНУ ФУНКЦІЮ - main()**, яка забезпечує створення точки входу в об'єктний модуль. Окрім цієї функції в програму може входити довільне число функцій, виконання яких ініціалізується або прями чи опосередковано викликається із функції **main**(). Кожна функція по відношенню до іншої є зовнішньою. Для виклику, функція повинна бути доступною, та до моменту її виклику про неї відомо було компілятору.

Опис функції.

Складається з двох частин: заголовку та тіла.

```
[тип_результату] <ім'я> ([список _параметрів])
{
<тіло функції>
}
```

Тип_результату – тип значення, яке повертає функція. У випадку відсутності специфікатора функція повертає ціле значення. Якщо функція не повертає ніякого значення та на місці типу записується **void**.

Тіло_функції - послідовність оголошень та операторів, які описують певний алгоритм. Важливим оператором тіла функції є оператор **return,** який має подвійне призначення. Він забезпечує повернення у функцію, що викликається та може бути використаний для передачі обчисленого значення функції.

Прототип функції – може вказуватися до виклику функції замість її опису для того щоб компілятор міг виконати перевірку відповідності типів аргументів та параметрів. Прототип має такий же вигляд як і опис функції, але к кінці ставиться крапка з комою „ ; ”. Мова С++ не передбачає автоматичного перетворення типів у випадку, коли аргументи не співпадають по типам з відповідними їм параметрами.

Параметри функції в прототипі можуть мати імена, але компілятору вони не потрібні.

Виклик функції може бути оформлений у вигляді оператору, якщо у функції відсутнє значення, яке вона повертає,

ім'я_функції (список_аргументів);

або у вигляді виразу, якщо існує значення яке повертає функція

Приклад програми, яка містить звертання до функції визначення максимального з 2-х чисел

```
#include<iostream>
#include<conio.h>
using namespace std;
int max(int,int);                    //прототип функції
void main()
{
    int x, y, z;
    cout<<" \n vvedite x y   >";
    cin>>x>>y;
    z=max(x,y);                      // виклик функції
    cout<< "z=" << z;
    _getch();
}
```

```
# include "d:\max.cpp"  // включення файлу з функцією
```

Опис функції **max** знаходиться у файлі **max.cpp** та має наступний вигляд:

```
int max ( int a, int b)
{
   int c;
   if(a>=b) c=a;
   else c=b;
   return c;
}
```

Прототип функції **max()** необхідний компілятору для перевірки відповідності типів аргументів і параметрів при виклику функції. При виклику функції значення аргументів **x, y** підставляється замість параметрів **a, b** відповідно. Після виконання тіла функції значення яке вона повертає передається у місце виклику функції та присвоюється змінній **z** . Опис функції знаходиться у певному файлі, тому для

включення файлу з програмою необхідно в тексті основної програми вказати директиву препроцесора **# include “<шлях до файлу>”;**

Опис функції може знаходиться в одному файлі з головною програмою. При цьому директива препроцесора не вказується, а замість неї поміщається опис функції.

Існує ряд задач, які не вимагають передачу значення яке повертає функція.

Приклад: Вивести на екран символ введений з клавіатури n-раз

```
#include<iostream>
#include<conio.h>
using namespace std;

int ris(int n, char c)
{
   for(int i=1; i<=n; i++)
   cout<<c;
   return 0;
}

void main()
{
   int a;
   char b;
   cout<<"\n введіть число > ";
   cin>>a;
   cout<<"\n введіть символ >";
   cin>>b;
   ris(a, b);
   _getch();
}
```

Приклад: Визначити чи введене з клавіатури число додатне.

```
#include<iostream>
#include<conio.h>
using namespace std;

void dod(int x)
{
   if(x>=0) cout<<"\n додатне";
   else cout<<"\n  від’ємне";
   _getch();
}

void main()
{
   int a;
   cout<<"\n введіть число > ";
   cin>>a;
   dod(a);
}
```

ТЕМА 10. МАСИВИ В МОВІ С++

Одновимірні масиви

Мова С++ дозволяє обробляти набори однотипних даних – масиви. У якості елементів масиву можуть виступати дані будь якого типу. А також вказівними на однотипні дані.

Оскільки всі елементи масиву відносяться до одного типу, вони також займають один розмір.

Використання масиву починається з його оголошення, яке виділяє під масив певний об'єм пам'яті

<*тип*> <*імя*>[n];

Наприклад: **int A[20];** *// у пам'яті зарезервовано місце для розташування 20 цілочислових елементів.*

Елементи масиву у пам'яті розташовується безпосередньо один за одним.

Ініціалізація масивів може відбуватися при оголошенні масиву, значення елементів записується у фігурних дужках та перераховуються через кому в порядку слідування у масиві.

int A [5]={2,4,6,8,10};

Якщо у списку ініціалізації значень вказано менше чим оголошено в розмірі масиву, має місце часткова ініціалізація. В цьому випадку після останнього значення ставиться кома.

int A [5]={2,4,6, };

При оголошенні одновимірного масиву з одночасною його ініціалізацією дозволяється опускати значення розміру у квадратних дужках. Компілятор сам підраховує кількість елементів у списку ініціалізації та виділяє необхідну область пам'яті.

int A [ ]={2,4,6,8,10};

Якщо у програмі необхідно визначити кількість елементів масиву, використовується функція **SIZEOF**.

Int K = sizeof(A) / sizeof(A[0]); *// кількість = загальний розмір масиву у байтах / розмір одного елементу*

Звертаючись до елементу масиву може здійснюватися за номером елементу в масиві. У мові С++ елементи масиву нумеруються починаючи з нуля.

Наприклад: **X=A [14];** *//змінній X присвоєно значення 15-го елементу.*

Приклад програми визначення суми елементів масиву

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
    int a[5], s=0, i;
    for (i=0; i<5; i++)
    {
       cout<<"\n a[" <<i<<"]=";
       cin>>a[i];
       s+=a[i];
    }
    cout<<"\n s="<<s<<"\n";
    for (i=0; i<5; i++)
    cout<< a[i] <<"  ";
    _getch();
}
```

Багатовимірні масиви

Багатовимірний масив розмірності **N** можна представити як одновимірний масив із масивів розмірності (**N-1**). Таким чином тривимірний масив – це масив, кожний елемент якого представляє двовимірну матрицю. При цьому розмірність масиву визначається числом парних квадратних дужок при його оголошенні.

Наприклад: char M [6] [9]: // *двовимірний масив 6*x*9*

Вираз **Array [idx] [idy]** представляючий двовимірний масив переводиться компілятором у еквівалентний вираз ***(*(Array +idx)+idy).**

int M [3] [3] = {1,2,3,4,5,6,7,8,9}

На початку роботи у пам'яті резервується місце для дев'яти елементів із заповненням масиву.

Багатовимірні масиви ініціалізуються в порядку найскорішої зміни самого правого індексу.

x_{00}=1; x_{01}=2; x_{02}=3;

x_{10}=4; x_{11}=5; x_{12}=6;

x_{20}=7; x_{21}=8; x_{22}=9;

Для багатовимірних масивів при ініціалізації дозволяється опускати величину першої розмірності **int M [ ] [3] = {1,2,3,4,5,6,7,8,9}**

Введення елементів двовимірного масиву може здійснюватися з клавіатури, або генеруватися випадковим чином.

```
// введення елементів масиву з клавіатури
for ( i=0; i<2; i++)
for ( j=0; j<3; j++)
{
   cout<<"\ n введіть елемент масиву";
   cin>>a[i][j];
}
```

```
// автоматичне заповнення масиву випадковими числами в діапазоні від 1 до 10
#include <time.h>
  …….
srand(time(NULL));
for ( i=0; i<2; i++)
for ( j=0; j<3; j++)
a[i][j] = rand()%10;
```

Якщо елементи масиву генеруються випадковим чином, необхідно підключити заготовочний файл **time.h**

Для встановлення діапазону можна задати такі параметри:

```
a[i][j] = rand()%10;  // числа генеруються в діапазоні 1..10
a[i][j] = rand()%100;  // числа генеруються в діапазоні 1..100
a[i][j] = rand()%(0+10);  // числа генеруються в діапазоні 0..10
a[i][j] = rand()%20-10;  // числа генеруються в діапазоні -10..10
```

Виведення елементів масиву у вигляді таблиці здійснюється за допомогою конструкції:

```
for (int i=0; i<3; i++)
{
   for (int j=0; j<3; j++)
   cout << a[i][j]<< "   ";
   cout <<"\n";
}
```

Приклад програми визначення максимального елементу двовимірного масиву

```
#include <time.h>
#include <conio.h>
#include <iostream>
using namespace std;
void main( )
{
  srand(time(NULL));
  int a[4][3],i, j;
  //заповненя масиву довільним чином
  for ( i=0; i<4; i++)
  for ( j=0; j<3; j++)
  a[i][j] = rand()%10;
  //визначення максимального елементу
  int max=a[0][0];
  for (i=0;i<4;i++)
  for (j=0;j<3;j++)
  if (a[i][j]>max) max=a[i][j];
  //виведення масиву на екран
  for (int i=0; i<4; i++)
  {
     for (int j=0; j<3; j++)
     cout<< a[i][j]<< "  ";
     cout<<"\n";
  }
  // виведення результату
  cout<< "\n max= "<<max;
  _getch();
}
```

Приклад програми визначення суми парних елементів масиву.

```
#include <time.h>
#include <conio.h>
#include <iostream>
using namespace std;
void main( )
{
    srand(time(NULL));
    int a[8][8], s=0, i, j;
    for ( i=0; i<8; i++)
    for ( j=0; j<8; j++)
    {
    a[i][j] = rand()%10;
    if (a[i][j] %2==0) s+=a[i][j];
    }
    for (int i=0; i<8; i++)
    {
        for (int j=0; j<8; j++)
        cout<< a[i][j]<< "  ";
        cout<<"\n";
    }
    cout<< " \n s=  "<<s;
    _getch();
}
```

Покажчики

В результаті процесу компіляції програми всі імена змінних будуть перетворені в адреси комірок пам'яті, в яких містяться відповідні значення даних.

Покажчик - це змінна або константа стандартного типу даних для збереження адреси змінної визначеного типу.

Значення покажчика - це беззнакове ціле, воно повідомляє, де розміщена змінна, і нічого не говорить про саму змінну.

Змінна типу покажчик оголошується подібно звичайним змінним із застосуванням унарного символу "*". Форма оголошення змінної типу покажчик наступна:

<*тип*> * <ідентифікатор> ;

Наприклад: float *pf; // - *покажчик - змінна на дані типу float*

Основні операції над покажчиками

Мова Сі++ надає можливість використання адрес змінних програми за допомогою основних операцій - **&** (амперсенд) та *****:

Призначення цих операцій:

& ім'я змінної - одержання адреси; визначає адресу розміщення значення змінної визначеного типу;

*** ім'я-покажчика** - отримання значення визначеного типу за вказаною адресою; визначає вміст змінної, розміщеної за адресою, що міститься у даному покажчику; це - непряма адресація (інші назви - "зняття значення за покажчиком" або "розіменування").

Оператор присвоювання значення адреси покажчику має вигляд:

<*ім'я_змінної_покажчика*> = & <*ім'я змінної*>;

Наприклад:

int i, *pi; *// pi -змінна покажчик*

pi = &i; *// pi одержує значення адреси 'i'*

Оператор присвоювання, що використовує ім'я покажчика та операцію непрямої адресації, можна представити у вигляді:

<*ім'я змінної*> = * <*ім'я-покажчика*>;

де ім'я-покажчика - це змінна або константа, що містить адресу розміщення значення, необхідного для змінної лівої частини оператора присвоювання.

*Наприклад:*i = *pi; *// 'i' одержує значення, розташоване за адресою, що міститься в покажчику* **'pi'**

Звертаючись до елементу масиву може здійснюватися одним з двох способів:

1. за номером елементу в масиві;
2. за покажчиком.

Доступ до елементів масиву через покажчики полягає у тому, що ім'я масиву асоціюється компілятором з адресою першого елементу (з індексом 0). Таким чином можна присвоїти покажчику адресу нульового елементу, використовуючи ім'я масиву.

int a[ ]={10, 5, -4, 7, 23};
int *p = a;

Розіменовуючи покажчик **p**, отримуємо доступ до вмісту **a [0].**

int b =*p; *//присвоєння змінній* **b** *значення, яке знаходиться за адресою на яку посилається покажчик.*

У С++ покажчики та масиви тісно пов'язані. Збільшуючи або зменшуючи значення покажчика на масив отримуємо доступ до елементів масиву шляхом модифікації покажчика.

p+ = 3; *// вказує на елемент з індексом [3]=***7**
p++; *// вказує на елемент з індексом [4]* **=23**
int b =*p; *// присвоєння змінній* **b** *значення, яке знаходиться за адресою на яку посилається покажчик, тобто* **' 23'**

Даний запис можна замінити еквівалентним **int b =*(a+4);**

Приклад програми визначення суми елементів масиву. Звертаючись до елементу масиву:

1. за номером елементу в масиві

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
    int a[5], s=0, i;
    for (i=0;i<5;i++)
    {
       cout<<"\n a[" <<i<<"]=";
       cin>>a[i];
       s+=a[i];
    }
    cout<<"\n s="<<s;
    _getch();
}
```

2. за покажчиком

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
    int a[5], s=0;
    int *p=a;
    for (int i=0;i<5;i++)
    {
      cout<<"\n a[" <<i<<"]=";
      cin>>*p;
      s+=*(p++);
    }
    cout<<" \n s= "<< s;
    _getch();
}
```

Доступ до елементів багатовимірного масиву через покажчики здійснюється таким чином, що для доступу до вмісту комірки покажчик необхідно розіменувати двічі ***p = *m;**.

```
{
 …
 int m [3] [2] = {3, 5, -8, 14, -5, 11};
 int *p = *m;
 p+=3;
 int d=*p;
 ….
}
```

У наведеному фрагменті коду оголошується масив розмірністю 3x2 і покажчик на нього *p (фактично – покажчик на **m** [0] [0] = 3).

У другому рядку ідентифікатор **m** вже є покажчиком на елемент з індексом 0, але оскільки масив двовимірний, необхідне повторне розіменування.

Збільшення значення покажчика **p** на 3 приводить до того, що він вказує на елемент масиву **m** [1] [1] = 14. Тобто змінній **d** буде присвоєно значення **14**.

Динамічне виділення масивів

У програмі кожна змінна може розташовуватися:

1. у області даних програми;
2. у стеку;
3. у вільній пам’яті (так званій купі)

Кожній змінній в програмі може відводитися пам’ять:

- статично – у момент завантаження програми (явно a[5], чи неявно a[ ]={2,3})
- динамічно – у процесі виконання програми. (Якщо недостатньо місця для розташування всіх елементів великого масиву, або їх декілька)

Для розміщення у пам’яті динамічних об’єктів, для них необхідно попередньо виділити у пам’яті відповідне місце. По завершенню роботи з об’єктом виділену для його збереження пам’ять необхідно вивільнити.

Виділення динамічної пам’яті під об’єкт здійснюється за допомогою виразів: **MALLOC , CALLOC, NEW**

Вивільнення виділених ресурсів пам’яті здійснюється виразами: **FREE, DELETE**

Приклад програми визначення суму елементів масиву

```
#include <iostream>
#include <conio.h>
#include <ctime>
using namespace std;
void main()
{
  srand(time(NULL));
  int s=0;
  int *a= new int[10];
  for (int i=0; i<10;i++)
  {
       a[i]=rand()%10;
       cout<<a[i]<<" ";
       s+=a[i];
  }
  cout<<" \n s=" <<s;
  delete[]a;
 _getch();
}
```

ТЕМА 11. РЯДКИ ТА ОПЕРАЦІЇ З НИМИ

Масиви символів

Символьних рядків в С++ представляються як масиви елементів типу CHAR, які закінчуються ***термінатором рядку*** (символом з нульовим значенням **\0**). Символьні рядки складаються із набору символьних констант, заключних у подвійні лапки.

При оголошенні рядкового масиву необхідно враховувати наявність термінатора у кінці рядку, відводячи тим самим під рядок на один байт більше.

Рядковий масив може ініціалізуватися при оголошенні, при цьому компілятор автоматично обчислює розмір майбутнього рядка та додає нуль-термінатор.

У якості оператора введення при роботі з рядками замість оператора запису в поток (**>>**) краще використовувати функцію **GETLINE ()** так як потоковий оператор введення ігнорує пробіли та може продовжувати введення елементів за межами масиву.

Функція **GETLINE (a, n)** де
a - вказує на рядок, в який відбувається введення
n - кількість символів, які буде введено.

Приклад:

```
void main()
{
char a[5];                          // оголошення рядкового масиву
cout<<"Як твоє ім'я \n";
cin.getline(a,5);                   // введення рядкового масиву
cout<<" Вітаю "<< a;
_getch();
}
```

Інколи у якості другого параметра функції **GETLINE()** використовується функція **SIZEOF**(). Це робить код універсальним та не вимагає явного визначення числа символів, які необхідно ввести.

Приклад: **cin.getline(a, sizeof(a));**

Стандартний клас String

Даний клас дозволяє визначити у програмі рядковий тип даних. Щоб отримати доступ до методів класу необхідно підключити заготовочний файл **string.** Об'єкти класу **string** не закінчується нульовим байтом, як це прийнято при представлені рядків за допомогою масиву символів.

Для введення змінних цього типу можна використовувати **cin**, або спеціальну функцію **getline.**

getline(cin, s);

де **s** — ім'я змінної типу **string**.

Оголошення та ініціалізація рядків.

string s1; *// порожній рядок*
string s2 ("Hi");
string s3="word";

У класі **string** визначено оператори для роботи з рядками, які дозволяють використовувати об'єкти класу у виразах програми.

Оператор	Приклад	Результат
=	s1=s3	s1="word"
+	s1=s2+s3; s1=s2+" , " +s3	s1="Hiword" s1="Hi, word"
+ =	s1+="!!!";	s1="Hi, word!!!"

Основні методи обробки рядків

Знищення символів

Метод **erase(n, k) –** знищує **k** символів рядку починаючи з позиції **n.** (*Індексування символів починається з нуля*)

Наприклад:

```
string s ="My program";
s.erase (3, 4);
cout<< s;     // буде виведено повідомлення «My ram»;
cout<<s.erase();   // буде виведено порожній рядок;
```

Вставка символів

Метод **insert(n, r) –** вставляє рядок **r** у позицію **n** визначеного рядка.

Наприклад:

string s =”My ram”, s1=”prog”;
s.insert (3, s1);
cout<< s; // *буде виведено повідомлення «My program»;*

Заміна символів

Метод **replace(n,k,r)** - Заміняє **k** символів рядку, починаючи з позиції **n** заданим рядком **r.**

Наприклад:

string s1 = “I like ice cream”,s2 = “apples”
s1.replace (7,9,s2);
cout<<s1; *// буде виведено повідомлення «I like apples»;*

Визначення довжини рядка

Для визначення довжини рядка використовують два методи **size()** та **length(),** які повертають кількість символів у рядку типу string

Наприклад:

string r = “Hi, world”;
cout<< r.size() << endl; *// на екран буде виведено 9*
cout<< r.length( ) << endl; *// на екран буде виведено 9*

Приклад визначення кількості літер «а» у рядку:

1) з використанням стандартного класу String

```
#include<iostream>
#include<conio.h>
#include<string>
using namespace std;
void main()
{
    string a;
    int k=0;
    cout<<">";
    getline(cin, a);
    int x=a.length();
    for(int i=0;i<x;i++)
    {if(a[i]=='a') k++;}
    cout<<" > "<< k;
    _getch();
}
```

2) з використанням масиву символів

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
   char a[50];
   int k=0;
   cout<<"\n > ";
   cin.getline(a,sizeof(a));
   for(int i=0;i<sizeof(a);i++)
   {if(a[i]=='a') k++;}
   cout<<" > "<< k;
   _getch();
}
```

ТЕМА 12. СТРУКТУРИ

Структури дозволяють об'єднувати в єдиному об'єкті сукупність значень, які можуть мати різні типи. Оголошення структури здійснюється за допомогою ключового слова **STRUCT**.

Використання структурованих даних в тілі програми можливе у випадку якщо буде оголошено об'єкт новоствореного типу.

```
STRUCT [імя_структури]
{
тип_1  елемент_1;
тип_2  елемент_2;
.....
тип_n  елемент_ n;
};
```

```
STRUCT [імя_структури]
{
тип_1  елемент_1;
тип_2  елемент_2;
.....
тип_n  елемент_ n;
} [список_описів];
```

Наприклад:

```
STRUCT date
{
int day;
int month;
int year;
char name [30];
} a[100], *pd;    // a[100] – масив елементів типу date, *pd – покажчик на тип date
```

На відміну від описів інших типів даних, опис структури не виділяє місяця у пам'яті під елементи структури. Її опис визначає лише шаблон, що описує характеристики змінних, що будуть розміщенні у конкретній структурі. Щоб ввести змінні та зарезервувати для них пам'ять необхідно або після фігурної дужки, що завершує опис структури, вказати список ідентифікаторів, або окремо оголосити змінну типу.

Доступ до окремого елемента структури забезпечується операторами вибору: (прямий селектор „ . „ *крапка*) та (непрямий селектор „ -> ”) - використовується для доступу до елементів структури через показники.

Наприклад:

```
a[10].day = 31;
*pd - > month = 12;          //використовується для доступу до елементів структури через покажчики.
```

Ініціалізація структури подібна до тієї, що і в масивах, але з урахуванням розміщення даних різного типу.

Наприклад: date b = {10, 7, 2001, “Petrov”};

Для змінних одного структурного типу визначена операція присвоювання, при цьому здійснюється поелементне копіювання значень полів. Але для порівняння структур необхідно перевірити рівність відповідних полів цих структур.

Зв’язок структур та покажчиків дозволяє розв’язати ряд складних задач, але вимагає високої кваліфікації програміста.

Приклад програми «книжкова крамниця» .

```
#include <iostream>
#include <conio.h>
using namespace std;
struct book
{
   char titel[10];
   char autor[10];
   float price;
} B;
void main()
{
   cout<<"\n vvedit avtora ";
   cin.getline(B.autor,sizeof(B.autor));
   cout<<"\n vvedit nazvu ";
   cin.getline(B.titel,sizeof(B.titel));
   cout<<"\n vvedit price ";
   cin>>B.price;
   cout<<B.autor<<"   "<<B.titel<<"   "<<B.price;
   _getch();
                                          }
```

ТЕМА 13. ГОЛОВНІ ПРИНЦИПИ ОБ'ЄКТНО - ОРІЄНТОВАНОГО СТИЛЮ ПРОГРАМУВАННЯ

Головні елементи об'єктно-орієнтованого програмування

Для об'єктно-орієнтованого стилю концептуальна основа полягає в об'єктному підході. Цьому підходу відповідають чотири головних елементи: **абстрагування, обмеження доступу, модульність** та **ієрархія.**

ВИЗНАЧЕННЯ

1. **Абстрагування** - виділення таких вагомих характеристик об'єктів, які відрізняють його від усіх інших об'єктів і які чітко визначають особливості даного об'єкта з точки зору подальшого аналізу.

2. **Обмеження доступу** - процес захисту окремих елементів, який не впливає на вагомі характеристики об'єкта, як цілого.

3. **Модульність** - властивість системи, яка зв'язана з можливістю декомпозиції на ряд тісно зв'язаних модулів (частин).

4. **Ієрархія** - впорядкування за деякими правилами об'єктів системи.

Ідея класів - це основа об'єктно-орієнтованого програмування (ООП). Мета ООП - намагання зв'язати дані й функції для їх обробки в єдине ціле - клас. В класах об'єднуються структури даних і функції їх обробки. Ідея класів відображає будову об'єктів реального світу - оскільки кожний предмет або процес має свої властивості, будову, поведінку.

Клас - це визначений користувачем тип даних. В класі задаються властивості і поведінка якого-небудь, об'єкта у вигляді полів-даних і функцій для роботи з ними.

Парадигма програмування - це набір теорій, методів, стандартів, які використовуються при розробці та реалізації програм на комп'ютері. ООП часто називають новою парадигмою програмування, хоча її революційний поступ розпочався давно у минулому.

ООП основане на трьох принципах, що надають класам нові властивості:

1. **Інкапсуляція** - об'єднання в єдине ціле даних і алгоритмів обробки цих даних. В ООП дані називаються полями, а алгоритми - методами або функціями-членами (methods, member functions).

2. **Успадкування** - властивість створення ієрархії класів, коли нащадки отримують від попередника поля і методи.

3. **Поліморфізм** (від гр. poly - багато і morphos форма, означає багато форм) - це властивість класів однієї ієрархії вирішувати схожі за змістом завдання за допомогою різних алгоритмів.

Технологія візуального програмування

Перехід до технології візуального програмування вимагав створення нової бібліотеки класів. Головною особливістю даної бібліотеки полягає в тому, що в її основі лежить концепція **властивостей, методів та подій.**

Властивості об'єкту визначають його характеристики та поведінку.

Ім'я об'єкту -> властивість = значення властивості;

Наприклад: **label1->Text =”*текстове повідомлення*”**

Другим елементом для управління об'єктами є **методи.** На відміну від властивостей, які представляють собою змінні чи об'єкти, методи представляють собою функції класу (об'єкту), які можна викликати з зовні для управління об'єктом.

Ім'я об'єкту -> метод();

Наприклад: **Form1->Close()**

Повний перелік властивостей і методів об'єкту приводиться в документації та у в будованій довідковій системі.

Третій елемент управління об'єктом – це **подія**. Сама операційна система Windows – це система яка управляється подіями. Для кожного об'єкту існує свій перелік подій, для яких можна змінити стандартну реакцію програми. Змінити реакцію на обробку події можна використовуючи вкладку подій у вікні властивостей.

ТЕМА 14. ВІЗУАЛЬНЕ СЕРЕДОВИЩЕ РОЗРОБКИ ПРОГРАМ МОВОЮ С++

Середовище Microsoft Visual C++ Express

Після завантаження на екрані з'являється інтегроване середовище розробки.

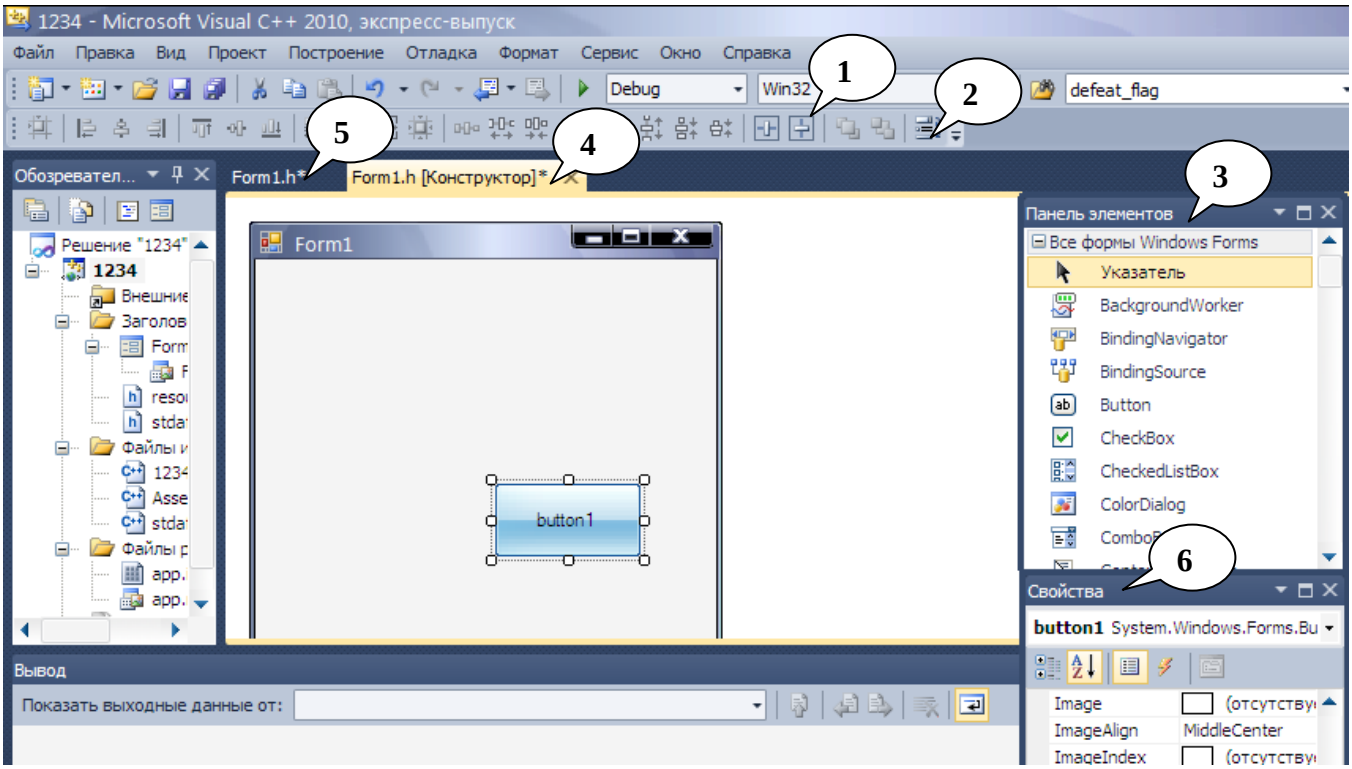


Рис.14.1. Інтерфейс середовища

До основних елементів середовища програмування відносяться:

Команда головного меню (1) – містить повний перелік команд, що керують процесом розробки програми.

Панель інструментів (2) – кнопки для виконання найпоширеніших команд.

Панель елементів (3) – містить основні візуальні та невізуальні компоненти, які програміст може включати у додаток на етапі розробки. Ці операції не вимагають програмування, необхідний код програми генерується автоматичною

Редактор форм (4) – (форма, це вікно яке використовується у додатку та містить свої власні компоненти) Редактор форм дозволяє створювати вікно необхідного вигляду розміщуючи на ньому необхідні компоненти.

Редактор коду (5) – вікно призначене для внесення змін у вихідний текст автоматично згенерованої програми.

Панель «Свойства» (6) - відображає властивості вибраного (*виділеного*) в даний момент об'єкту.

У верхній частині вікна знаходиться рядок меню і область відображення панелей інструментів. За умовчанням у області відображення панелей інструментів виводиться панель **«Стандартная»**. Щоб зробити доступними інші панелі інструментів, треба вибрати команду «**Вид > Панель инстументов».**

Центральну частину вікна Visual C++ займає вікно редактора форми (4). У ньому знаходиться *форма* - заготовка вікна додатку (*вікно програми під час його розробки прийняте називати формою*). У вікні дизайнера форми у графічній формі відображаються інструкції програми (*у заголовку вікна вказане ім'я h-файлу*), що забезпечують створення вікна. Щоб побачити формму, виберіть в меню «**Вид**» команду **«Конструктор»**. Щоб переглянути код програми «**Вид >Код**».

У правій частині головного вікна Visual C++ зазвичай відображаються вікна «**Панель элементов**» і **«Свойства»** (*рис.14.2*).

У вікні «**Панель элементов**» знаходяться *компоненти*, які можна помістити (перетягнути) на форму. **Компонент** - це об'єкт, що реалізовує деяку функціональність.

У вікні **«Свойства»** відображаються властивості вибраного (виділеного) в даний момент об'єкту - компоненту або, якщо жоден з елементів не виділений, самої форми. Ім'я і тип об'єкту відображаються у верхній частині вікна **«Свойств»**.

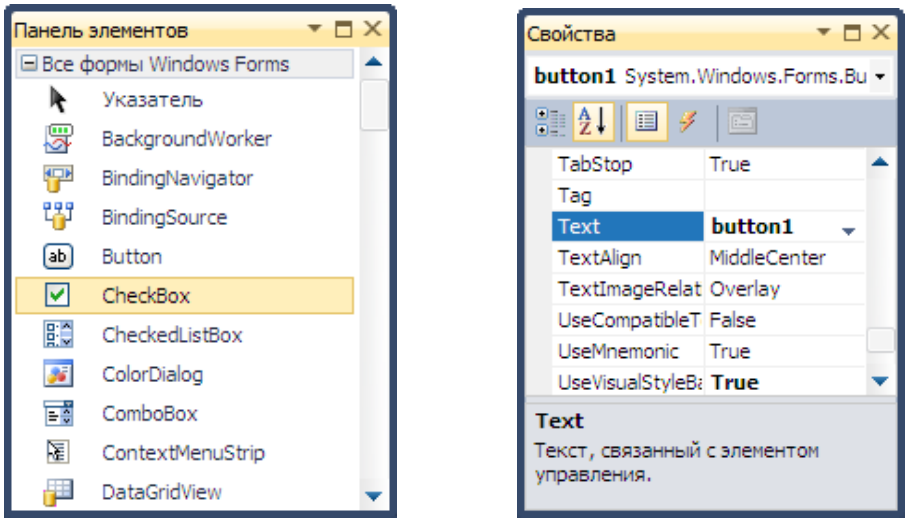


*Рис.14.2. Вікна «**Панель элементов**» та «**Свойства**»*

Вікно **«Свойства»** використовується для редагування значень властивостей об'єктів і, як наслідок, зміни їх вигляду і поведінки. Наприклад, щоб змінити надпис на кнопці (**button**), треба змінити значення властивості **Text**.

При відображенні властивості можуть бути об'єднані в групи за функціональною ознакою «**По категориям**» або впорядковані за абеткою «**В алфавитном порядке**» (*рис.14.3)*.

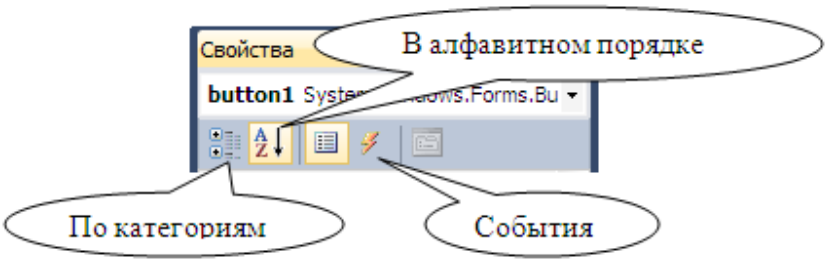


*Рис.14.3 Панель інструментів вікна **«Свойства»***

Натиснувши у вікні **«Свойства»** кнопку **«События»**, можна побачити події, які здатний сприймати вибраний об'єкт (*рис.14.4)*. **Подія** - це те, що відбувається під час роботи програми. Наприклад, командна кнопка може реагувати на клацання кнопкою миші - подія «**Click»**.

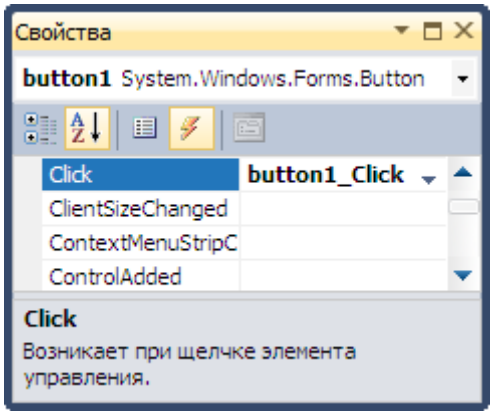


Рис.14.4. *Вікно **«Свойства»**вкладка **«События»***

У вікні **«Обозреватель решений»** відображається структура проекту. У ньому перераховані файли, створюючи проект. Функція «**main»** знаходиться в головному файлі проекту (*.cpp), опис класу форми - в h-файлі (Form1.h). Тут же відображаються імена файлів ресурсів (*.rc) і значка (*.ico) проекту та ін. (*рис.14.5)*.

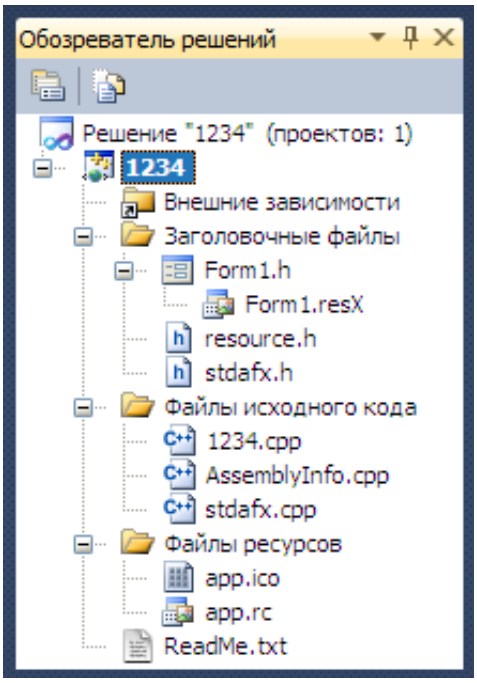


Рис.14.5. *Вікно* ***«Обозреватель решений»***

Вікно **«Обозреватель решений»** зручно використовувати для швидкого доступу до необхідного елементу проекту.

Етапи розробки додатків

На початку створення проекту необхідно виконати команду «**Файл > Создать > Проект».**

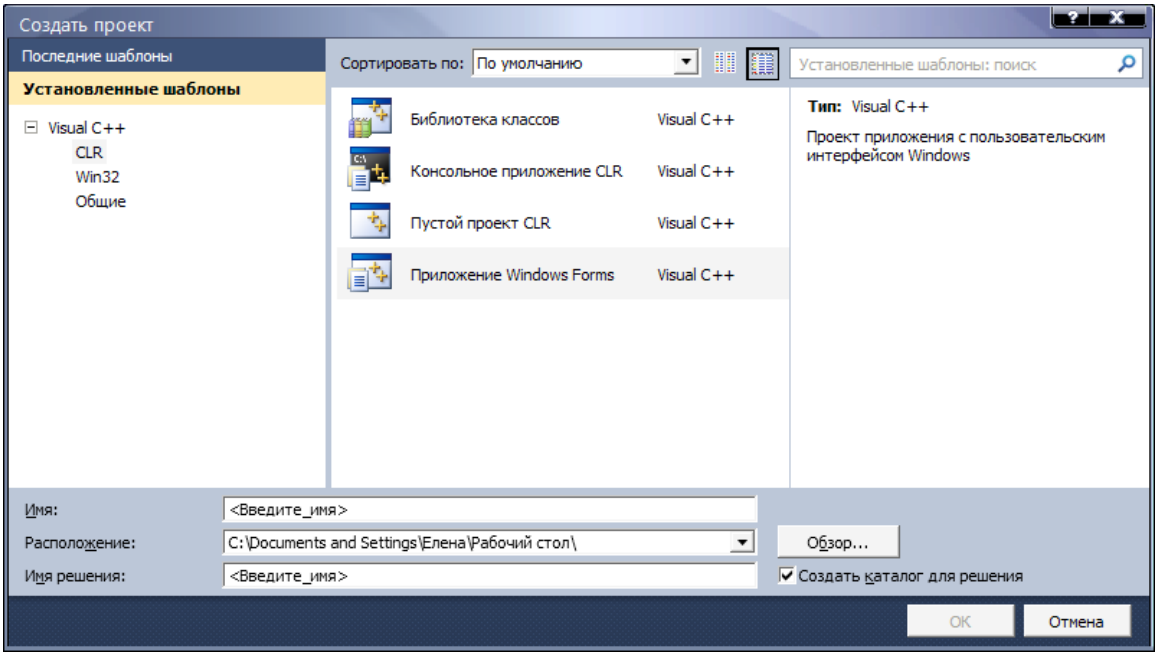


Рис.14.6. *Створення проекту*

У вузлі Visual C++ області встановлених шаблонів необхідно вибрати **CLR**, а потім у області шаблонів вибрати шаблон «**Приложение Windows Forms (Visual C++)».** Після чого необхідно ввести ім'я проекту в поле «**Имя»** і шлях до файлів проекту (*рис.14.6*).

У вікні (*рис.14.1*) відображається *екранна форма* «**Form1»**. Форма відобразиться у вкладці «**Form1.h [Конструктор]».** У формі розташовують компоненти графічного інтерфейсу користувача (елементи управління). Це поля для введення тексту «**TextBox»,** командні кнопки «**Button»,** мітки «**Label» -** які не можуть бути відредаговані користувачем, і інші елементи управління. Візуальне програмування, передбачає просте перетягування елементів за допомогою миші з «**Панели элементов»,** де розташовані всі елементи управління. Це допомагає звести до мінімуму безпосереднє написання програмного коду.

Властивостями кнопки «**button**1» є: ім'я кнопки «**Name**», напис на кнопці «**Text**», розташування кнопки «**Location**» в системі координат форми X, Y, розмір кнопки «**Size**», активність «**Enabled**» і видимість «**Visible**» та ін.

Властивості можна побачити, якщо клацнути правою кнопкою миші в межах форми і вибрати у контекстному меню команду **«Свойства»**, при цьому з'явиться панель властивостей **(***якщо вона не відображається на екрані***).**

Об'єкти не тільки мають властивості, але і обробляються подіями. Подією, наприклад, є клацання на кнопці «**Click»**, клацання в межах форми, завантаження форми «**Load»** в оперативну пам'ять при старті програми.

Управляють тим або іншою подією за допомогою написання процедури обробки події в програмному коді. Для цього спочатку потрібно одержати «порожній» обробник події. Для отримання порожнього обробника цієї події необхідно у властивостях елементу клацнути на значку блискавки «**События**» і в списку всіх можливих подій вибрати подвійним клацанням необхідна подія. Після цього активізується вкладка програмного коду «**Form1.h»**. Перемикатися між вкладками форми і програмного коду можна за допомогою миші або за допомогою комбінації клавіш «**Ctrl+Tab»** (*як це прийнято звичайно при перемиканні між вкладками в Windows).*

Запуск проекту здійснюється командою «**Отладка > Начать отладку»,** або натиснути клавішу **F5**.

Приклад програми знаходження суми 2-х чисел.

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
{
        double a,s,b;
        a=Convert::ToDouble(textBox1->Text);
        b=Convert::ToDouble(textBox2->Text);
        s=a+b;
        label1->Text=Convert::ToString(s);
}
```

Для виведення повідомлень у окремому вікні можна використати метод **Show**.

```
MessageBox::Show("Текст");
```

Тут викликається метод **Show** об'єкту **MessageBox** з текстом. Оператор розширення області дії (::) вказує системі знайти метод **Show** серед методів об'єкту **MessageBox**. Таким чином, об'єкти окрім властивостей мають також і методи, тобто програми, які обробляють об'єкти.

ТЕМА 15. ІНСТРУМЕНТИ ВІЗУАЛЬНОЇ РОЗРОБКИ ДОДАТКІВ

Особливості роботи з формами

Форми є основою додатків. Створення інтерфейсу користувача додатку полягає в додаванні у вікно форми об'єктів C++, які називаються компонентами. Компоненти розташовуються на панелі елементів, виконаній у вигляді розгортаючогося списку.

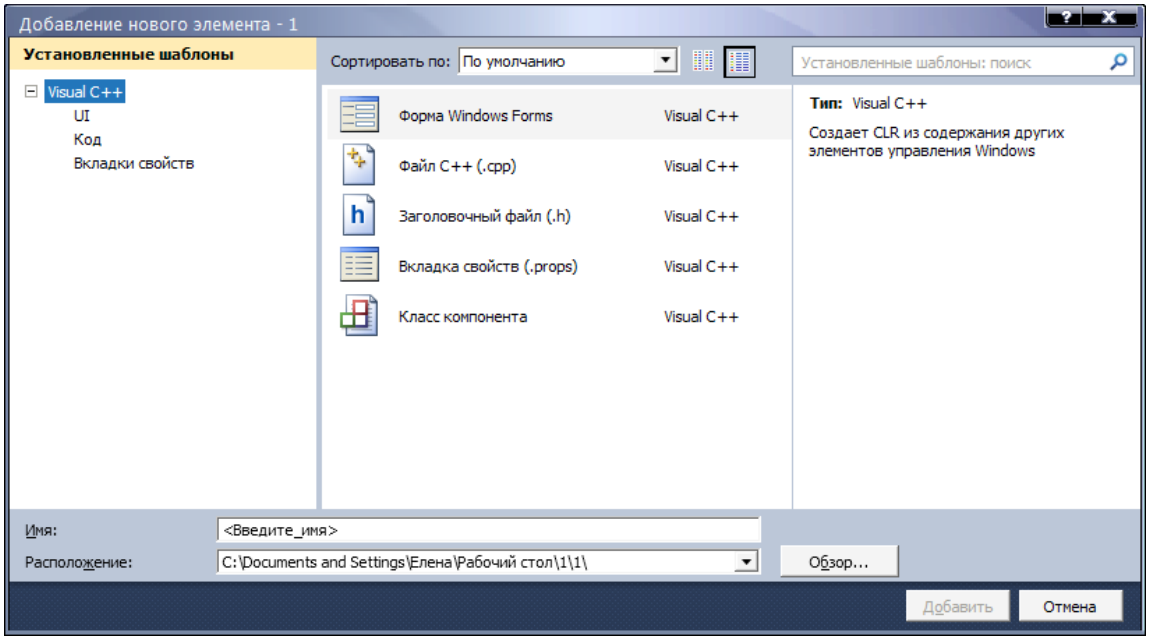


Рис.15.1. Додавання нової форми

Для додавання у проект нової форми необхідно обрати команду «**Проект > Добавить новий елемент…».** У діалоговому вікні обрати **«Форма Windows Forms»** та задати ім'я форми. У результаті з'явиться вкладка **ім'я.h [Конструктор]** поряд з вкладкою конструктора головної форми **Form1.h [Конструктор].**

Форма, що з'являється у проекті при його створенні завантажується першою за замовчуванням, тобто завжди вважається стартовою (*головною*).

Якщо у проект додано форму її необхідно викликати відповідним методом.

Форма викликається на виконання у двох режимах: модальному та немодальному. У модальному режимі – метод форми **ShowDialog()**, у звичайному – методом форми **Show().**

Форма має велику кількість методів, зокрема:

Close() – закрити форму. Для головної форми, це означає закриття додатку.

Hide() – приховати форму (зробити невидимою)

Show() – виведення форми на екран

Характеристика основних компонентів середовища

Компоненти розділяються на видимі (візуальні) і невидимі (невізуальні). Візуальні компоненти з'являються під час виконання точно так, як і під час проектування. Прикладами є кнопки і поля редагування. Невізуальні компоненти з'являються під час проектування як піктограми на формі. Вони не видні під час виконання, але мають певне функціональне навантаження (*наприклад, забезпечують доступ до даних, викликають стандартні діалоги Windows*).

Для додавання компоненту у форму можна вибрати мишею потрібний компонент в палітрі і клацнути лівою клавішею миші в потрібному місці проектованої форми. Компонент з'явиться на формі, і далі його можна переміщати та встановлювати характеристики .

Кожен компонент має три різновиди характеристик: **властивості, події і методи.**

Властивості є атрибутами компоненту, що визначають його зовнішній вигляд і поведінку. Багато властивостей компоненту мають значення, що встановлюється за замовчуванням (*наприклад, висота кнопок*). Властивості компоненту відображаються а сторінці властивостей Можна визначити властивості під час проектування або написати код для видозміни властивостей компоненту під час виконання додатку (*видимий чи не видимий компонент в залежності від виконання умови*).

Сторінка **подій** показує список подій, розпізнаваних компонентом (*програмування для операційних систем з графічним інтерфейсом користувача*). Кожен компонент має свій власний набір обробників подій. Створюючи обробник події, ви доручаєте програмі виконати написану функцію, якщо ця подія відбудеться.

Метод - є функцією, яка пов'язана з компонентом, і яка оголошується як частина об'єкту. Створюючи обробники подій, можна викликати методи, використовуючи наступну нотацію: **->**.

Наприклад: **Form1->Close();** *//закрити форму 1.*

У бібліотеці візуальних компонентів існує множина компонентів, що дозволяють відображати, вводити та редагувати текстову інформацію

Компонент Button

Компонент **Button** є командною кнопкою.

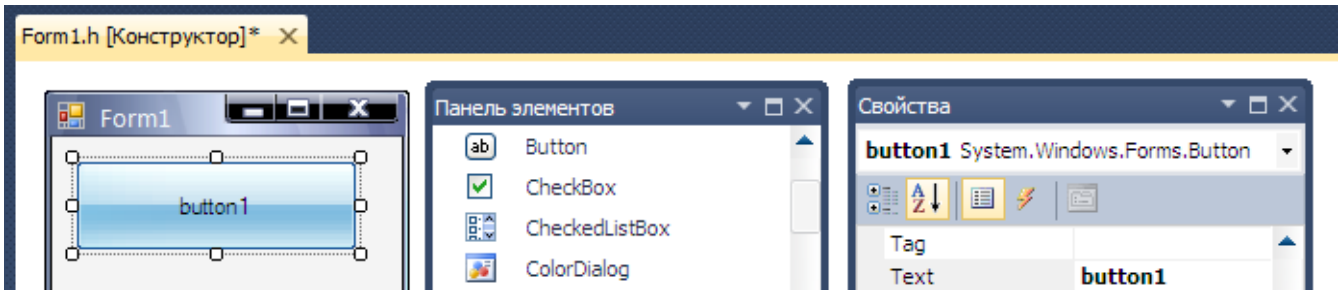


Рис.15.2. Компонент ***Button***

Кожен компонент має визначений набір властивостей методів та подій.

Таблиця 15.1. Характеристики компоненту **Button**

Name	Ім'я компоненту. Використовується в програмі для доступу до властивостей компоненту
Text	Текст (напис) на кнопці
TextAlign	Положення тексту на кнопці. Напис може розташовуватися в центрі (**MiddleCenter**), бути притиснута до лівої (**MiddleLeft**) або правої (**MiddleRight**) межі.
Location	Положення кнопки на поверхні форми.
Size	Розмір кнопки
Font	Шрифт тексту
ForeColor	Колір тексту
Enabled	Ознака доступності кнопки. Кнопка доступна, якщо значення властивості рівне **True**, і недоступна (*наприклад, подія* ***Click*** *на кнопці не виникає*), якщо значення властивості рівне **False**
Visible	Дозволяє приховати кнопку (**False**) або зробити її видимою (**True**)
Image	Картинка на поверхні форми. Рекомендується використовувати gif-файл, в якому визначений прозорий колір фону
ImageAlign	Положення картинки на кнопці

Anchor	Закріплення позиції компоненту
DialogResult	Забезпечує роботу з модальними формами
TabStop	Відключення отримання фокуса за допомогою клавіші Tab (=False)
Події компоненту Button	
Click	Виникає, коли на кнопці клікають мишею
Enter	Виникає, коли кнопка отримує фокус, тобто кнопка стає активною
Методи Button	
Hide()	Приховує кнопку
Focus()	Робить кнопку активною
Show()	Відображає кнопку

Компонент Label

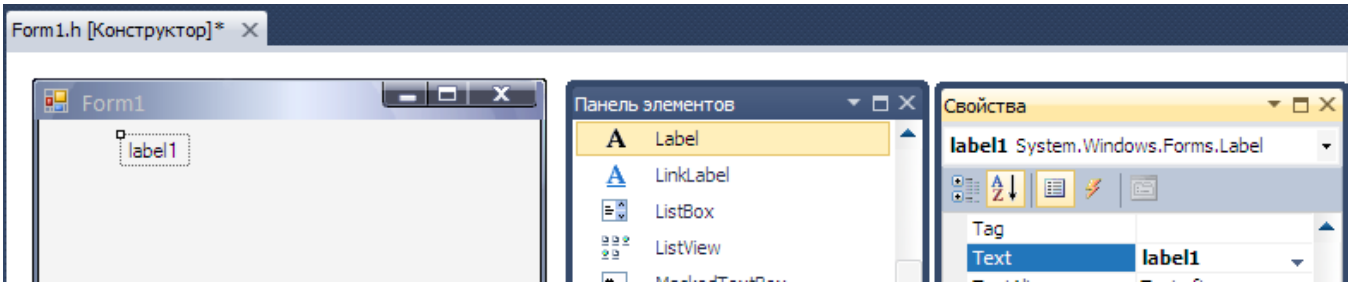


Рис.15.3. Компонент ***Label***

Компонент **Label** призначений лише для відображення текстової інформації (редагування тексту не можливо). Задати текст, що відображається в полі компоненту, можна як під час розробки форми, так і під час роботи програми, присвоївши значення властивості **Text**.

Компонент може бути використаний також для виведення зображень та ідентифікації об’єктів.

Таблиця 15.2. Властивості компоненту **Label**

Name	Ім'я компоненту. Використовується в програмі для доступу до властивостей компоненту
Text	Текст (напис) у полі
TextAlign	Положення тексту у полі компоненту (вирівнювання)
Location	Положення компоненту на поверхні форми
Size	Розмір компоненту
Font	Шрифт тексту

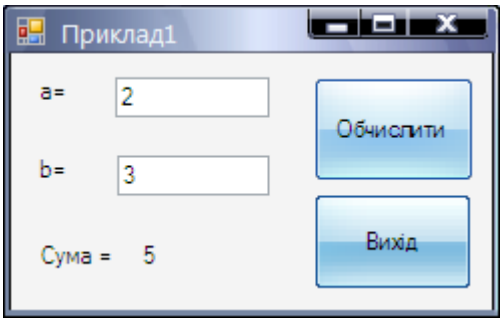
ForeColor	Колір тексту
BackColor	Колір фону
BorderStyle	Вид рамки (межі) компоненту

Щоб в полі компоненту **Label** вивести числове значення, це значення необхідно перетворити в рядок. Зробити це можна за допомогою методу **ToString**.

Convert::ToString(s) - *перетворення змінної до рядкового типу*

Колір тексту «**ForeColor**» і фону «**BackColor**» можна задати, вказавши назву кольори (**Color::Red, Color::Blue, Color::Green** і т. д.), або елемент колірної схеми операційної системи. У другому випадку колір буде "прив'язаний" до поточної колірної схеми операційної системи і автоматично змінюватиметься при кожній її зміні. За замовчанням для елементів управління використовується другий спосіб кодування кольору. Колір фону може бути "прозорим" (**Color::Transparent**).

Приклад програми обчислення суми двох чисел




```
private: System::Void button1_Click(…)
{
  double a,s,b;
  a=Convert::ToDouble(textBox1->Text);
  b=Convert::ToDouble(textBox2->Text);
  s=a+b;
  label1->Text=Convert::ToString(s);
}
```

//**Convert::ToDouble** – перетворення змінної до дійсного типу

//**Convert::ToString** - перетворення змінної до рядкового типу

Компонент TextBox

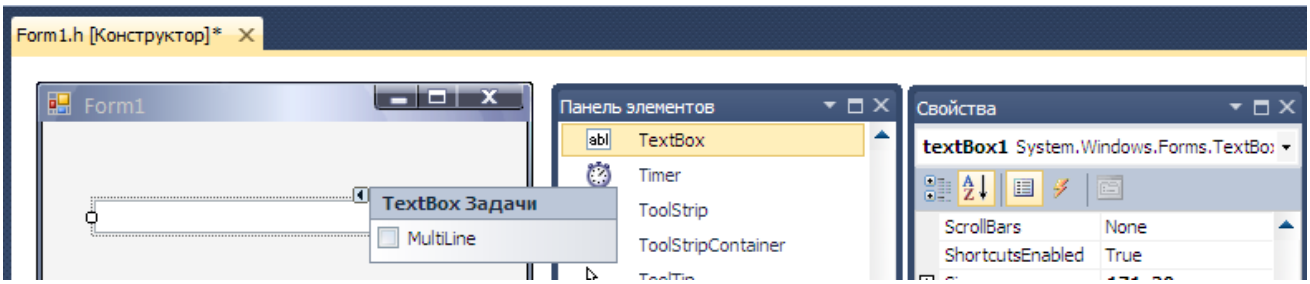


Рис.15.4. Компонент ***TextBox***

Компонент **TextBox** призначений для введення даних з клавіатури. Залежно від налаштування компоненту, в полі редагування можна вводити один або декілька (**Multiline = True**) рядків тексту.

Всі дані ведені у текстове поле сприймаються програмою, як набір сиволів. Для перетворення символів до дійсного типу використовують методи **ToDouble** або **ToInt32** (*цілочисловий тип*).

Convert::ToDouble – *перетворення змінної до дійсного типу*

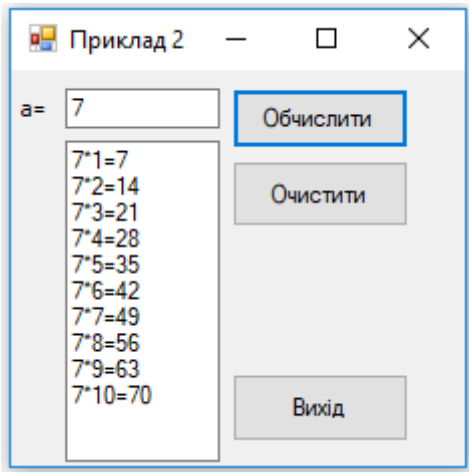
Таблиця 15.3. Властивості компоненту **TextBox**

Name	Ім'я компоненту. Використовується в програмі для доступу до компоненту і його властивостей, зокрема до тексту, який знаходиться в полі редагування
Text	Текст, який знаходиться в полі редагування
Location	Положення компоненту на поверхні форми
Size	Розмір компоненту
Font	Шрифт, використаний для відображення тексту в полі компоненту
ForeColor	Колір тексту
BackColor	Колір фону поля компоненту
BorderStyle	Вид рамки (межі) компоненту. Межа компоненту може бути звичайною (**Fixed3D**), тонкою (**FixedSingle**). Межа навколо компоненту може бути відсутньою (в цьому випадку значення властивості рівне **None**)
TextAlign	Спосіб вирівнювання тексту в полі компоненту.
MaxLength	Максимальна кількість символів, яку можна ввести в поле компоненту
PasswordChar	Символ, який використовується для відображення символів, що вводяться користувачем (*введений користувачем рядок знаходиться у властивості* **Text**)
Multiline	Вирішує (**True**) або забороняє (**False**) введення декількох рядків тексту
Lines	Масив рядків, елементи якого містять текст, що знаходиться в полі редагування, якщо компонент знаходиться в режимі **MultiLine**. Доступ до рядка здійснюється по номеру. Рядки нумеруються з нуля
SkroolBars	Задає смуги прокрутки, що відображаються: **Horizontal** - горизонтальна; **Vertical** - вертикальна;

Методи компоненту TextBox

AppendText ()	Додає текст до поточного тексту у вікні компоненту
Clear ( )	Очищення поля
Copy ( )	Копіювання обраного рядка у буфер обміну
Cut ( )	Вирізання виділеного блоку тексту у буфер обміну
Focus ( )	Встановлення фокуса
Hide ( )	Приховування компоненту
Paste ( )	Заміняє поточну вибірку в полі вмістом буферу обміну
Select ( )	Вибирає текст у компоненті
Show ( )	Робить компонент видимим

Приклад програми виведення таблиці множення на введене з клавіатури число.




```
//Кнопка «Обчислити»
private: System::Void button1_Click(…)
{
  double a, r;
  a=Convert::ToDouble(textBox1->Text);
  for(int i=1;i<=10;i++)
  {
     r=i*a;
     textBox2->AppendText(a+"*"+i+"="+r+"\r\n");
  }
}

// Кнопка «Очистити». Очистка полів форми.
private: System::Void button3_Click(…)
{
  textBox1->Clear();
  textBox2->Clear();
}

// Кнопка «Вихід» . Закриття додатку.
private: System::Void button2_Click(…)
{
    Close();
}
```

Компоненти CheckBox і RadioButton

Радіокнопки утворюють групи взаємозв'язаних індикаторів, з яких звичайно може бути вибраний тільки один. Вони використовуються для вибору користувачем однієї з декількох взаємовиключних альтернатив

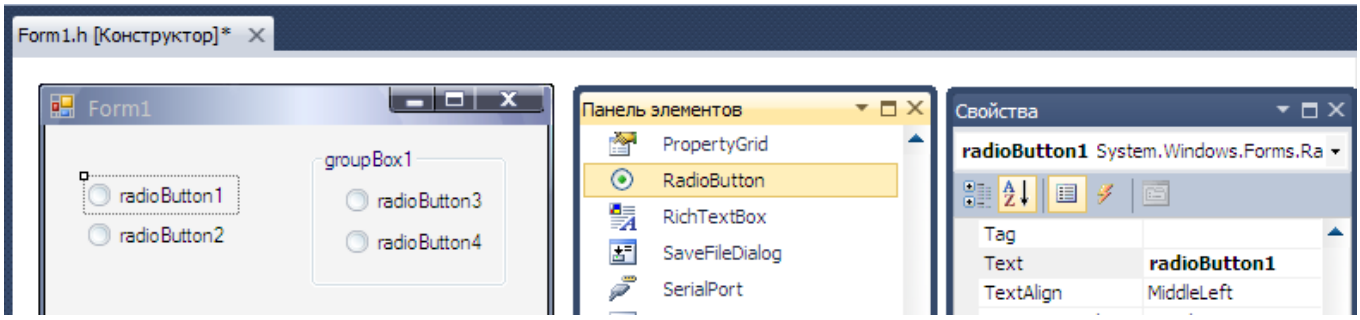


Рис.15.5. Компонент ***RadioButton***

Окрема радіокнопка **RadioButton** особливого сенсу не має. А радіокнопки мають сенс, коли вони взаємодіють один з одним в групі.

Компонент **GroupBox** (*вкладка* ***«Контейнери»*** *панелі елементів*) – є контейнером, об'єднуючим групу зв'язаних елементів управління, таких, як радіокнопки **RadioButton** або індикатори **Checkbox** і т.д.

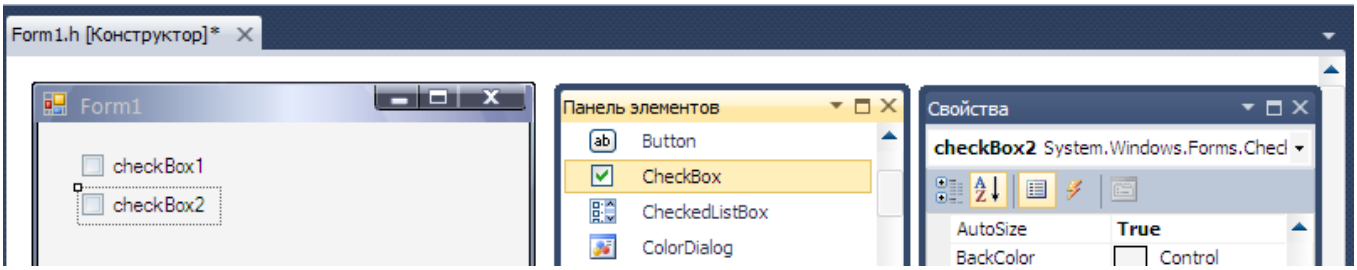


Рис.15.6. Компонент ***CheckBox***

Індикатори з прапорцем **CheckBox** використовуються у додатках для того, щоб вмикати та вимикати опції, або для індикації стану. При кожному клацанні користувача на індикаторі його стан змінюється, проходячи в загальному випадку послідовно через три значення: виділення (*поява чорної галочки*), проміжне (*сіре вікно індикатора і сіра галочка*) і не виділене (*порожнє вікно індикатора*). Цим трьом станам відповідають три значення властивості компоненту.

Основні властивості компоненту CheckBox

Checked – властивість, яка визначає чи включено прапорець.

CheckState – встановлює 3 вида станів компоненту:

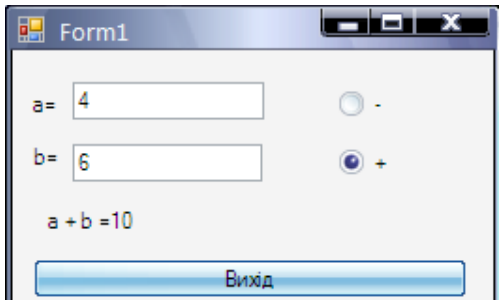
- **Checked** - Прапорець увімкнуто;
- **Uncheked** - Прапорець вимкнуто
- **Indeterminate** - Невизначений стан (сірий колір прапорця).

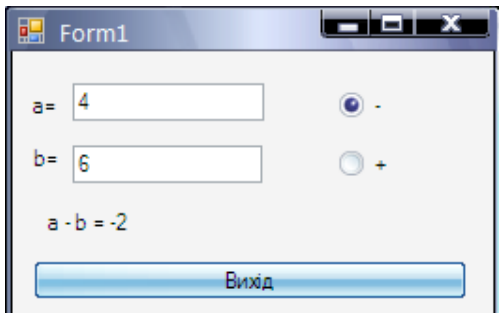
CheckAlign – властивість, що дозволяє відкривати розгортаючийся список, для вибору розташування прапорця у полі компоненту (9 позицій).

AutoCheck - властивість визначає, чи повинно автоматично змінюватися стан прапорця в результаті клацання на його зображенні. За умовчанням значення рівне True.

Стан прапорця змінюється в результаті клацання на його зображенні (якщо значення властивості **AutoCheck** = **True**). При цьому виникає подія **CheckedChanged**, а потім подія **Click**. Якщо значення властивості **AutoCheck** = **False**, то в результаті клацання на прапорці виникає подія **Click**, а потім, якщо процедура обробки цієї події змінить стан кнопки, виникає подія **CheckedChanged**.

Приклад програми обчислення суми або різниці двох чисел.






```
private: System::Void radioButton1_CheckedChanged(…)
{
    int a,b,s;
    a=Convert::ToInt32(textBox1->Text);
    b=Convert::ToInt32(textBox2->Text);
    if(radioButton1->Checked)
    {
      s=a+b;
      label1->Text="a+b ="+Convert::ToString(s);
    }
}

private: System::Void radioButton2_CheckedChanged(…)
{
  int a,b,s;
  a=Convert::ToInt32(textBox1->Text);
  b=Convert::ToInt32(textBox2->Text);
  if(radioButton2->Checked)
  {
    s=a-b;
    label1->Text="a-b="+Convert::ToString(s);
  }
}
```

Компонент ComboBox

Компонент **ComboBox** є комбінацією поля редагування і списку, що дозволяє вводити дані з клавіатури або вибору значення в списку.

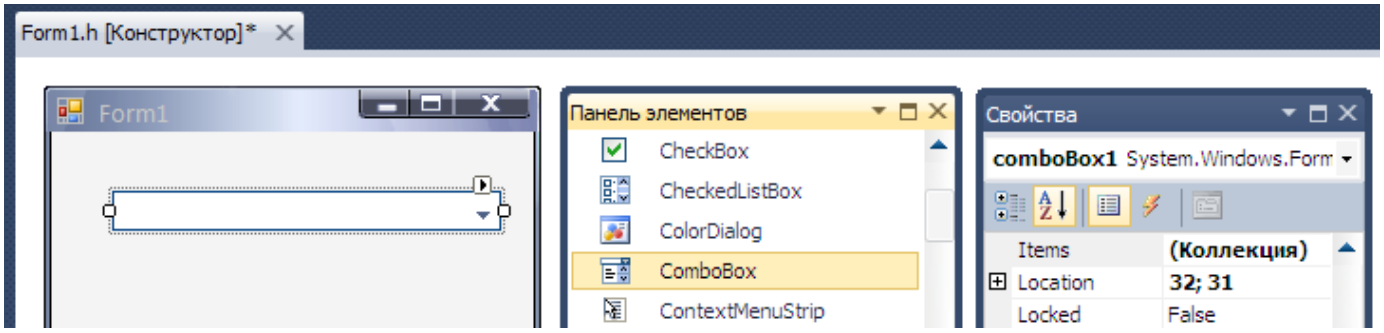


Рис.15.7. Компонент ***ComboBox***

Властивості компоненту ComboBox

Items – елементи списку «**Коллекция**».

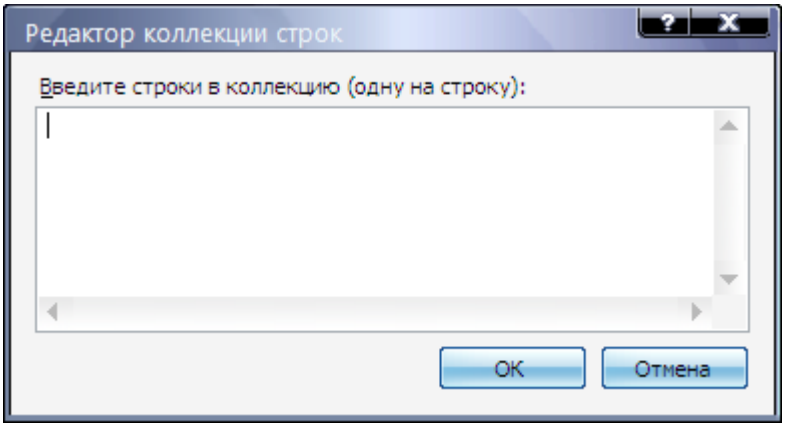


Рис.15.8. Налаштування властивості ***Items***

Items.Count – кількість елементів списку

Items.SelectedIndex – номер елементу обраного у списку. Якщо жоден елемент не вибрано значення властивості –1.

Sorted – автоматичне сортування списку при додаванні нового елементу

Список, що відображається в полі компоненту, можна сформувати під час роботи програми або під час створення форми за допомогою властивості **Items** у вікні «**Редактор коллекции строк** » (*рис.15.8*).

До методів компоненту відносяться:

Focus( ), Hide( ), Clear( ), Add( ), IndexOf( ) *(виводить номер рядка в полі компоненту).*

Щоб сформувати список під час роботи програми, треба застосувати метод **Add** до властивості **Items**.

Наприклад **comboBox1->Items->Add("**рядок1**");** ..//*індекс ряда1 = 0*

comboBox1->Sorted = true;

Компонент ListBox

Компонент представляє собою список у якому обирається необхідний елемент. Більшість властивостей аналогічні компоненту **ComboBox**.

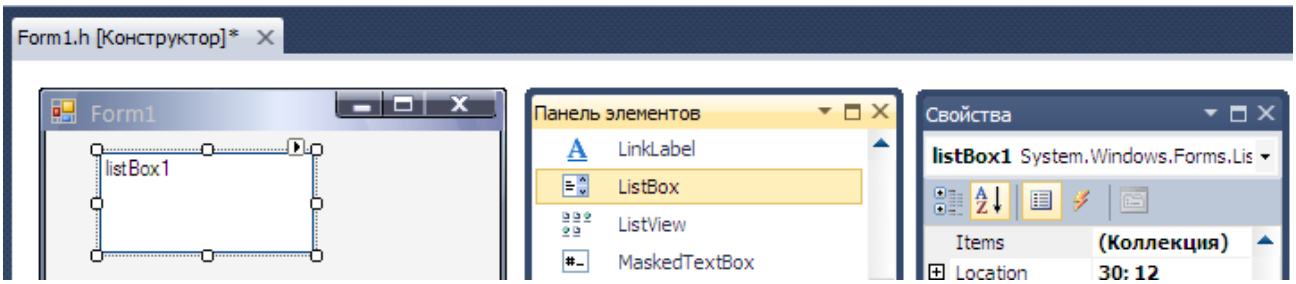


Рис.15.9. Компонент ***ListBox***

Список, що відображається в полі редагування, можна сформувати під час створення форми або під час роботи програми. Щоб сформувати список під час створення форми, треба клацнути на кнопці з трьома крапками в рядку властивості **Items** і у вікні «**Редактор коллекции строк** » (*рис.15.8*) ввести елементи списку. Формування списку під час роботи програми забезпечує метод **Add** властивості **Items**.

Властивість **MultiColumn = true** дозволяє виводити компоненти списку у декілька стовпців, при цьому з'являється горизонтальна смуга прокручування.

SelectionMode - властивість, яка задає кількість елементів можна обрати за один раз.

Властивість **SelectedIndex** повертає цілочислове значення, що відповідає першому елементу у списку обраних. Якщо вибірка порожня =-1.

Властивість **SelectedItem** повертає обраний елемент у вигляді текстового рядку.

Щоб додати або видалити рядок зі списку використовують методи:

Add() – додати елемент у кінець списку

Insert() – вставити елемент у список

Clear() – очистити список

Remove() – знищити заданий елемент зі списку

ТЕМА 16. РОЗРОБКА ГРАФІЧНОГО ІНТЕРФЕЙСУ КОРИСТУВАЧА

Компонент MenuStrip

Компонент **MenuStrip** є головним меню програми.

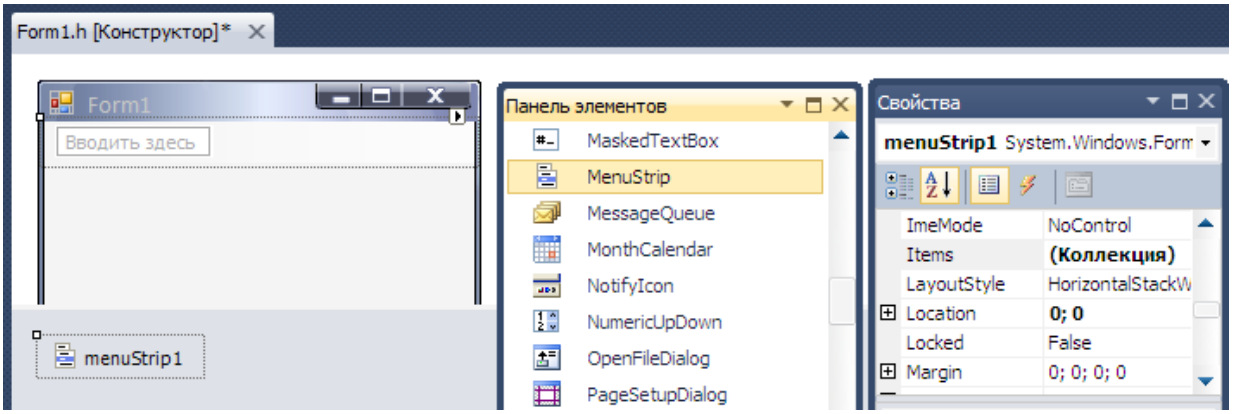


*Рис.16.1. Компонент **MenuStrip***

Після того, як у форму буде доданий компонент **MenuStrip**, у верхній частині форми з'являється рядок меню, на початку якого знаходиться область введення тексту.

Окрім команди в меню можна додати роздільник - горизонтальну лінію. Роздільник звичайно служить для об'єднання в групи однотипних команд. Щоб додати його в список команд, слід виділити команду, перед якою треба помістити роздільник, і з контекстного меню вибрати команду «**Вставка-Separator».**

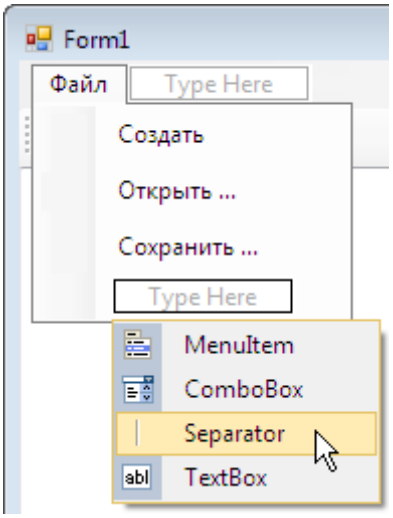


Рис.16.2. Створення пунктів меню

Після того, як структура меню буде сформована, можна виконати налаштування меню шляхом зміни значень властивостей пунктів меню (об'єктів типу **ToolStripMenuItem**).

Властивості об'єкту **ToolStripMenuItem**

Text - Назва елементу меню

Image - Картинка, яка відображається поряд з командою.

Enabled - Ознака доступності елементу меню.

Checked - Ознака того, що елемент меню вибраний. Якщо значення властивості = **True**, то елемент позначається галочкою. Властивість **Checked** звичайно застосовується для тих елементів меню, які використовуються для відображення параметрів, наприклад, пункт меню **Відображати рядок стану.**

ShortcutKeys - властивість визначає комбінацію клавіш, натиснення якої активує виконання команди.

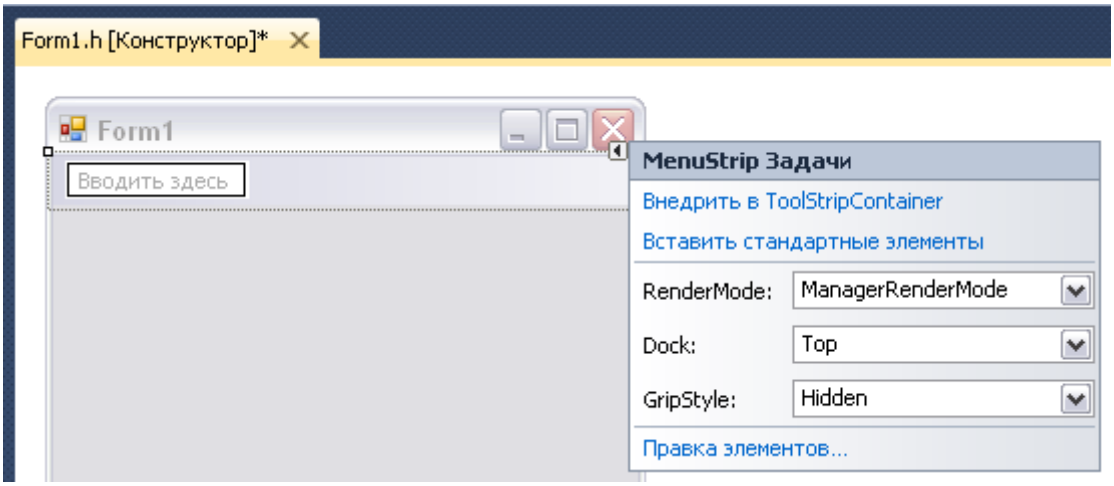


Рис.16.3. Налаштування головного меню

Діалогове вікно «**MenuStripЗадачи**» забезпечує доступ до типових команд та властивостей.

- **Внедрить ToolStripContainer** – дозволяє розмістити меню у спеціальний контейнер. Контейнер – це оберт з власним набором властивостей, встановлення яких дозволяє створювати зручне меню.
- **Внедрить стандартные элементы** – додає загальноприйняті опції меню.
- **RenderMode** – опція дає можливість вибору зі списку способу зображення меню (системне, професійне, управляєме).
- **Doc** – схема розташування меню відносно сторін форми.
- **GripStyle** – вибір зі списку стиля полоси меню (видима або невидима спеціальна пунктирна лінія)

- **Правка элементов** – встановлення опцій меню. Дана команда відкриває діалогове вікно для встановлення оцій меню, додавання нових опцій та їх реорганізація.

Основні властивості компоненту MenuStrip

BackgroundImage – задає фонове зображення компоненту.

Items – дозволяє викликати діалогове вікно для налаштування опцій меню.

LayoutStyle - стиль розміщення меню.

Checked – властивість, яка визначає чи вибрано дану команду меню.

Компонент ContextMenuStrip

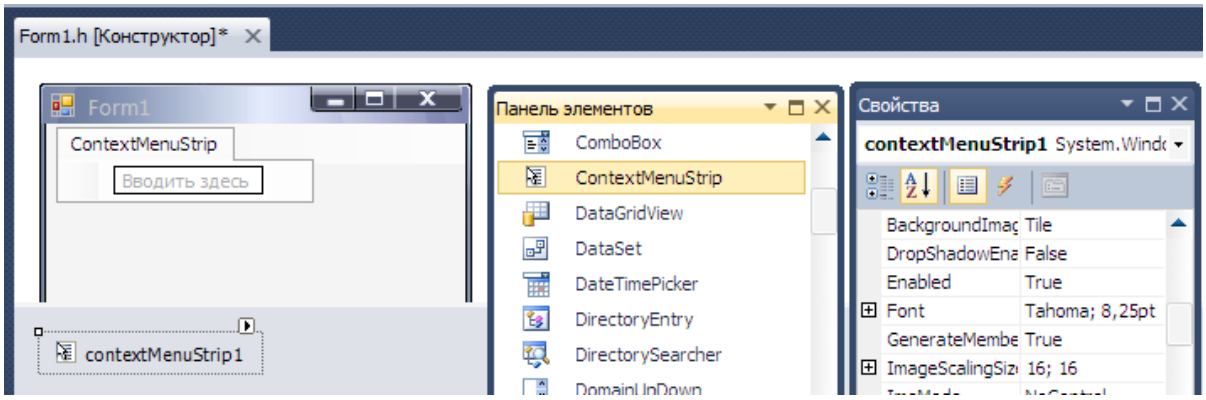


Рис.16.4. Компонент ***ContextMenuStrip***

Даний компонент може бути прив'язаний до будь якого компоненту, який має властивість **ContextMenuStrip.** Коли компонент поміщають у форму його ім'я буде видно у будь якому компоненті форми. Встановлення опцій контекстного меню аналогічне головному меню.

Компонент ToolStrip

Компонент **ToolStrip** - смуга (панель) інструментів використовується для розміщення в ньому інших компонентів. У панель інструментів можна помістити командну кнопку, поле відображення тексту, поле редагування, список, що розкривається.

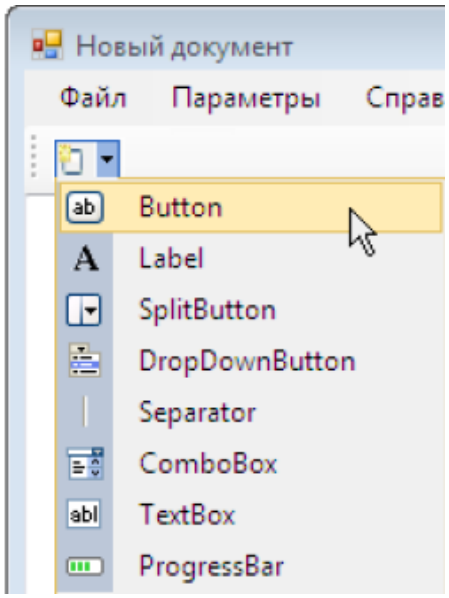


Рис.16.5. Створення панелі інструментів

Властивості компоненту **ToolStrip**

Buttons – (Items) колекція компонентів, що знаходяться у панелі інструментів.

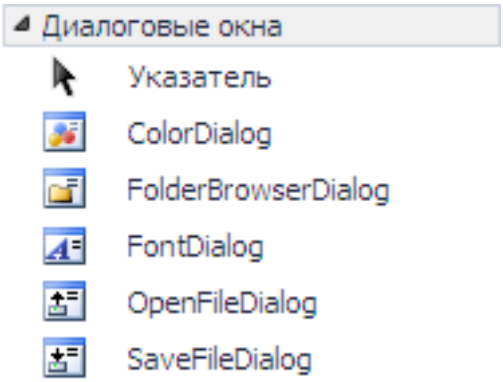
Dock – межа батьківського компоненту (форма) до якої прив'язано панель інструментів.

Visible – дозволяє приховати або відобразити панель інструментів.

Enabled – доступ до компонентів панелі інструментів.

Для того, щоб в панель інструментів додати, наприклад, командну кнопку, треба вибрати компонент **ToolStrip**, клікнути на значку розкриваючогося списку, який відображається в його полі, і вибрати **Button**. Після цього треба виконати налаштування доданої кнопки **toolStripButton** - задати значення властивостей.

Стандартні діалогові вікна



Компонент OpenFileDialog

Компонент представляє собою діалогове вікно «**Открыть**».

При додаванні на форму компонент розташовується у окремій області вікна дизайнера форм. Якщо необхідно відкрити папку, а не файл – використовують клас **FolderBrowserDialog**.

Діалогове вікно для вибору файлу з'являється у режимі виконання додатку при виклику метода **ShowDialog().** Коли користувач у діалоговому вікні натисне на кнопку «**Открыть**», метод **ShowDialog()** повертає значення **DialogResult,** яке порівнюється зі значенням такої ж властивості у форми. Для форми значення властивості =ОК.

Основні властивості компоненту:

Multiselect – дозволяє вибрати групу файлів.

Filter – умовна фільтрація файлу.

```
Text files (*.txt)|*.txt|All files (*.*)|*.*
```

ShowReadOnly – дає можливість з'являтися індикатору поряд з файлом, якщо він має тип «тільки для читання»

Компонент SaveFileDialog

Компонент для відображення діалогового вікна «**Сохранить**» та збереження файлу. Але компонент дає лише шлях до місця розташування файлу, написання коду, який відповідає за збереження даних відводиться програмісту.

Діалогове вікно з'являється у режимі виконання додатку при виклику метода **ShowDialog().** Більшість властивостей компоненту аналогічна компоненту **OpenFileDialog**. Для відображення вікна «Сохранить как» необхідно значення властивості **OverwritePromp = true.**

Коли користувач обирає ім'я файлу та натискає кнопку збереження метод **ShowDialog**() заносить у властивість **FileName** компоненту ім'я файлу та шлях до нього. При цьому жодного перезапису файлу не відбувається.

Компонент ColorDialog - викликає діалогове вікно вибору кольору. Значення кольору записується у властивість **Color**.

Компонент FontDialog – виклик вікна налаштування параметрів шрифту. Обрані параметри присвоюються властивості компоненту **Font**.

Компонент PrintDialog – відкриває діалогове вікно налаштування параметрів друку

ТЕМА 17. ГРАФІКА У MICROSOFT VISUAL C++ EXPRESS

Компонент PictureBox

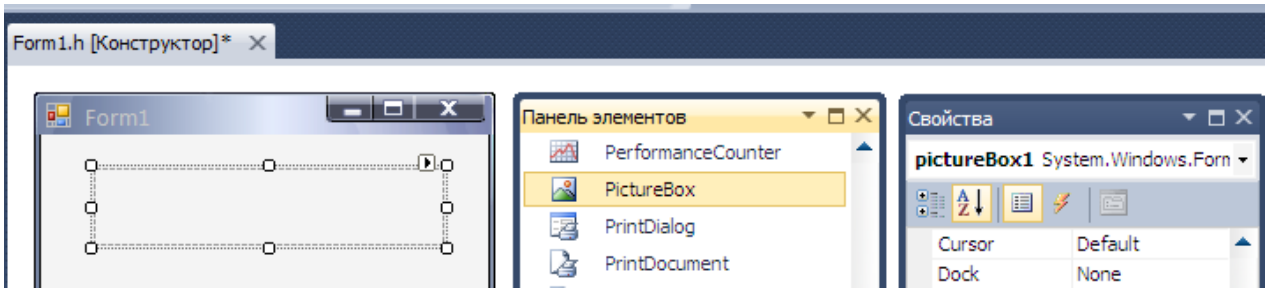


Рис.17.1. Компонент ***PictureBox***

Відображення графіки забезпечує компонент **PictureBox .** Графічна поверхня компоненту **PictureBox** є об'єктом **Graphics**, методи якого і забезпечують виведення графіки. Таким чином, для того, щоб на поверхні компоненту **PictureBox** з'явилася лінія, прямокутник, коло або завантажена з файлу ілюстрація, необхідно викликати відповідний метод об'єкту **Graphics**.

Таблиця 17.1. Властивості компоненту ***PictureBox***

Властивість	Опис властивостей компоненту
Image	Ілюстрація, яка відображається в полі компоненту
SizeMode	Режим відображення ілюстрації (спосіб масштабування), якщо розмір ілюстрації не відповідає розміру компоненту: **Normal** - масштабування не виконується (якщо розмір компонента менше розміру ілюстрації, то відображається тільки частина ілюстрації); **StretchImage** - виконується масштабування ілюстрації таким чином, що вона займає всю область відображення (якщо розмір компоненту не пропорційний розміру ілюстрації, вона буде спотворена); **AutoSize** - розмір компоненту автоматично змінюється і відповідає розміру ілюстрації; **CenterImage** - центрування ілюстрації в полі компоненту, якщо розмір ілюстрації менше розміру області відображення; **Zoom** - виконується масштабування так, щоб ілюстрація в полі компоненту відображалася без спотворення
Location	Положення компоненту (області відображення ілюстрації) на поверхні форми. Що уточнює властивість X визначає відстань від лівої межі області до лівої межі форми, що уточнює властивість Y - від верхньої межі області до верхньої межі клієнтської області форми (нижньої межі заголовка)

Dock	Спосіб прив'язки компоненту до компоненту-батька. Компонент може бути прив'язаний до лівої межі (**Left**), правої (**Right**), верхньої (**Top**), нижньої (**Bottom**) або займати всю вільну область компоненту-батька (**Fill**)
Size	Розмір компоненту (області відображення ілюстрації). Що уточнює властивість **Width** визначає ширину області, **Height** - висоту
Visible	Ознака указує, чи відображається компонент і, відповідно, ілюстрація на поверхні форми
BorderStyle	Вид межі компоненту: **None** - межа не відображається; **FixedSingle** - тонка; **Fixed3D** - об'ємна
Graphics	Поверхня, на яку можна виводити графіку (доступ до графічної поверхні компоненту є у процедури обробки події **Paint**)

Ілюстрацію, що відображається в полі компоненту **PictureBox**, можна задати під час розробки форми або завантажити з файлу під час роботи програми. Щоб задати ілюстрацію під час створення форми, треба в рядку властивості **Image** клацнути на кнопці з трьома крапками і в стандартному вікні, що з'явилося, «**Открыть**» вибрати файл ілюстрації. Середовище розробки створить бітовий образ ілюстрації і помістить його у файл ресурсів (*таким чином, надалі файл ілюстрації програмі буде не потрібен*). Завантаження ілюстрації з файлу під час роботи програми забезпечує метод **FromFile**. Як параметр методу треба вказати ім'я файлу ілюстрації.

Наприклад

```
pictureBox1->Image = System::Drawing::Bitmap::FromFile("d:\Pict.jpg");
```

Дана інструкція забезпечує завантаження і відображення ілюстрації, яка знаходиться у файлі d:\Photo\Pict0025.jpg. Метод **FromFile** дозволяє працювати з файлами BMP, JPEG, GIF, PNG і інших форматів.

Виведення графічних зображень

Доступ до графічної поверхні об'єкту (властивості **Graphics**) є тільки у функції обробки події **Paint**. Подія **Paint** виникає на початку роботи програми, коли її вікно і всі компоненти з'являються на екрані, а також під час роботи програми всякий раз, коли вікно або його частина знов з'являється на екрані, наприклад.

Для створення процедури обробки події **Paint** необхідно вибрати компонент **PictureBox**, потім у вікні **Properties** відкрити вкладку **Events** і в поле події **Paint** ввести ім'я функції обробки події (*або зробити подвійне клацання лівою кнопкою миші*).

Графічна поверхня складається з окремих точок - пікселів. Положення точки на графічній поверхні характеризується горизонтальною (X) і вертикальною (Y) координатами. Координати точок відраховуються від лівого верхнього кута і зростають зліва направо (координата `X`) і зверху вниз (координата `Y`). Ліва верхня точка графічної поверхні має координати (0, 0). Розмір графічної поверхні форми відповідає розміру клієнтської області (*тобто без урахування висоти області заголовка і ширини меж)* форми (властивість **ClientSize**), а розмір графічної поверхні компоненту **PictureBox** - розміру компоненту.

Олівці і пензлі

Методи малювання графічних примітивів (наприклад, **DrawLine** - лінія, **DrawRectangle** - прямокутник, **FillRectangle** - область) використовують *олівці* і пензлі. Олівець (об'єкт **Pen**) визначає вид лінії, пензель (об'єкт **Brush**) - вид заливки області.

Метод

DrawLine(Pens::Black,10,20,100,20);

малює з точки (10, 20) горизонтальну лінію, використовуючи чорний (**Black**) олівець із стандартного набору олівців (**Pens**).

Метод

FillRectangle(Brushes::Green,5,10,20,20);

за допомогою стандартного пензля зеленого (**Green**) кольору малює зелений квадрат, лівий верхній кут якого знаходиться в точці (5, 10).

Олівці і пензлі визначені в просторі імен **System::Drawing**.

Олівець

Олівець визначає вид лінії - **колір, товщину і стиль**. У розпорядженні програміста є два набори олівців: **стандартний і системний**. Також програміст може створити власний олівець.

Стандартний набір олівців - це кольорові олівці (всього їх 141), які малюють безперервну лінію товщиною в *один піксель*.

Таблиця 17.2. Кольорова гамма олівців

Олівці	Колір
Pens::Red	Червоний
Pens::Orange	Помаранчевий
Pens::Yellow	Жовтий
Pens::Green	Зелений
Pens::LightBlue	Блакитний
Pens::Blue	Синій
Pens::Purple	Пурпуровий
Pens::Black	Чорний
Pens::LightGray	Сірий
Pens::White	Білий
Pens::Transparent	Прозорий

Системний набір олівців - це олівці, колір яких визначається поточною колірною схемою операційної системи і співпадає з кольором якого-небудь елементу інтерфейсу користувача. Наприклад, колір олівця **SystemPens::ControlText** співпадає з кольором, який в поточній колірній схемі використовується для відображення тексту на елементах управління (командних кнопках і ін.), а колір олівця **SystemPens::WindowText** - з кольором тексту у вікнах повідомлень.

Олівець із **стандартного** (**Pens**) і **системного** (**SystemPens**) наборів малює безперервну лінію товщиною в один піксель. Якщо треба намалювати пунктирну лінію або лінію завтовшки більше одиниці, то необхідно використовувати **олівець програміста**.

Олівець програміста - це об'єкт типу **Pen**, властивості якого визначають вид лінії, що малюється олівцем. Колір, ширину лінії і стиль олівця, створеного програмістом, можна змінити. Щоб це зробити, треба встановити значення відповідної властивості.

Пензель

Пензлі використовуються для зафарбовування внутрішніх областей геометричних фігур.

Наприклад

e->Graphics->FillRectangle(Brushes::DeepSkyBlue, x, y, w, h);

Дана інструкція малює зафарбований прямокутник. **Brushes::DeepSkyBlue** - це стандартний пензель темно-небесного кольору. Параметри **x, y** визначають положення прямокутника; **w, h** - його розмір.

У розпорядженні програміста є три типу пензлей: **стандартні, штрихові і текстурні.**

Стандартний пензель зафарбовує область одним кольором (суцільне зафарбовування). Всього є 140 пензлів.

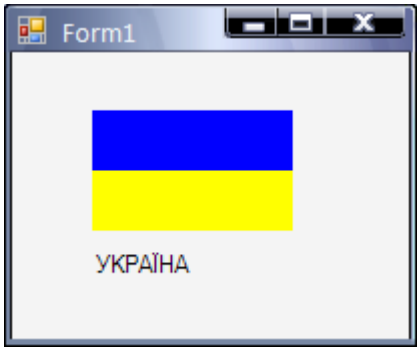
Таблиця 17.3. Кольорова гамма пензлів

Пензлі	Колір
Brushes::Red	Червоний
Brushes::Orange	Помаранчевий
Brushes::Yellow	Жовтий
Brushes::Green	Зелений
Brushes::LightBlue	Блакитний
Brushes::Blue	Синій
Brushes::Purple	Пурпуровий
Brushes::Black	Чорний
Brushes::LightGray	Сірий
Brushes::White	Білий
Brushes::Transparent	Прозорий

Штриховий пензель (**HatchBrush**) зафарбовує область шляхом штрихування. Область може бути заштрихована горизонтальними, вертикальними або лініями під кутом різного стилю і товщини.

Текстурний пензель (**TextureBrush**) є малюнком, який звичайно завантажується під час роботи програми з файлу (bmp, jpg або gif) або з ресурсу. Зафарбовування області виконується шляхом дублювання малюнка у середині області.

Приклад програми «Зображення прапора України».




```
private: System::Void pictureBox1_Paint(…)
{
   int x =10, y=10; // лівий верхній кут
   int w =100, h=30; // ширина та висота смуг
   e->Graphics->FillRectangle(Brushes::Blue,x,y,w,h);  // блакитна смуга
   y += h;
   e->Graphics->FillRectangle(Brushes::Yellow,x,y,w,h);  // жовта смуга
   // Підпис. Використовується шрифт, заданий властивістю Font форми.
   e->Graphics->DrawString("УКРАЇНА",this->Font, Brushes::Black, 10, 80);
}
```

Графічні примітиви

Будь-яка картинка, креслення, схема є сукупність графічних примітивів: крапок, ліній, кіл, дуг, тексту і ін. Побудова графічних примітивів на графічній поверхні (**Graphics**) виконують відповідні методи.

Таблиця 17.4. Методи побудови графічних примітивів

Метод	Дія
DrawLine(Pen, x1, y1, x2, y2)	Лінія. Параметр **Pen** визначає колір, товщину і стиль лінії; параметри x1, y1, x2, y2 або p1 і p2 - координати точок почала і кінця лінії
DrawRectangle(Pen, x, y, w, h)	Прямокутник. Параметр **Pen** визначає колір, товщину і стиль межі прямокутника: параметри **x, y** - координати лівого верхнього кута; параметри **w** і **h** задають розмір прямокутника
FillRectangle(Brush, x, y, w, h)	Малює зафарбований прямокутник. Параметр **Brush** визначає колір і стиль зафарбовування прямокутника; параметри **x, y** - координати лівого верхнього кута; параметри **w і h** задають розмір прямокутника

DrawEllipse(Pen, x, y, w, h)	Малює еліпс (контур). Параметр **Pen** визначає колір, товщину і стиль лінії еліпса; параметри **x, y, w, h** - координати лівого верхнього кута і розмір прямокутника, усередині якого викреслюється еліпс
FillEllipse(Brush, x, y, w, h)	Малює закрашений еліпс. Параметр **Brush** визначає колір і стиль зафарбовування внутрішньої області еліпса; параметри **x, y, w, h** - координати лівого верхнього кута і розмір прямокутника, усередині якого викреслюється еліпс
DrawPolygon(Pen, P)	Малює контур багатокутника. Параметр **Pen** визначає колір, товщину і стиль лінії межі багатокутника; параметр **P** (масив типу **Point**) - координати кутів багатокутника
FillPolygon(Brush, P)	Малює закрашений багатокутник. Параметр **Brush** визначає колір і стиль зафарбовування внутрішньої області багатокутника; параметр **P** (масив типу **Point**) - координати кутів багатокутника
DrawString(str, Font, Brush,x,y)	Виводить на графічну поверхню рядок тексту. Параметр **Font** визначає шрифт; **Brush** - колір символів; **x** і **y** - точку, від якої буде виведений текст
DrawImage(Image, x, y)	Виводить на графічну поверхню ілюстрацію. Параметр **Image** визначає ілюстрацію; **x** і **y** - координату лівого верхнього кута області виведення ілюстрації

Один і той же елемент можна намалювати за допомогою різних, але методів, що мають однакові імена (пригадайте: можливість оголошення функцій, що мають однакові імена, але різні параметри, називається перевантаженням).

Наприклад, прямокутник можна намалювати методом **DrawRectangle**, якому як параметри передаються координати лівого верхнього кута і розміри прямокутника:

e->Graphics->DrawRectangle(Pens::Black, x, x, w, h)

Це ж завдання може вирішити метод **DrawRectangle**, якому як параметр передається структура типу **Rectangle**, поля якої визначають прямокутник (*положення і розмір*):

Rectangle aRect = Rectangle(20,100,50,50);

e->Graphics->DrawRectangle(Pens::Blue, aRect);

Існування декількох методів, що виконують одне і те ж завдання, дозволяє програмісту вибрати метод, найбільш відповідний для вирішення конкретного завдання.

Як параметри методів викреслювання графічних примітивів часто використовується структура **Point**. Її поля **X** і **Y** визначають положення (координати) точки графічної поверхні.

Наприклад:

Point p1 = Point(10,10);

Point p2 = Point(100,10);

e->Graphics->DrawLine(Pens::Green, p1, p2); *// малюємо лінію з p1 у p2*

Лінія

Метод **DrawLine** малює пряму лінію. У інструкції виклику методу слід вказати олівець, яким треба намалювати лінію, і координати точок початка і кінця лінії:

DrawLine(aPen x1, y1, x2, y2)

Параметр **aPen** задає олівець, який малює лінію: x1 і y1 - точка початку лінії, а x2 і y2 - точка кінця лінії. Параметри x1, y1, x2 і y2 повинні бути одного типа (Integer або Single).

Наприклад

e->Graphics->DrawLine(Pens::Green,10,10,300,10);

Дана інструкція малює зелену лінію товщиною в один пиксел з точки (10, 10) у точку з координатами (300, 10).

Ламана лінія

Метод **DrawLines** малює ламану лінію. Як параметри методу передається олівець (**Pen**) і масив типу **Point**, елементи якого містять координати вузлових точок

лінії. Метод малює ламану лінію, послідовно сполучаючи крапки, координати яких знаходяться в масиві: першу з другою, другу з третьою, третю з четвертою і т.д.

Приклад фрагменту коду побудови ламаної лінії, що складається з чотирьох ланок.

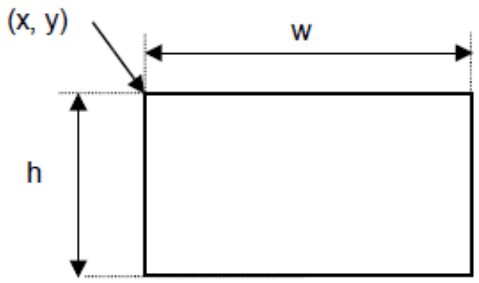
```
Array <Point>^ p;    // масив крапок
p = gcnew array<Point>(5);
p[0].X = 10; p[0].Y =50;
p[1].X = 20; p[1].Y =20;
p[2].X = 30; p[2].Y =50;
p[3].X = 40; p[3].Y =20;
p[4].X = 50; p[4].Y =50;
e->Graphics->DrawLines(Pens::Green,p);
```

Метод **DrawLines** можна використовувати для побудови замкнутих контурів. Для цього перший і останній елементи масиву повинні містити координати однієї і тієї ж точки.

Прямокутник

Метод **DrawRectangle** креслить прямокутник . Як параметри методу треба вказати олівець, координати лівого верхнього кута і розмір прямокутника:

DrawRectangle(aPen, x, y, w, h);



DrawRectangle(aPen, x, y, w, h)

Параметри **X** і **Y** задають координати лівого верхнього кута прямокутника, а **W** і **H** - розмір (ширину і висоту).

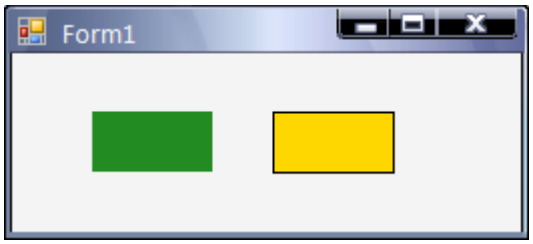
Вид лінії межі прямокутника (*колір, стиль і ширину*) визначає параметр **aPen**, як який можна використовувати один із стандартних олівців або олівець, створений програмістом.

Метод **FillRectangle** малює зафарбований прямокутник. Як параметри методу треба передати пензель, координати лівого верхнього кута і розмір прямокутника:

FillRectangle(aBrush, x, y, w, h);

Параметр **aBrush**, як який можна використовувати стандартну або створену програмістом штриховий (**HatchBrush**), градієнтний (**LineadGradientBrush**) або текстурний (**TextureBrush**) пензель, визначає колір і стиль зафарбовування області.

*Приклад програми (процедури обробки події **Paint**), яка демонструє використання методів **DrawRectangle** і **FillRectangle**.*




```
private: System::Void pictureBox1_Paint(…)
{
   // Зелений прямокутник розміром 60x30 лівий верхній кут якого в точці (10, 10)
   e->Graphics->FillRectangle(Brushes::ForestGreen, 10,10,60,30);

   // Жовтий прямокутник з чорим контуром розміром 60x30 лівий верхній кут
   якого в точці (100, 10)
   e->Graphics->FillRectangle(Brushes::Gold, 100,10,60,30); // прямокутник
   e->Graphics->DrawRectangle(Pens::Black, 100,10,60,30); // контур
}
```

Точка

У об'єкту **Graphics** немає методу, який дозволяє намалювати точку а поверхні. Метод **SetPixel** є у об'єкту **Bitmap**. Тому якщо необхідно сформувати картинку з крапок, доведеться створити об'єкт **Bitmap**, сформувати на його поверхні зображення, а потім це зображення за допомогою методу **DrawImage** вивести на графічну поверхню. Але можна поступити простіше - замість крапки вивести квадрат розміром в один піксель.

Наприклад

e->Graphics->FillRectangle(Brushes::Red,x,y,1,1);

Дана інструкція малює на графічній поверхні червону крапку.

Багатокутник

Метод **DrawPolygon** креслить багатокутник (контур). Інструкція виклику методу в загальному вигляді виглядає так:

DrawPolygon(aPen, p)

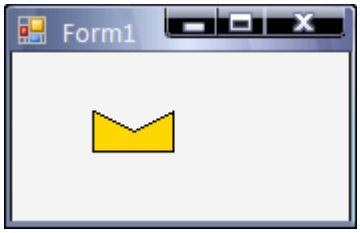
Параметр **p** - масив типу **Point**, визначає координати вершин багатокутника. Метод **DrawPolygon** креслить багатокутник, сполучаючи прямими лініями крапки, координати яких знаходяться в масиві: першу з другою, другу з третьою і т.д. Остання крапка з'єднується з першою. Вид межі багатокутника визначає параметр **aPen**, як який можна використовувати стандартний або створений програмістом олівець.

Зафарбований багатокутник малює метод **FillPolygon**. Інструкція виклику методу в загальному вигляді виглядає так:

FillPolygon(aBrush, p)

Параметр **aBrush**, як який можна використовувати стандартну або створену програмістом штрихову (**HatchBrush**), градієнтну (**LineadGradientBrush**) або текстуровану (**TextureBrush**) кисть, визначає колір і стиль зафарбовування внутрішньої області багатокутника.

Приклад фрагменту коду, який демонструє використання методів ***DrawPolygon*** *і* ***FillPolygon*** *- малює зафарбований жовтий багатокутник з чорним контуром*.



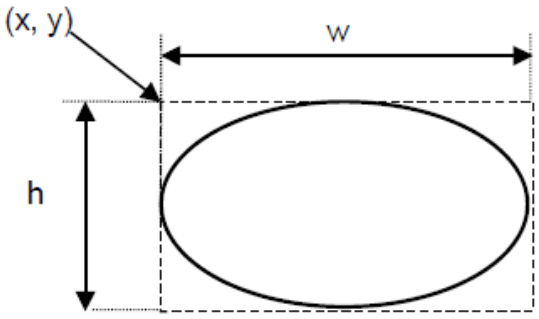

```
array<Point>^ p;
p = gcnew array<Point>(5);
p[0].X = 10; p[0].Y =30;
p[1].X = 10; p[1].Y =10;
p[2].X = 30; p[2].Y =20;
p[3].X = 50; p[3].Y =10;
p[4].X = 50; p[4].Y =30;
e->Graphics->FillPolygon(Brushes::Gold, p);
e->Graphics->DrawPolygon(Pens::Black,p);
```

Еліпс і коло

Метод **DrawEllipse** креслить еліпс усередині прямокутної області. Якщо прямокутник є квадратом, то метод малює коло.

Інструкція виклику методу **DrawEllipse** в загальному вигляді виглядає так:

DrawEllipse (aPen, x, y, w, h);



DrawEllipse(aPen, x, y, w, h)

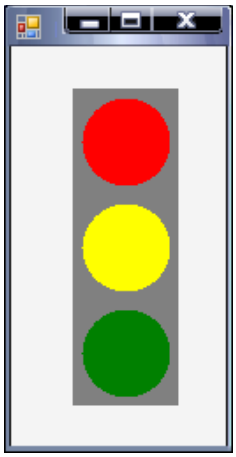
Параметр **aPen**, як який можна використовувати один із стандартних олівців або олівець, створений програмістом, визначає вид (колір, товщину, стиль) межі еліпса. Параметри **`x, y, w`** і **`h`** задають координати лівого верхнього кута і розмір прямокутника, усередині якого метод малює еліпс.

Метод **FillEllipse** малює зафарбований еліпс. У інструкції виклику методу слід вказати пензель (стандартну або створену програмістом), координати і розмір прямокутника, усередині якого треба намалювати еліпс:

FillEllipse(aBrush, x, y, w, h);

Пензель визначає колір і спосіб зафарбовування внутрішньої області еліпса.

Приклад програми побудови графічного забраження «Світлофор»



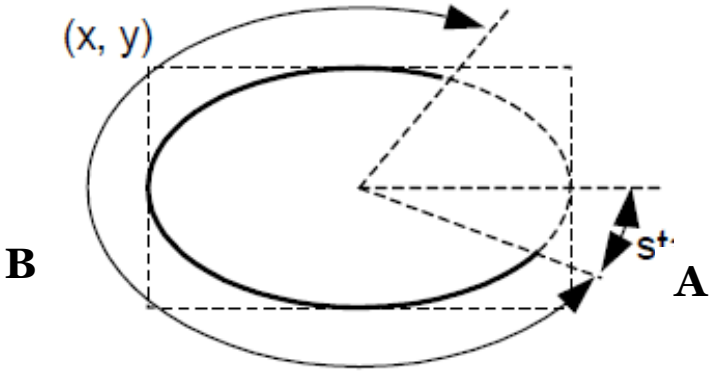
```
private: System::Void pictureBox1_Paint(…)
{
e->Graphics->FillRectangle(Brushes::Gray,5,5,60,180);
e->Graphics->FillEllipse(Brushes::Red, 10,10,50,50);
e->Graphics->FillEllipse(Brushes::Yellow, 10,70,50,50);
e->Graphics->FillEllipse(Brushes::Green, 10,130,50,50);
}
```

Дуга

Метод **DrawArc** малює дугу - частина еліпса. Інструкція виклику методу в загальному вигляді виглядає так:

DrawArc(aPen, x, y, w, h, A, B)

Параметри **x, y**, **w** і **h** визначають еліпс (коло), частиною якого є дуга. Параметр **B** задає початкову точку дуги - перетин еліпса і прямої, проведеної з центру еліпса і створюючої кут **A** з горизонтальною віссю еліпса (кутова координата зростає за годинниковою стрілкою). Параметр **B** задає довжину дуги (у градусах). Якщо значення **B** позитивне, то дуга малюється від початкової точки за годинниковою стрілкою, якщо від`ємне - то проти. Величини кутів задаються в градусах.



У інструкції виклику методу **DrawArc** замість параметрів **x, y, w** і **h** можна вказати структуру типу **Rectangle**:

DrawArc(aPen, aRect, A, B)

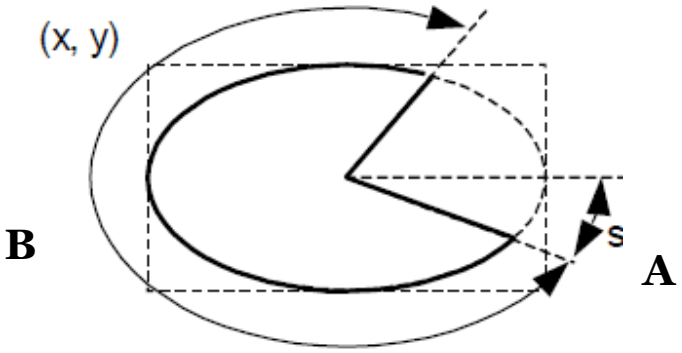
Сектор

Метод **DrawPie** малює межу сектора. Інструкція виклику методу:

DrawPie(aPen, x, y, w, h, A, B);

Параметри **x, y, w** і **h** визначають еліпс, частиною якого є сектор. Параметр **A** задає початкову точку дуги сектора - перетин еліпса і прямої, проведеної з центру еліпса і створюючої кут **A** з горизонтальною віссю еліпса (кутова координата зростає за годинниковою стрілкою). Параметр **B** - довжину дуги сектора (у градусах). Якщо значення **B** позитивне, то дуга сектора малюється від

початкової точки за годинниковою стрілкою, якщо негативне - проти. Величини кутів задаються в градусах.



У інструкції виклику методу **DrawPie** замість параметрів **`x`, `y`, `w`** і **`h`** можна вказати структуру типу **Rectangle**:

```
DrawPie(aPen, aRect, A, B)
```

Метод **FillPie** малює сектор. Параметри у методу **FillPie**, за винятком першого, замість якого треба вказати кисть, такі ж, як і у методу **DrawPie**.

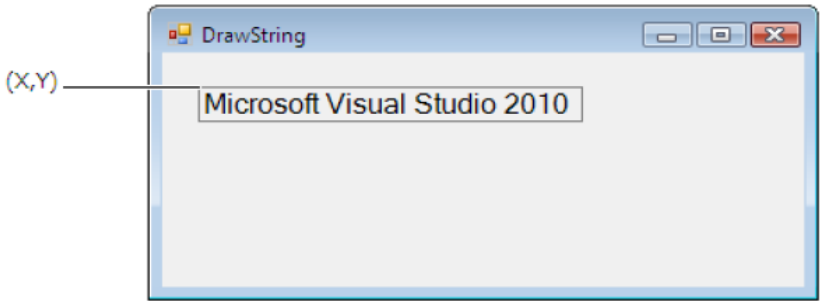
Текст

Виведення тексту на графічну поверхню виконує метод **DrawString**. У інструкції виклику методу указується рядок, шрифт, кисть і координати точки, від якої треба вивести текст:

```
DrawString(st, aFont, aBrush, x, y);
```

Параметр **`st`** задає текст, параметр **`aFont`** - шрифт, який використовується для відображення тексту, а **`aBrush`** - колір тексту. Параметри **`x`** і **`y`** визначають координати лівого верхнього кута області відображення тексту.

Приклад програми виведеня текстового повідомлення. У наведеному прикладі для виведення тексту використовується шрифт форми, заданий властивістю **Font**.



```
private: System::Void Form1_Paint(…)
{
String^ st1 = "Microsoft Visual Studio 2010";
e->Graphics->DrawString(st1,this->Font, Brushes::Black, 20,20);
}
```

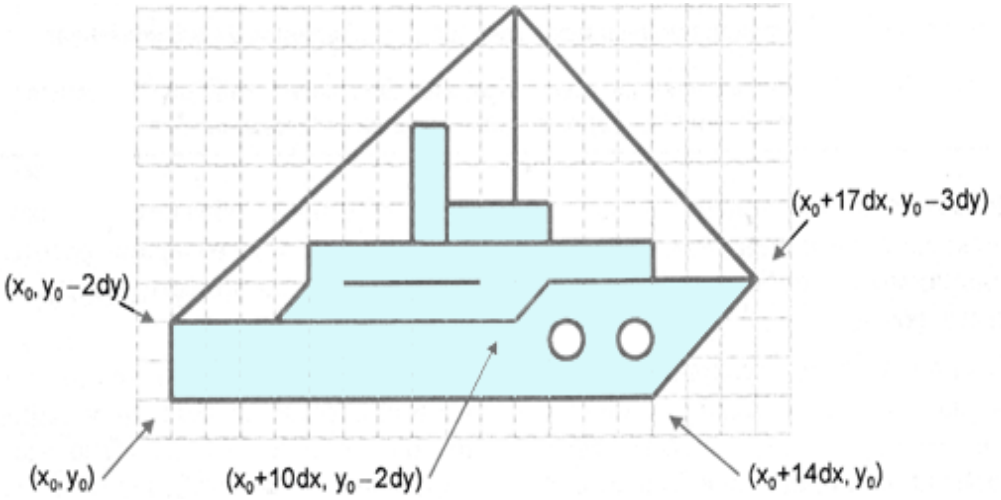
Метод базової точки

При програмуванні складних зображень, що складаються з безлічі елементів, використовується метод, який називається **методом базової точки**. Суть цього методу полягає в наступному:

1. Вибирається деяка точка зображення, яка береться за базову.

2. Координати інших точок відлічуються від базової точки.

3. Якщо координати точок зображення відлічувати від базової у відносних одиницях, а не у пікселах, то забезпечується можливість масштабування зображення.



Анімація

Під мультиплікацією звичайно розуміється рухомий малюнок. Забезпечити переміщення малюнка досить просто: треба спочатку вивести малюнок на екран, потім через деякий час стерти його і знову вивести цей же малюнок, але вже на деякій відстані від його первинного положення. Підбором часу між виведенням і видаленням малюнка, а також відстані між старим і новим положенням малюнка (крок переміщення), можна досягти того, що у спостерігача складатиметься враження, що малюнок рівномірно рухається по екрану.

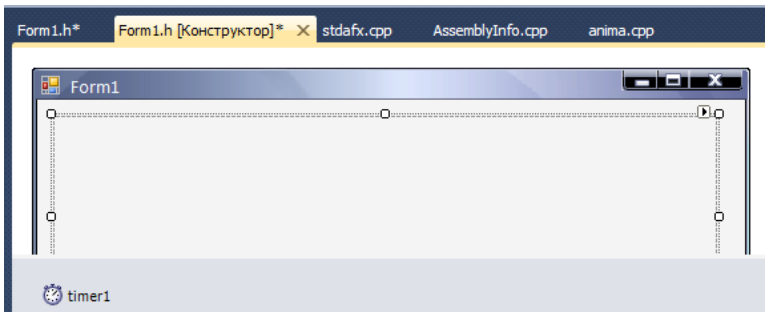
Компонент **Timer**, який використовується для генерації послідовності подій, функція обробки яких забезпечує виведення і видалення малюнка. Компонент **Timer** є невізуальним. Це означає, що під час роботи програми компонент в діалоговому вікні не відображається. Тому компонент **Timer** можна помістити в будь-яку точку форми. Властивості компоненту **Timer** приведені у табл. 17.5.

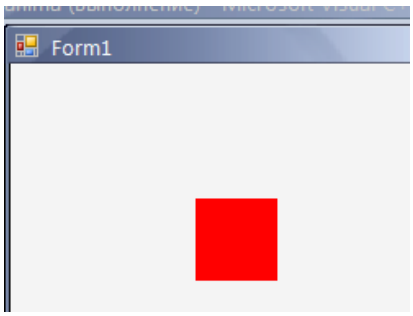
Компонент **Timer** генерує подію **Tick**. Період виникнення події **Tick** вимірюється в мілісекундах і визначається значенням властивості **Interval**. Слід звернути увагу на властивість **Enabled**. Воно дає можливість програмі "запустити" або "зупинити" таймер. Якщо значення властивості **Enabled** = **False**, та подія **Tick** не виникає.

*Таблиця 17.5. Властивості компоненту **Timer***

Властивості	Опис
Interval	Період генерації події **Tick**. Задається в мілісекундах
Enabled	Дозвіл роботи. Дозволяє (значення **True**) або забороняє (значення **False**) генерацію події **Tick**

Приклад програми, яка імітує рух геометричної фігури (червоий квадрат)






```
namespace anima {
…
using namespace System::Drawing;
int x=50;  // оголошення глобальної змінної
…
private: System::Void pictureBox1_Paint (…)
{  e->Graphics->FillRectangle(Brushes::Red,x,70,50,50); }

private: System::Void timer1_Tick(…)
{
  x+=3;
  pictureBox1->Refresh();
}
```

ЛАБОРАТОРНІ РОБОТИ

Лабораторна робота №1.
Ознайомлення з основами мови С++.
Компіляція та запуск програми на виконання.

Теоретичні відомості

Програма мовою С++ може містити наступні символи:

- ✓ Літери латинської абетки A,B,C…x, y,z.
- ✓ Арабські цифри 0,1,2,...9
- ✓ Спеціальні символи +, -, /, =,<>, [], (), {}, **:**, **;**, **.**, **$**, **#** , **_**, @

В якості імен програм, типів, констант, змінних, модулів та інших об'єктів мови використовуються ідентифікатори. **Ідентифікатор** – ім'я, яке вільно обирається програмістом для елементів програми. Компілятор мови С++ розглядає літери верхнього та нижнього регістрів як різні символи.

Арифметичні операції:

+	додавання
-	віднімання
*	множення
/	ділення
%	залишок від ділення

Унарні арифметичні операції:

+	унарний плюс
-	унарний мінус
+ +	операція інкременту (збільшення на 1)
- -	операція декременту (зменшення на 1)

Операція присвоєння: „=”

Операція слідування: „ , ”.

Умовна операція : (умова)? вираз_1: вираз_2;

Оператори введення та виведення. Існують стандартні потоки для введення інформації з клавіатури та виведення на екран.

В С++ стандартний поток введення пов'язаний з константою **cin** та виведення **cout.** Для використання цих констант підключається заготовочний файл **iostream.h.**

Для підключення математичних засобів використовують файл **math.h**

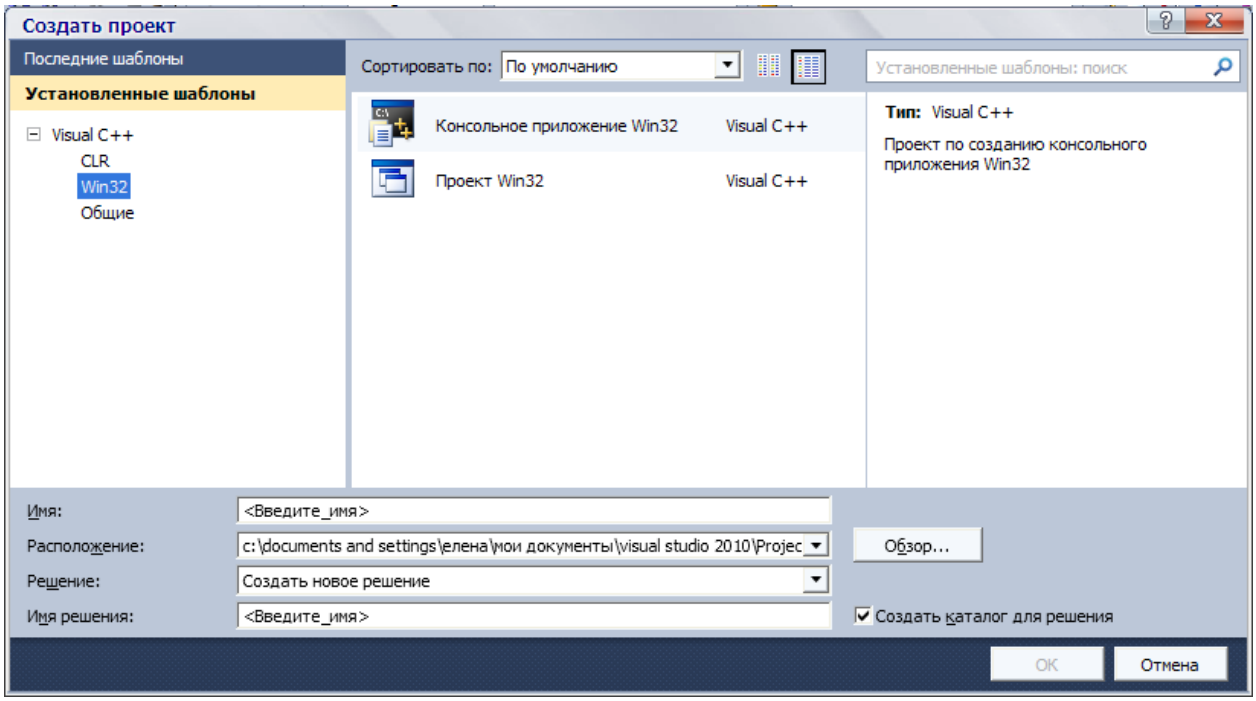
Для створення програм, які не є Windows-додатками та їх роботи необхідний текстовий режим для введення та виведення інформації.

Консольний режим роботи середовища Visual C++2010 Express. Програма, що створюється в середовищі Visual C++, завжди оформляється у вигляді окремого проекту. **Проект** (project) - це набір взаємозв'язаних початкових файлів, призначених для вирішення певного завдання, компіляція і компоновка яких дозволяє одержати програму. У проект входять як файли, що безпосередньо створюються програмістом, так і файли, які автоматично створює і редагує середовище програмування.

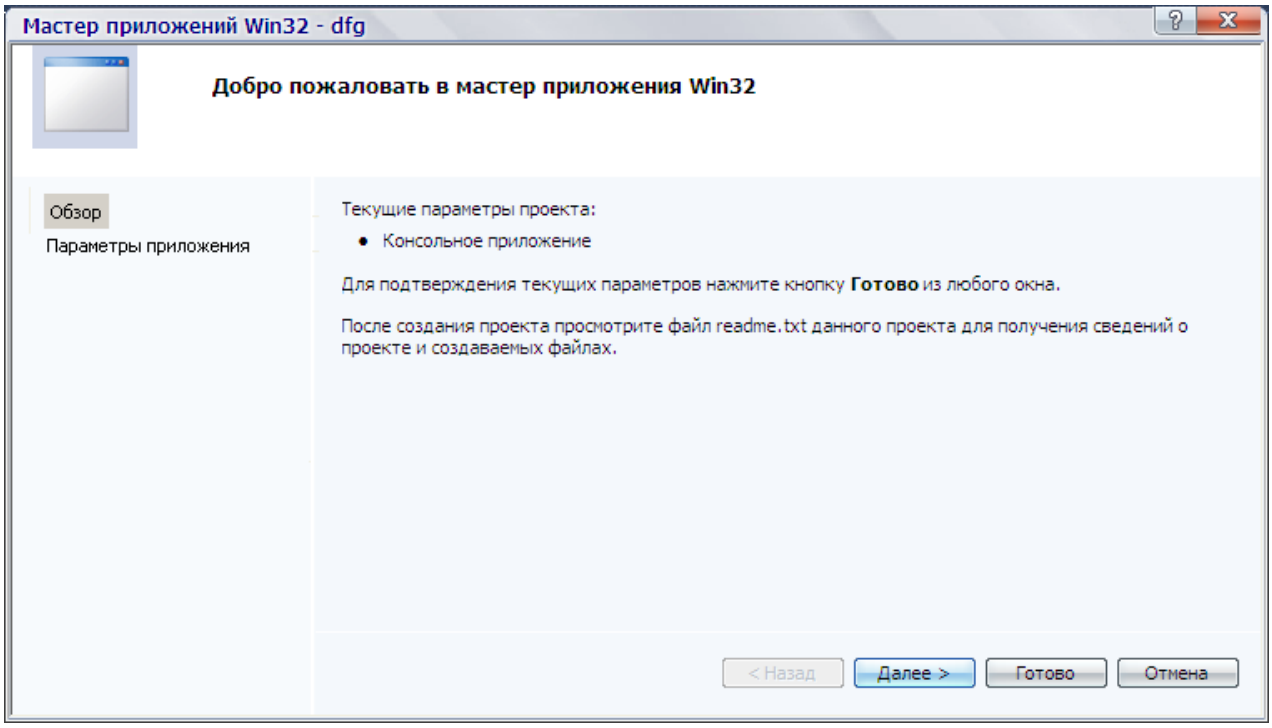
Для створення нового проекту необхідно:

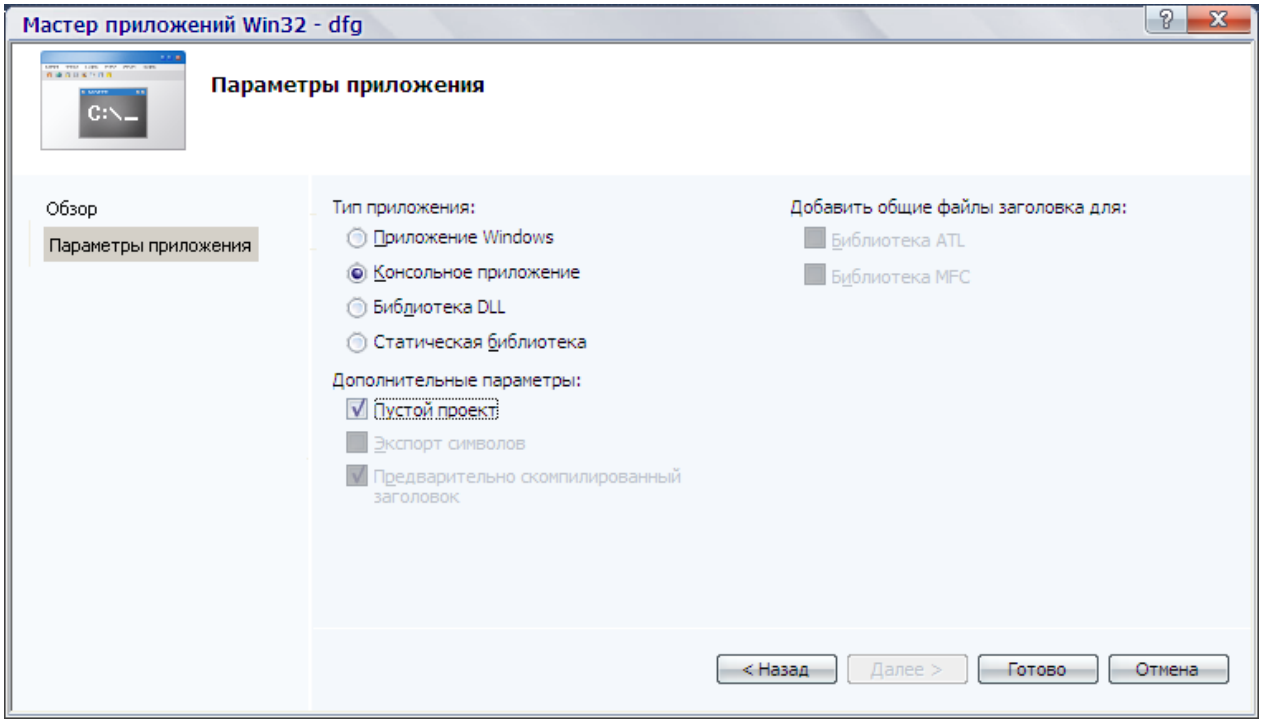
• вибрати «**Создать проект**» або команду головного меню «**Файл-Создать-Проект**»;

• У вузлі Visual C++ області встановлених шаблонів необхідно вибрати **Win32**, а потім у області шаблонів (у середній колонці) обрати шаблон «**Консольное приложение Win32 (Visual C++)**». Після чого необхідно ввести ім'я проекту в поле «**Имя**» і шлях до файлів проекту.



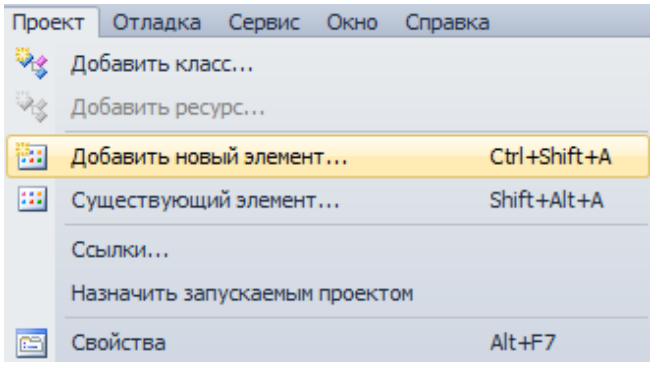
• у вікні майстра додатку, що відкрилося вибрати додаткові параметри «**Пустой проект**» і клацнути по кнопці «**Готово**»;

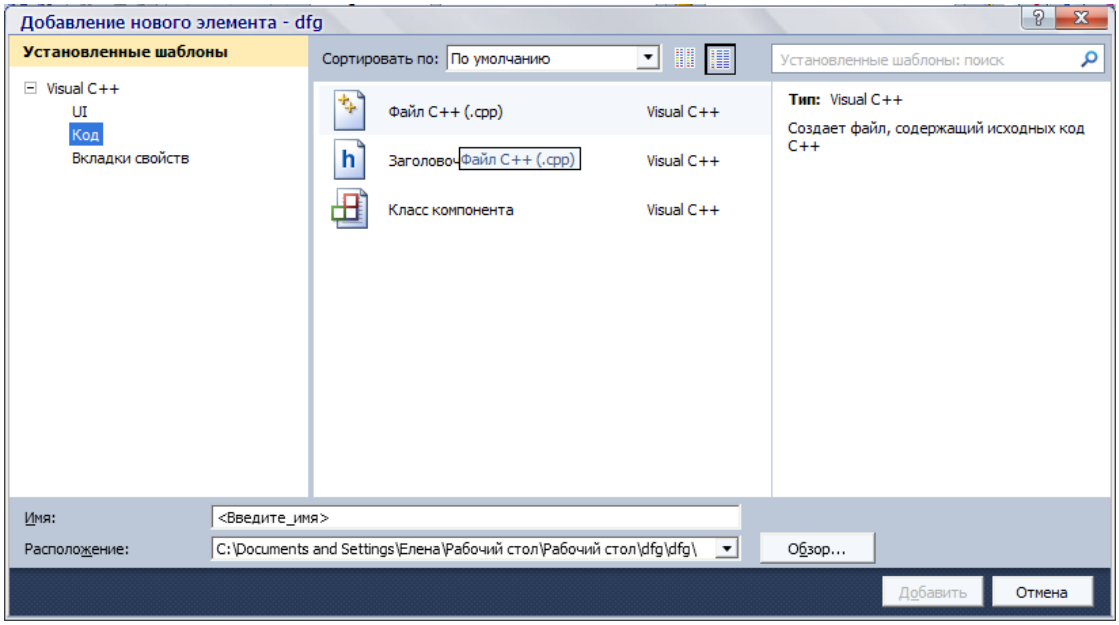


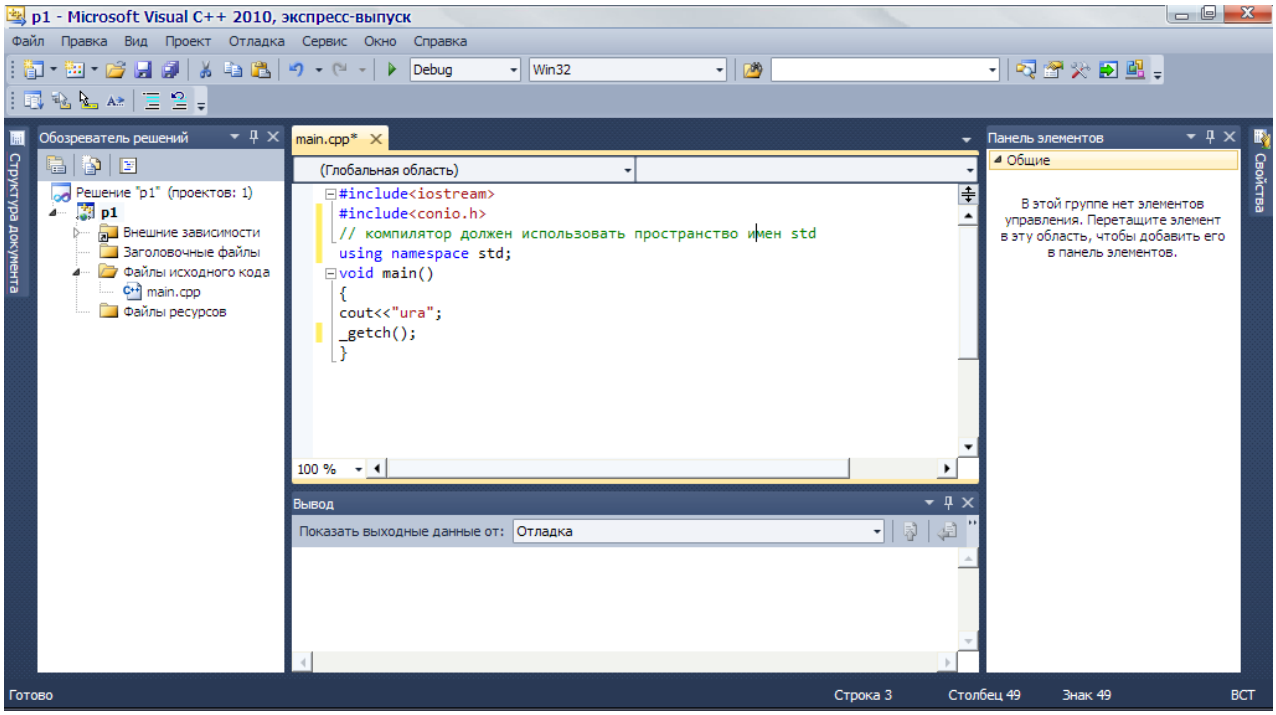


Для роботи з консольним додатком необхідно створити новий або додати існуючий файл з текстом програми.

Для **створення нового файлу** необхідно:







Запуск проекту здійснюється командою «**Отладка-Начать отладку**» або клавіша **F5** або кнопка із стрілкою на панелі інструментів.

Функції бібліотеки math.lib

Функції для розрахунку математичних виразів знаходяться в бібліотеці math.lib (підключення бібліотеки: **#include<math.h>.** Всі аргументи в тригонометричних функціях задаються в радіанах. Параметри і аргументи всієї решти функцій мають тип **double** (окрім **abs(x)**).

Математична функція	**Функція бібліотеки**	**Опис**
|x|	abs(x)	Обчислення абсолютного значення (тільки для цілих чисел!)
arccos(x)	acos(x)	Обчислення значення арккосинуса x
arctg(x)	atan(x)	Обчислення значення арктангенса x
arctg(x/y)	atan2(x,y)	Обчислення значення арктангенса двох аргументів x і y
Округлення до більшого	ceil(x)	Функція повертає дійсне значення, відповідне найменшому цілому числу, яке більше або рівне x
cos(x)	cos(x)	Обчислення косинуса x
ch(x)= $(e^{x}+e^{-x})/2$	cosh(x)	Обчислення косинуса гіперболічного x
e^{x}	exp(x)	Обчислення експоненти числа x
$\|x\|$	fabs(x)	Обчислення абсолютного значення x
Округлення до меншого	floor(x)	Функція повертає дійсне значення, відповідне найбільшому цілому числу, яке менше або рівне x
Залишок від ділення x на y	fmod(x,y)	Функція повертає дійсне значення, відповідне залишку від цілочисельного ділення x на y
ln(x)	log(x)	Обчислення натурального логарифма x
$lg_{10}(x)$	log10(x)	Обчислення десяткового логарифма x
x^{y}	pow(x, y)	Зведення x в ступінь y
sin(x)	sin(x)	Обчислення синуса x
sh(x)= $(e^{x}-e^{-x})/2$	sinh(x)	Обчислення синуса гіперболічного x
$\sqrt{x}$	sqrt(x)	Обчислення квадратного кореня x
tg(x)	tan(x)	Обчислення тангенса x
tgh(x)	tanh(x)	Обчислення тангенса гіперболічного x

Завдання для самостійного виконання

Завдання 1. Створити проект, наберіть текст програми, відкомпілюйте та запустіть її на виконання. Напишіть коментарі до кожного рядка програми.

Приклад програми обчислення суми двох чисел

```
#include<iostream>
#include<conio.h>
using namespace std; // компілятор повинен використовувати простір імен std
void main()
{
        setlocale(LC_ALL, "Russian"); // забезпечує коректне виведення кирилиці
        int a,b,s;
        cout <<"\n Введіть 2 числа \n";   //  \n- перехід на наступний рядок
        cin >> a>>b;
        s=a+b;
        cout << "\n  сума=" <<s;
        _getch();       // затримка екрану (при підключенні файлу conio.h)
}
```

Завдання 2.

1. Напишіть програму, яка буде запитувати рік вашого народження та виводитиме ваш вік.

2. Задано два дійсних числа визначте максимальне з них використовуючи умовну операцію.

3. Написати програму, яка визначає з двох дійсних чисел додатне (*умовна операція*) та збільшує його значення на одиницю (*операція інкременту*).

4. Напишіть програму переведення метрів в сантиметри.

Завдання 3 (*відповідно до номеру в списку*). Обчислити значення виразу при заданих початкових даних. Порівняти набутого значення з вказаним правильним результатом.

1. $s = \dfrac{2\cos\left(x - \dfrac{2}{3}\right)}{\dfrac{1}{2} + \sin^2 y}\left(1 + \dfrac{z^2}{3 - z^2/5}\right).$

При x = 14.26; y = – 1.22; z = 3.5×10^{-2} відповідь s = 0.749155.

2. $s=\dfrac{\sqrt[3]{9+(x-y)^2}}{x^2+y^2+2}-e^{|x-y|}\mathrm{tg}^3 z.$

При x = –4.5; y = 0.75×10^{-4}; z = -0.845×10^{2} відповідь s = –3.23765.

3. $s=\dfrac{1+\sin^2(x+y)}{\left|x-\dfrac{2y}{1+x^2y^2}\right|}x^{|y|}+\cos^2\left(\mathrm{arctg}\dfrac{1}{z}\right).$

При x = 3.74×10^{-2}; y=–0.825; z = 0.16×10^{2} відповідь s = 1.05534.

4. $s=\left|\cos x-\cos y\right|^{\left(1+2\sin^2 y\right)}\left(1+z+\dfrac{z^2}{2}+\dfrac{z^3}{3}+\dfrac{z^4}{4}\right).$

При x = 0.4×10^{4}; y = –0.875; z = -0.475×10^{-3} відповідь s = 1.98727.

5. $s=\ln\left(y^{-\sqrt{|x|}}\right)\left(x-\dfrac{y}{2}\right)+\sin^2(\mathrm{arctg}(z)).$

При x = –15.246; y = 4.642×10^{-2}; z = 21 відповідь s = –182.038.

6. $s=\sqrt{10\left(\sqrt[3]{x}+x^{y+2}\right)}\left(\arcsin^2 z-|x-y|\right).$

При x = 16.55×10^{-3}; y = –2.75; z = 0.15 відповідь s = –40.6307.

7. $s=5\mathrm{arctg}(x)-\dfrac{1}{4}\arccos(x)\dfrac{x+3|x-y|+x^2}{|x-y|z+x^2}.$

При x = 0.1722; y = 6.33; z = 3.25×10^{-4} відповідь s = –205.306.

8. $s=\dfrac{e^{|x-y|}|x-y|^{x+y}}{\mathrm{arctg}(x)+\mathrm{arctg}(z)}+\sqrt[3]{x^6+\ln^2 y}.$

При x = -2.235×10^{-2}; y = 2.23; z = 15.221 відповідь s = 39.3741.

9. $s=\left|x^{\frac{y}{x}}-\sqrt[3]{\dfrac{y}{x}}\right|+(y-x)\dfrac{\cos y-\dfrac{z}{(y-x)}}{1+(y-x)^2}.$

При x = 1.825×10^{2}; y = 18.225; z = -3.298×10^{-2} відповідь s = 1.21308.

10. $s=2^{-x}\sqrt{x+\sqrt[4]{|y|}}\sqrt[3]{e^{x-1/\sin z}}.$

При x = 3.981×10^{-2}; y = -1.625×10^{3}; z = 0.512 відповідь s = 1.26185.

11. $s=y^{\sqrt[3]{|x|}}+\cos^3(y)\dfrac{|x-y|\left(1+\dfrac{\sin^2 z}{\sqrt{x+y}}\right)}{e^{|x-y|}+\dfrac{x}{2}}.$

При x = 6.251; y = 0.827; z = 25.001 відповідь s = 0.712122.

12. $s = 2^{\left(y^x\right)} + \left(3^x\right)^y - \dfrac{y\left(\text{arctgz} - \dfrac{1}{3}\right)}{|x| + \dfrac{1}{y^2+1}}$.

При x = 3.251; y = 0.325; z = 0.466×10^{-4} відповідь s = 4.23655.

13. $s = \dfrac{\sqrt[4]{y + \sqrt[3]{x-1}}}{|x-y|\left(\sin^2 z + \text{tgz}\right)}$.

При x = 17.421; y = 10.365×10^{-3}; z = 0.828×10^{5} відповідь s = 0.330564.

14. $s = \dfrac{y^{x+1}}{\sqrt[3]{|y-2|}+3} + \dfrac{x + \dfrac{y}{2}}{2|x+y|}(x+1)^{-1/\sin z}$.

При x = 12.3×10^{-1}; y = 15.4; z = 0.252×10^{3} відповідь s = 82.8256.

15. $s = \dfrac{x^{y+1} + e^{y-1}}{1 + x|y - \text{tgz}|}\left(1 + |y-x|\right) + \dfrac{|y-x|^2}{2} - \dfrac{|y-x|^3}{3}$.

При x = 2.444; y = 0.869×10^{-2}; z = -0.13×10^{3} відповідь s = –0.498707.

Додаткові завдання

1. Обчислити значення функції **$y=7x^2-3x+6$** при будь яких значеннях **x**.
2. Дано сторони прямокутника. Визначити його периметр та довжину діагоналі.
3. З початку доби пройшло **n** секунд. Визначити скільки пройшло годин.
4. У трьохзначному числі невідома перша цифра. Коли отримане число помножили на 10, а добуток додали до першої цифри то отримали число n. За заданим **n** знайти число **x** (**n** вводиться з клавіатури, 1<= **n**<=999).

Контрольні запитання

1. Які символи входять до складу алфавіту мови С++?
2. Як записується коментар до тексту програми у мові С++?
3. Що таке тип даних? Які типи даних виділяють у мові С++?
4. Які операції можна виконати над даними в мові С++?
5. Які основні оператори використовуються для введення та виведення даних?

Лабораторна робота №2.
Програмування розгалужень у мові С++

Теоретичні відомості

Мова С++ володіє набором конструкцій, які дозволяють керувати порядком виконання окремо взятих гілок програми в залежності від результату перевірки певної умови.

Для реалізації розгалуження може бути використано УМОВНИЙ ОПЕРАТОР.

1) Оператор **IF.**

Реалізує розгалуження програми в залежності від перевірки умови на істинність.

if (<*умова*>) <*оператор*>;

У конструкціях мови С++ оператори можуть бути блочними. Блок починається з відкриваючої фігурної дужки „{” та закінчується „}”. Весь вміст блоку розглядається компілятором як єдиний оператор.

2) Оператор **IF-ELSE.**

Реалізує розгалуження програми в залежності від перевірки умови. Якщо умова набуває значення „істина”, то виконується оператор_1, якщо умова „Хибна” – оператор_2.

if (<*умова*>) <*оператор_1*>; else <*оператор_2*>;

Якщо умова складається з декількох частин то у мові С++ застосовуються

Операції порівняння:

>	Більше
<	Менше
>=	більше або дорівнює
<=	менше або дорівнює
= =	Рівно
!=	не рівно

Логічні операції:

&&	логічне І
||	логічне АБО
!	логічне заперечення

Приклад програми визначення максимального із 3-х чисел

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
   int a, b, c, max;
   cout<<"\n  Введіть 3 числа \n";
   cin>>a>>b>>c;
   if(a>b&&a>c) max=a;
   else
   if(b>a&&b>c) max=b;
   else max=c;
   cout<<"max= "<<max;
   _getch();
}
```

3) Оператор вибору **SWITCH**.

Оператор вибору застосовується коли необхідно здійснити вибір одного варіанту із багатьох.

switch (<*вираз*>**)**
{
case <*константне значення 1*>**:** <*оператор 1*>**; break;**
case < *константне значення вираз 2*>**:** <*оператор 2*>**; break;**
default>: <*оператор n*>**; break;**
}

Приклад програми визначення назви сезону за номером (1-4)

```
void main()
{
   int n;
   cout<<"\n  Введіть N сезону -> ";
   cin>>n;
   switch(n)
   {
      case 1:cout<<"зима"; break;
      case 2:cout<<"весна"; break;
      case 3:cout<<"літо"; break;
      case 4:cout<<"осінь"; break;
      default:cout<<"помилка";break;
   }
_getch();
}
```

Приклад програми яка за введеним символом математичної операції здійснює відповідну дію над двома числами

```
void main()
{
    char c;
    double a,b;
    cout<<"\n Введіть 2 числа \n";
    cin>>a>>b;
    cout<<"\n Введіть символ операції -> ";
    cin>>c;
    switch(c)
    {
        case '+':cout<<"a+b="<<a+b; break;
        case '-':cout<<"a-b="<<a-b; break;
        case '*':cout<<"a*b="<<a*b; break;
        case '/':cout<<"a/b="<<a/b; break;
        default:cout<<" помилка ";break;
     }
     _getch();
}
```

Завдання для самостійного виконання

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Завд. **1**	1	1	1	1	1	1	1	1	1	1
Завд. **2**	непарні	парні	непарні	парні	непарні	парні	непарні	парні	непарні	парні
Завд. **4**	2, 4	1, 3	4, 2	1, 3	2,4	1, 3	2, 4	1, 3	2, 4	1, 3

Завдання 1 Написати програму обчислення коренів квадратного рівняння.

Завдання 2

1. Маємо три дійсних числа x, y, z. Знайдіть найменше з них.

2. Маємо дійсні додатні числа x, y, z. Складіть програму, яка буде перевіряти існування трикутника з довжинами сторін x, y, z. Визначити периметр трикутника.

3. Серед значень функції $y = \sin x$ у трьох заданих точках a, b, c визначити кількість різних.

4. Складіть програму, яка перевірятиме введене з клавіатури число на парність.

5. Перевірити чи є введене з клавіатури число додатнім, від'ємним, нулем.

6. Визначити, у якій з трьох даних точок a, b, c функція y=cos(x) набуває найбільшого значення.

7. Дано три різних дійсних числа a, b, c. Знайти добуток двох менших чисел.

8. Дано три різних дійсних числа a, b, c. Замінити найменше число добутком двох більших.

Завдання 3 Скласти програму для знаходження значення функції (*номер завдання відповідає номеру у списку***):**

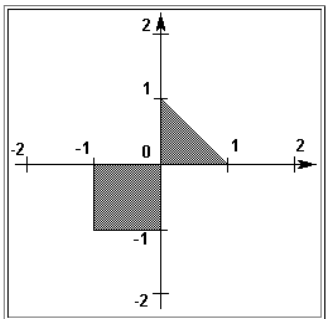
1) $f(x)=\begin{cases} x^2, \text{ при } -2{,}4 \le x \le 5.7, \\ 4 \quad \text{в іншому випадку} \end{cases}$	2) $f(x)=\begin{cases} 4x^2+2x-19, \text{ якщо } x \ge -3.5 \\ -\dfrac{2x}{-4x+1}, \text{ якщо } x<3.5 \end{cases}$
3) $f(x)=\begin{cases} \sin x \text{ при } -2.4 \le x \le 0.9, \\ 1 \quad \text{в іншому випадку} \end{cases}$	4) $f(x)=\begin{cases} -x^2+3x+9, \text{ якщо } x \le 3 \\ \dfrac{x}{x^2+1}, \text{ якщо } x>3 \end{cases}$
5) $f(x)=\begin{cases} x^2-3x+9, \text{ якщо } x \le 3, \\ \dfrac{1}{x^3+6}, \quad \text{якщо } x>3 \end{cases}$	6) $f(x)=\begin{cases} 45x^2+5, \text{ якщо } x>3.6 \\ \dfrac{5x}{10x^2+1}, \text{ якщо } x \le 3.6 \end{cases}$
7) $f(x)=\begin{cases} 0, \text{ якщо } x \le 1, \\ \dfrac{1}{x+6}, \text{ якщо } x>1 \end{cases}$	8) $f(x)=\begin{cases} x^4+9, \text{ якщо } x<3.2 \\ \dfrac{54x^4}{-5x^2+7}, \text{ якщо } x \ge 3.2 \end{cases}$
9) $f(x)=\begin{cases} -3x+9, \text{ якщо } x \le 7, \\ \dfrac{1}{x-7}, \text{ якщо } x>7 \end{cases}$	10) $f(x)=\begin{cases} 1.2x^2-3x-9, \text{ якщо } x>3 \\ \dfrac{12.1}{2x^2+1}, \text{ якщо } x \le 3 \end{cases}$

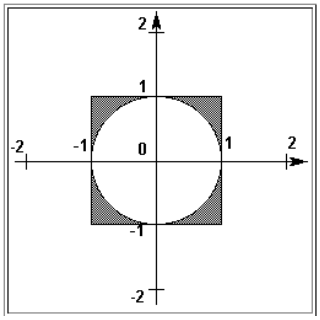
Завдання 4 (*оператор switch***)**

1. Написати програму, яка за введеним символом математичної операції здійснює відповідну дію над двома числами. При помилковому введенні числа виводиться повідомлення про помилку (*приклад 3*).
2. Написати програму, яка за введеним номером дня тижня виводить його назву. При помилковому введенні числа виводиться повідомлення про помилку.
3. Написати програму переведення оцінки з числового формату в текстовий.(Наприклад 2 – "незадовільно").
4. Написати програму, яка за введеною відповіддю користувача виводить повідомлення про завершення роботи програми. Врахувати регістр літер та мову введення відповіді („y”- продовжити роботу, „n” – завершити.)

Додаткові завдання

1. Написати програму яка визначає, чи потрапляє точка з координатами (x, y) у зафарбовану область.





2. Дано три різних дійсних числа. Визначити, яке з них найбільше, найменше та середнє.
3. Відомі рік, номер місяця та день народження двох людей. Визначити хто з них найстарший.
4. Дано три дійсні додатні числа. Якщо існує трикутник з такими сторонами, то визначити його вид (рівносторонній, рівнобедрений, різносторонній)

Контрольні запитання

1. Який обчислювальний процес називається розгалуженим?
2. Які форми умовного оператора Ви знаєте? Опишіть їх структуру.
3. Які відношення використовуються при записі умов в мові С++?
4. Що таке складений оператор?
5. Що таке логічний вираз? Що таке простий і складений логічний вираз?
6. Які структури відповідають оператору вибору у мові С++?

Лабораторна робота №3.
Розробка програм мовою програмування С++
з використанням циклу «з передумовою» та «з післяумовою»

Теоретичні відомості

Оператор циклу з передумовою

Оператор **WHILE** використовується для організації циклічного обчислювального процесу.

Синтаксис: **while (**<*логічний вираз*>**)** *оператор***;**

Цикл закінчується у випадках:

- умовний вираз у заголовку приймає нульове значення (хибне);
- у тілі циклу досягнуто місця, де розташований оператор **BREAK**;
- у тілі циклу виконаний оператор **RETURN**;

У перших двох випадках керування передається оператору, розташованому безпосередньо за циклом, у третьому випадку активна на той момент функція завершує свою роботу, повертаючи якесь значення.

Якщо у тілі циклу необхідно виконати більше ніж один оператор то їх заключають у фігурні дужки.

Оператор циклу з постумовою DO…WHILE.

Оператор циклу з постумовою використовують для організації циклічного виконання оператору, який називається тілом циклу, до тих пір, поки умова не стане хибною.

Синтаксис: **do** <*оператор*> **while (**<*логічний вираз*>**);**

Ситуації що призводять до виходу з циклу, аналогічні наведеним для циклу з передумовою. Характерним є те, що тіло циклу виконується хоча б один раз. Цикл з післяумовою в С++ виконується поки умова істинна і переходить до виконання наступного за циклом оператору, коли умовний вираз набуває нульового значення (хибно).

Приклад програми для виведення таблиці множення на довільне число.

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
    int  i = 0, r ,x;
    cout<<"x=";
    cin>>x;
    while (++i<=10)
    {
       r=x*i;
       cout<<endl <<x<<"*"<<i<<"="<<r;
    }
    _getch();
}
```

Приклад програми, яка обчислює суму послідовності парних чисел, що вводяться з клавіатури. Для завершення послідовності ввести нуль.

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
   int s=0 ,x;
   do
   {
      cout<<"x=";
      cin>>x;
      if(x % 2==0) s+=x;
   }
   while (x!=0);
   cout<<"\n s=" <<s;
   _getch();
}
```

Завдання для самостійного виконання

№проб.місця / №завдання	1	2	3	4	5	6	7	8	9	10
1	1,4	3,5	2,4	1,5	2,4	3,5	1,4	2,3	1,5	2,4
2	3,5	2,4	1,3	2,5	3,4	1,2	3,5	2,4	1,3	2,5
3	1,4	2,5	3,4	1,5	2,4	3,5	1,4	2,5	3,4	1,5

Завдання 1

1. Написати програму, яка обчислює суму послідовності додатних чисел, що вводяться з клавіатури. Для завершення послідовності ввести нуль.

2. Написати програму, яка обчислює середнє арифметичне послідовності додатних чисел, що вводяться з клавіатури. Для завершення послідовності ввести нуль.
3. Написати програму, яка обчислює кількість додатних чисел у послідовності, що вводиться з клавіатури. Для завершення послідовності ввести нуль.
4. Написати програму, яка обчислює кількість від'ємних чисел у послідовності, що вводиться з клавіатури. Для завершення послідовності ввести нуль.
5. Написати програму, яка обчислює середнє арифметичне послідовності від'ємних чисел, що вводяться з клавіатури. Для завершення ведення послідовності ввести нуль.

Завдання 2

1. Написати програму, яка визначає максимальне число у введеній з клавіатури послідовності цілих додатних чисел. Для завершення введення послідовності ввести нуль.
2. Написати програму, для знаходження суми всіх непарних чисел від 101 до 301.
3. Написати програму для знаходження суми всіх цілих додатних парних чисел від 1 до 100.
4. Написати програму, яка виводить таблицю значень кубів чисел, що належать проміжку [1, 9] з кроком 2. Оформити виведення у вигляді: 1^3=1;3^3=9;5^3=125 і т.д.
5. Написати програму, яка обчислює суму всіх парних чисел, менших 20.

Завдання 3

1. Написати програму для знаходження залишку від ділення двох цілих чисел a і b, не використовуючи при цьому функцію mod.
2. Написати програму для обчислення суми $S=x+x^2+\ldots+x^n$.

3. Написати програму для обчислення добутку всіх цілих чисел від 1 до 100 кратних 3 але не кратних 6.
4. Написати програму для обчислення суми всіх цілих чисел від 1 до 100 кратних 4.
5. Написати програму для знаходження першої часткової суми ряду 1+1/2+1/3… , що більше даного числа **a.**

Додаткові завдання

1. Дано натуральне число. Визначити його максимальну та мінімальну цифру та на скільки максимальна більше від мінімальної.
2. Відомі оцінки з інформатики учнів класу. Визначити кількість «5», «4», «3», «2».
3. Відомо зріст учнів класу. Зріст хлопчиків умовно заданий від'ємними числами. Визначити середній зріст хлопчиків та дівчаток.

Контрольні запитання

1. Яку структуру має цикл з післяумовою та цикл з передумовою?
2. Охарактеризуйте особливості роботи циклу з передумовою.
3. Як визначається кількість повторень циклу з післяумовою?
4. Що відбувається, якщо умова стає хибною у циклі з передумовою?
5. Коли найчастіше використовують цикли з післяумовою?

Лабораторна робота №4.
Розробка програм, що містять цикл FOR

Теоретичні відомості

Оператор **FOR** забезпечує циклічне повторення деякого оператора певне число раз. Повторення циклу здійснюється з використанням деякої змінної (лічильника), яка змінюється при кожному виконанні тіла циклу. Повторення завершується, коли лічильник досягає заданого значення.

Синтаксис:

for ([*ініціалізація***]; [***перевірка_умови***]; [***нове _значення***])** *оператор***;**

Кожен із трьох виразів у дужках може бути відсутнім.

Перший вираз служить для ініціалізації лічильника, другий для перевірки кінця циклу, а третій – для зміни значення лічильника.

Приклад програми, яка виводить таблицю квадратів перших п'яти цілих додатних непарних чисел.

```
#include<iostream>
#include<conio.h>
using namespace std;
void main()
{
   float  i, r, g=0;
   for(i=1; ++g<=5 ; i+=2)
   {
      r = pow(i, 2);
      cout<<i<<"^2="<<r<<endl;
   }
   _getch();
}
```

ОПЕРАТОР РОЗРИВУ. Оператор розриву **BREAK** перериває виконання операторів: do, for, while, switch.

У операторі **switch** він використовується для завершення блоку **case**. У операторах циклу – для негайного завершення циклу, що не пов'язано з перевіркою звичайної умови завершення циклу. Коли оператор **BREAK**

зустрічається всередині оператору циклу, то здійснюється негайний вихід з циклу та перехід до виконання оператору, що слідує за оператором циклу.

ОПЕРАТОР ПРОДОВЖЕННЯ. Оператор **CONTINUE** передає управління на наступну ітерацію в операторах циклу. Він може розміщуватися лише в тілі цих операторів. В операціях **do, while** наступна ітерація починається з обчислення виразу умови, а для оператора **for** наступна ітерація починається з обчислення виразу зміни значення лічильника.

Завдання для самостійного виконання.

Роб.місце / Завдання	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	1, 4	3, 5	2, 4	1, 5	2, 4	3, 5	1, 4	2, 3	1,5	2, 4
2	3, 5	2, 4	1, 3	2, 5	3, 4	1,2	3, 5	2, 4	1, 3	2,5
3	1, 4	2, 5	3, 4	1, 5	2, 4	3, 5	1, 4	2, 5	3, 4	1, 5

Завдання 1

1. Написати програму, яка виводить таблицю квадратів перших десяти цілих додатних чисел.

2. Написати програму, яка виводить таблицю квадратів перших п’яти цілих додатних непарних чисел.

3. Написати програму, яка виводить суму перших N цілих додатних чисел. (кількість вводиться під час роботи програми).

4. Написати програму, яка виводить суму перших N цілих додатних парних чисел. (кількість вводиться під час роботи програми).

5. Написати програму, яка виводить таблицю множення на довільне число.

Завдання 2

1. Написати програму, яка обчислює суму перших N елементів ряду 1+1/2+1/3+.... (кількість вводиться під час роботи програми).

2. Написати програму, яка обчислює суму перших N елементів ряду 1/sin(x)+1/sin(2x)+1/sin(3x).... (кількість вводиться під час роботи програми).

3. Написати програму, яка обчислює суму перших N елементів ряду 1/tg(2x)+1/tg(4x)+1/tg(6x).... (кількість вводиться під час роботи програми).

4. Написати програму, яка обчислює суму перших N елементів ряду 1/2+1/4+1/6... (кількість вводиться під час роботи програми).

5. Написати програму, яка обчислює суму перших N елементів ряду 1+1/3+1/5+.... (кількість вводиться під час роботи програми).

Завдання 3

1. Написати програму, яка виводить таблицю значень функції y = 3x+2 в діапазоні (-2;2) з кроком 0,5.

2. Написати програму, яка виводить на екран таблицю вартості яблук в діапазоні від 100 г до 1 кг. з кроком 100г.(вартість вводиться під час роботи програми).

3. Написати програму визначення середнього арифметичного послідовності чисел введених з клавіатури (кількість вводиться під час роботи програми).

4. Написати програму визначення кількості доданих та від’ємних чисел введених з клавіатури (кількість вводиться під час роботи програми).

5. Написати програму визначення суми доданих та добутку від’ємних чисел введених з клавіатури (кількість вводиться під час роботи програми).

Додаткові завдання

1. Дано натуральні числа x та y. Визначити добуток цих чисел використовуючи лише операцію додавання.
2. Написати програму, яка виводить на екран таблицю множення (квадрат Піфагора).

Контрольні запитання

1. Що таке цикл і навіщо він використовується в програмуванні?
2. Яки види циклічних структур існують в мові C++?
3. Яку структуру має цикл **for**?
4. Як реалізується зміна параметру циклу?

Лабораторна робота №5.
Функції у мові С++

Теоретичні відомості

Функції використовуються для спрощення процесу розробки програми у випадку, коли аналогічні перетворення над різними даними необхідно виконати декілька раз у різних частинах програми.

Кожна програма у своєму складі повинна містити **ГОЛОВНУ ФУНКЦІЮ - main()**, яка забезпечує створення точки входу в об'єктний модуль. Окрім цієї функції в програму може входити довільне число функцій, виконання яких ініціалізується або прямо чи опосередковано викликається із функції **main**().

Опис функції. Складається з двох частин: заголовку та тіла.

[тип_результату] <ім'я> ([список _параметрів])
{
<тіло функції>
}

Виклик функції може бути оформлений у вигляді оператору, якщо у функції відсутнє значення, яке вона повертає

ім'я_функції (*список_аргументів*);

Опис функції може знаходиться в одному файлі з головною програмою.

Приклад опису та виклику функції

```
#include<iostream>
#include<conio.h>
using namespace std;
/*функція визначення
максимального із 2-х чисел*/
int max(int x, int y)
{
   if(x>y) return x;
   else return y;
}
```

```
void main()
{
   int a,b,c;
   cout<<"\n введіть 2 числа > ";
   cin>>a>>b;
   c=max(a,b);
   cout<<"\n max = "<<c;
   _getch();
}
```

Завдання для самостійного виконання

Завдання1

1. Реалізувати підпрограму, яка визначає максимальне з трьох чисел.
2. Реалізувати підпрограму, яка визначає периметр трикутника.
3. Реалізувати підпрограму визначення площі трикутника по трьом сторонам.
4. Реалізувати підпрограму, яка перевірятиме введене з клавіатури число на парність.
5. Реалізувати підпрограму виведення імені у стовпчик декілька раз.
6. Реалізувати підпрограму визначення площі трикутника по координатам вершин.

Додаткові завдання

1. Реалізувати підпрограму виведення поля заповненого зірочками.
2. Реалізувати підпрограму створення рамки з довільних символів (символ та розмір рамки вводиться підчас роботи програми)

Контрольні запитання

1. Як описується функція?
2. Як викликається функція в тексті основної програми?
3. Що таке прототип функції?
4. Які змінні називаються локальними, а які глобальними?

Лабораторна робота №6.
Обробка одновимірних масивів у мові С++

Теоретичні відомості

Мова С++ дозволяє обробляти набори однотипних даних – масиви. У якості елементів масиву можуть виступати дані будь якого типу. А також вказівними на однотипні дані.

Використання масиву починається з його оголошення, яке виділяє під масив певний об’єм пам’яті

<тип> <імя>[n];

Ініціалізація масивів може відбуватися при оголошенні масиву, значення елементів записується у фігурних дужках та перераховуються через кому в порядку слідування у масиві.

int A [5]={2,4,6,8,10};

При оголошенні одномірного масиву з одночасною його ініціалізацією дозволяється опускати значення розміру у квадратних дужках. Компілятор сам підраховує кількість елементів у списку ініціалізації та виділяє необхідну область пам’яті.

int A [ ]={2,4,6,8,10};

Звертаючись до елементу масиву може здійснюватися одним з двох способів:

3. за номером елементу в масиві;
4. за покажчиком.

При звертанні через індекс за ім’ям масиву в квадратних дужках вказується номер елементу, до якого звертаються. У мові С++ елементи масиву нумеруються починаючи з нуля.

X=A [14]; *//змінній X присвоєно значення 15-го елементу.*

Присвоєння значень одного масиву значенням другого є недопустимим, оскільки компілятор не може самостійно скопіювати всі значення одного

масиву в значення іншого. Для реалізації цієї задачі необхідно використати, наприклад, циклічне присвоєння.

Завдання для самостійного виконання

Роб.місце / Завдання	1	2	3	4	5	6	7	8	9	10
1	3,5	2,4	1,3	2,5	3,4	1,2	3,5	2,4	1,3	2,5
2	1,4	2,5	3,4	1,5	2,4	3,5	1,4	2,5	3,4	1,5
3	1,4	3,2	1,4	3,2	1,4	3,2	1,4	3,2	1,4	3,2

Приклад програми визначення суми елементів масиву

```
#include<iostream>
#include<conio.h>
using namespace std;
void main( )
{
   int a[5], s=0;
   //введення елементів масиву
   for (int i=0; i<5; i++)
   {
      cout<<"a[" <<i<<"]=";
      cin>> a[i];
      s+=a[i];
   }
   cout<<"\n s="<<s<<"\n";
   _getch();
}
```

Приклад програми заміни від'ємних елементів масиву на нуль

```
include<iostream>
#include<conio.h>
using namespace std;
void main( )
{
   int a[5], i;
   for ( i=0; i<5; i++)
   {
      cout<<"a["<<i<<"]=";
      cin>>a[i];
      if (a[i]<0) a[i]=0;
   }
   //виведення елементів масиву в рядок
   for (i=0; i<5; i++)
   cout<< a[i] <<"  ";
   _getch();
}
```

Завдання 1.

1. Написати програму, яка визначає кількість нульових елементів масиву
2. Написати програму, яка підраховує кількість від'ємних елементів масиву
3. Написати програму, яка підраховує кількість парних елементів масиву

4. Написати програму, яка визначає, скільки раз у масиві зустрічається введене з клавіатури число.

5. Написати програму, яка визначає, чи зустрічається у масиві введене з клавіатури число.

Завдання 2.

1. Написати програму, яка визначає середнє арифметичне непарних елементів масиву

2. Написати програму, яка визначає середнє арифметичне ненульових елементів масиву.

3. Написати програму, яка визначає суму елементів, які мають непарні порядкові номери.

4. Написати програму, яка визначає кількість учнів у класі чий зріст вище середнього.

5. Написати програму, яка визначає середню температуру повітря за тиждень.

Завдання 3.

1. Написати програму, яка заміняє всі парні елементи масиву на „0”.

2. Написати програму, яка підраховує кількість нульових елементів масиву.

3. Написати програму, яка визначає мінімальний елемент масиву та його порядковий номер.

4. Написати програму, яка визначає максимальний елемент масиву та його порядковий номер.

Додаткові завдання

1. Замінити всі від’ємні елементи одновимірного масиву цілих чисел на значення найменшого з додатних чисел (нуль не враховувати).

2. Підрахувати кількість пар сусідніх елементів одновимірного масиву цілих чисел з однаковими значеннями.
3. Визначити, яке значення зустрічається у масиві дійсних чисел найчастіше.

Контрольні запитання

1. Який масив називається простим або одновимірним?
2. Які здійснюється оголошення масиву.?
3. Як здійснюється ініціалізація елементів масиву?
4. Як здійснюється введення та виведення елементів одновимірного масиву?

Лабораторна робота №7.
Обробка багатовимірних масивів у мові С++

Теоретичні відомості

Багатовимірний масив розмірності **N** можна представити як одновимірний масив із масивів розмірності (**N-1**). Таким чином тривимірний масив – це масив, кожний елемент якого представляє двовимірну матрицю. При цьому розмірність масиву визначається числом парних квадратних дужок при його оголошенні.

int M [3] [3] = {1,2,3,4,5,6,7,8,9}

На початку роботи у пам'яті резервується місце для дев'яти елементів із заповненням масиву.

Для багатовимірних масивів при ініціалізації дозволяється опускати величину першої розмірності **int M [ ] [3] = {1,2,3,4,5,6,7,8,9}**

Введення елементів двовимірного масиву може здійснюватися з клавіатури, або генеруватися випадковим чином.

```
// введення елементів масиву з клавіатури
for ( i=0; i<2; i++)
for ( j=0; j<3; j++)
{
   cout<<"\ n введіть елемент масиву";
   cin>>a[i][j];
}
```

```
// автоматичне заповнення масиву випадковими числами в діапазоні від 1 до 10
#include <time.h>
  …….
srand(time(NULL));
for ( i=0; i<2; i++)
for ( j=0; j<3; j++)
a[i][j] = rand()%10;
```

Якщо елементи масиву генеруються випадковим чином, необхідно підключити заготовочний файл **time.h**

Для встановлення діапазону можна задати такі параметри:

a[i][j] = rand()%10; *// числа генеруються в діапазоні 1..10*

a[i][j] = rand()%100; *// числа генеруються в діапазоні 1..100*

a[i][j] = rand()%(0+10); *// числа генеруються в діапазоні 0..10*

a[i][j] = rand()%20-10; *// числа генеруються в діапазоні -10..10*

Виведення елементів масиву у вигляді таблиці здійснюється за допомогою конструкції:

```
for (int i=0; i<3; i++)
{
    for (int j=0; j<3; j++)
    cout << a[i][j]<< “   “;
    cout <<”\n”;
}
```

Приклад програми визначення максимального елементу двовимірного масиву

```
#include <time.h>
#include <conio.h>
#include <iostream>
using namespace std;
void main( )
{
  srand(time(NULL));
  int a[4][3], i,  j;
  //заповненя масиву довільним чином
  for ( i=0; i<4; i++)
  for ( j=0; j<3; j++)
  a[i][j] = rand()%10;
  //визначення максного елементу
  int max=a[0][0];
  for (i=0;i<4;i++)
  for (j=0;j<3;j++)
  if (a[i][j]>max) max=a[i][j];
  //виведення масиву на екран
  for (int i=0; i<4; i++)
  {
     for (int j=0; j<3; j++)
     cout<< a[i][j]<< "  ";
     cout<<"\n";
  }
  // виведення результату
  cout<< "\n max= "<<max;
  _getch();
}
```

Приклад програми визначення суми парних елементів масиву.

```
#include <time.h>
#include <conio.h>
#include <iostream>
using namespace std;
void main( )
{
    srand(time(NULL));
    int a[8][8], s=0, i, j;
    for ( i=0; i<8; i++)
    for ( j=0; j<8; j++)
    {
    a[i][j] = rand()%10;
    if (a[i][j] %2==0) s+=a[i][j];
    }
    for (int i=0; i<8; i++)
    {
        for (int j=0; j<8; j++)
        cout<< a[i][j]<< "  ";
        cout<<"\n";
    }
    cout<< " \n s=  "<<s;
    _getch();
}
```

Завдання для самостійного виконання

Роб.місце / Завдання	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	2,4	3,5	2,4	1,3	2,5	3,4	1,2	3,5	2,4	1,3
2	3,5	1,4	2,5	3,4	1,5	2,4	3,5	1,4	2,5	3,4
3	3,2	1,4	3,2	1,4	3,2	1,4	3,2	1,4	3,2	1,4

Завдання 1

1. Написати програму, яка визначає кількість нульових елементів масиву
2. Написати програму, яка підраховує кількість від'ємних елементів масиву
3. Написати програму, яка підраховує кількість парних елементів масиву
4. Написати програму, яка визначає, скільки раз у масиві зустрічається введене з клавіатури число.
5. Написати програму, яка визначає, чи зустрічається у масиві введене з клавіатури число.

Завдання 2

1. Написати програму, яка визначає мінімальний елемент масиву та його порядковий номер.
2. Написати програму, яка визначає максимальний елемент масиву та його порядковий номер.
3. Написати програму, яка визначає середнє арифметичне непарних елементів масиву
4. Написати програму, яка визначає середнє арифметичне ненульових елементів масиву.
5. Написати програму, яка визначає суму елементів, які мають непарні порядкові номери.

Завдання 3

1. Написати програму, яка заміняє всі парні елементи масиву на „0”.
2. Написати програму, яка заміняє всі елементи масиву з парними порядковими номерами на „0”.
3. Написати програму, яка підраховує суму елементів першого стовпчика.
4. Написати програму, яка підраховує суму елементів другого рядка.

Додаткові завдання

1. Визначити суму елементів головної діагоналі.
2. Написати програму, яка заміняє всі елементи першого та останнього рядку та стовпчику на „0”.
3. Заповніть двовимірний масив випадковими числами. Відобразіть матрицю симетрично відносно головної діагоналі.

Контрольні запитання

1. Який масив називається багатовимірним?
2. Які здійснюється оголошення багатовимірного масиву?
3. Як необхідно звертатися до масиву та до елементу багатовимірного масиву?
4. Як здійснюється введення та виведення елементів багатовимірного масиву?
5. Як здійснюється доступ до елементів багатомірного масиву через покажчики

Лабораторна робота №8.
Представлення символьних даних у мові С++
Теоретичні відомості

Масиви символів

Символьних рядків в С++ представляються як масиви елементів типу CHAR, які закінчуються ***термінатором рядку*** (символом з нульовим значенням **\0**). Символьні рядки складаються із набору символьних констант, заключних у подвійні лапки.

У якості оператора введення при роботі з рядками замість оператора запису в поток (**>>**) краще використовувати функцію **getline()** так як потоковий оператор введення ігнорує пробіли та може продовжувати введення елементів за межами масиву.

Функція **getline(a, n)** де
a - вказує на рядок, в який відбувається введення
n - кількість символів, які буде введено.

```
void main()
{
 char a[5];                          // оголошення масиву символів
 cout<<"Як твоє ім'я \n";
 cin.getline(a,5);                   // введення масиву символів
 cout<<" Вітаю "<< a;
 _getch();
}
```

Інколи у якості другого параметра функції **GETLINE()** використовується функція **SIZEOF**(). Це робить код універсальним та не вимагає явного визначення числа символів, які необхідно ввести.

cin.getline(a, sizeof(a));

Стандартний клас String

Даний клас дозволяє визначити у програмі рядковий тип даних. Щоб отримати доступ до методів класу необхідно підключити заготовочний файл **string.** Об'єкти класу **string** не закінчується нульовим байтом, як це прийнято при представлені рядків за допомогою масиву символів.

Для введення змінних цього типу можна використовувати **cin**, або спеціальну функцію **getline.**

getline(cin, s);

де **s** — ім'я змінної типу **string**.

Оголошення та ініціалізація рядків.

string s1; // *порожній рядок*
string s2 ("Hi");
string s3="word";

У класі **string** визначено оператори для роботи з рядками, які дозволяють використовувати об'єкти класу у виразах програми.

Оператор	**Приклад**	**Результат**
=	s1=s3	s1="word"
+	s1=s2+s3; s1=s2+" , " +s3	s1="Hiword" s1="Hi, word"
+ =	s1+="!!!";	s1="Hi, word!!!"

Основні методи обробки рядків

Знищення символів. Метод **erase(n, k)** – знищує **k** символів рядку починаючи з позиції **n.** (*Індексування символів починається з нуля*)

Наприклад:
string s ="My program";
s.erase (3, 4);
cout<< s; *// буде виведено повідомлення «My ram»;*
cout<<s.erase(); *// буде виведено порожній рядок;*

Вставка символів. Метод **insert(n, r)** – вставляє рядок **r** у позицію **n** визначеного рядка.

Наприклад:
string s ="My ram", s1="prog";
s.insert (3, s1);
cout<< s; *// буде виведено повідомлення «My program»;*

Заміна символів. Метод **replace(n,k,r)** - Заміняє **k** символів рядку, починаючи з позиції **n** заданим рядком **r.**

Наприклад:

string s1 = “I like ice cream”,s2 = “apples”
s1.replace (7,9,s2);
cout<<s1; *// буде виведено повідомлення «I like apples»;*

Визначення довжини рядка. Для визначення довжини рядка використовують два методи **size()** та **length(),** які повертають кількість символів у рядку типу string

Наприклад:

string r = “Hi, world”;
cout<< r.size() << endl; *// на екран буде виведено 9*
cout<< r.length( ) << endl; *// на екран буде виведено 9*

Приклад визначення кількості літер «а» у рядку:

1) з використанням стандартного класу String

```
#include<iostream>
#include<conio.h>
#include<string>
using namespace std;
void main()
{
   string a;
   int k=0;
   cout<<">";
   getline(cin, a);
   int x=a.length();
   for(int i=0;i<x;i++)
   if(a[i]=='a') k++;
   cout<<" > "<< k;
   _getch();
}
```

2) з використанням масиву символів

```
#include<iostream>
#include<conio.h>
using namespace std;

void main()
{
  char a[50];
  int k=0;
  cout<<"\n > ";
  cin.getline(a,sizeof(a));
  for(int i=0;i<sizeof(a);i++)
  if(a[i]=='a') k++;
  cout<<" > "<< k;
  _getch();
}
```

Завдання для самостійного виконання

Роб.місце / Завдання	1	2	3	4	5	6	7	8	9	10
1	2, 4	3, 5	2, 4	1, 3	2, 5	3, 4	1, 2	3, 5	2, 4	1, 3
2	3, 5	1, 4	2, 5	3, 4	1, 5	2, 4	3, 5	1, 4	2, 5	3, 4

Завдання І.

1. Написати програму, яка визначає кількість літер „а” у рядку.
2. Написати програму, яка підраховує кількість ком у рядку.
3. Написати програму, яка підраховує кількість слів у рядку.
4. Написати програму, яка визначає, скільки раз у рядку зустрічається літера введена з клавіатури.
5. Написати програму, яка визначає, чи зустрічається у рядку введена з клавіатури літера.

Завдання ІІ.

1. Написати програму, яка знищує всі пробіли.
2. Написати програму, яка заміняє всі літера „а” на „1”.
3. Написати програму, яка заміняє кожен другий символ рядку на „1”.
4. Написати програму, яка визначає кількість літер у речення. (до першої крапки)
5. Написати програму, яка визначає кількість ком у реченні (до першої крапки).

Додаткові завдання

1. Написати програму, яка обертає слово, якщо воно має парну кількість літер.
2. Написати програму, яка збільшує рядок символів до більшого регістру, якщо він починається з літери А.

3. З клавіатури вводиться текстовий рядок. Скласти програму, яка підраховує кількість слів, які мають непарну довжину.
4. З клавіатури вводиться текстовий рядок. Скласти програму яка видаляє текст, що розміщено у круглих дужках.
5. З клавіатури вводиться текстовий рядок. Скласти програму, яка перевіряє, чи співпадає кількість відкритих та кількість закритих дужок у введеному рядку.
6. З клавіатури вводиться текстовий рядок. Скласти програму яка виводить найдовше слово
7. Скласти програму яка підраховує кількість розділових знаків у рядку.

Контрольні запитання

1. Як оголошується стандартний клас String?
2. Як описати рядкову змінну?
3. Які операції над рядками можна виконати?
4. Назвіть методи обробки рядків.

Лабораторна робота №9
Структури

Теоретичні відомості

Оголошення структури

Структури дозволяють об'єднувати в єдиному об'єкті сукупність значень, які можуть мати різні типи. Оголошення структури здійснюється за допомогою ключового слова **STRUCT**.

Використання структурованих даних в тілі програми можливе у випадку якщо буде оголошено об'єкт новоствореного типу.

```
STRUCT [імя_структури]
{
тип_1  елемент_1;
тип_2  елемент_2;
.....
тип_n  елемент_ n;
};
```

Наприклад:

```
STRUCT date
{
    int day;
    int month;
    int year;
    char name [30];
} b;    // елемент типу date
```

На відміну від описів інших типів даних, опис структури не виділяє місяця у пам'яті під елементи структури. ЇЇ опис визначає лише шаблон, що описує характеристики змінних, що будуть розміщенні у конкретній структурі. Щоб ввести змінні та зарезервувати для них пам'ять необхідно або після фігурної дужки, що завершує опис структури, вказати список ідентифікаторів, або окремо оголосити змінну типу.

Доступ до окремого елемента структури забезпечується операторами вибору: (прямий селектор „ . „ *крапка*).

Наприклад: **b.day = 31;**

Ініціалізація структури подібна до тієї, що і в масивах, але з урахуванням розміщення даних різного типу.

Наприклад: **date b = {10, 7, 2001, “Petrov”};**

Для змінних одного структурного типу визначена операція присвоювання, при цьому здійснюється поелементне копіювання значень полів. Але для порівняння структур необхідно перевірити рівність відповідних полів цих структур.

Приклад програми «книжкова крамниця».

```
#include <iostream>
#include <conio.h>
using namespace std;
struct book
{
   char titel[10];
   char autor[10];
   float price;
} b;
void main()
{
   cout<<"Введіть автора >";
   cin.getline(b.autor, 10);
   cout<<"Введіть назву  >";
   cin.getline(b.titel, 10);
   cout<<"Введіть ціну >";
   cin>>b.price;
   cout<<b.autor<<"   "<<b.titel<<"   "<<b.price;
   _getch();
}
```

Масиви структур

Масивами структур можна оперувати як звичайними масивами. Оголошення масиву структур вимагає вказати в якості типу масиву, який створюється новоутворений тип структури. Доступ до полів структури аналогічний доступу до звичайних змінних.

Наприклад:

```
STRUCT date
{
int day, month, year;
char name [30];
} a[100];    // a[100] – масив елементів типу date,
```

Доступ до окремого елемента структури забезпечується операторами вибору: (прямий селектор „ . „ *крапка*). *Наприклад:* **a[10].day = 31;**

Приклад. База даних книжкова крамниця з використанням классу STRING.

```
#include <iostream>
#include <conio.h>
#include <string>
using namespace std;
struct book
{
   string titel;
   string autor;
   float price;
} B[3];
void main()
{
   for(int i=0;i<3;i++)
   {
      cout<<" Введіть автора >";
      cin>>B[i].autor;
      cout<<" Введіть назву  >";
      cin>>B[i].titel;
      cout<<" Введіть ціну >";
      cin>>B[i].price;
   }
   for(int i=0;i<3;i++)
   cout<<B[i].autor<<"   "<<B[i].titel<<"   "<<B[i].price<< endl;
   _getch();
}
```

Завдання для самостійного виконання

Завдання 1. Реалізуйте приклади. Проаналізуйте особливості роботи із масивом структур. Порівняйте роботу із типом даних **char** та стандартним класом **string**.

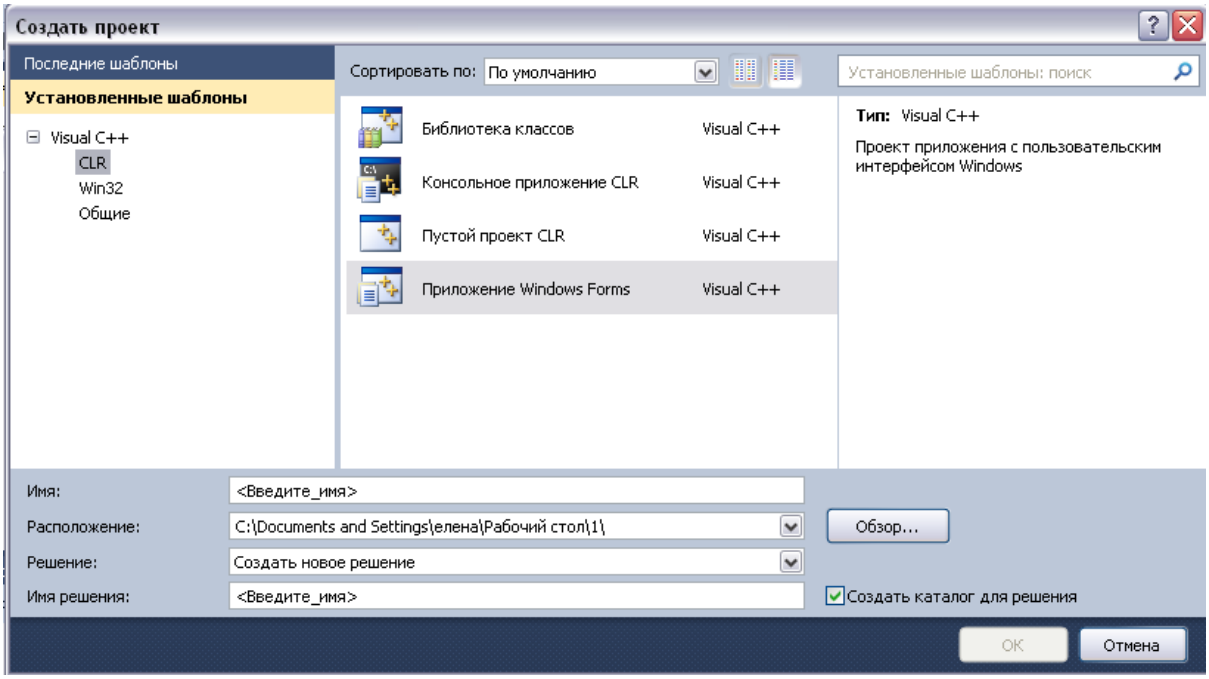
Завдання 2. Використовуючи тип даних масив структур, скласти програму, за допомогою якої можна ввести у пам'ять комп'ютера дані про наявність на складі автомашин і отримати інформацію про марки та рік випуску машин, ціна яких менша, ніж 3000. Нехай запис містить такі поля: марка, рік випуску та ціна машини. Вивести на екран інформацію про всі машини і додатково про ті, ціна яких менша, ніж 3000.

Лабораторна робота №10.
Ознайомлення з візуальним середовищем Visual C++. Компіляція та запуск програми на виконання.

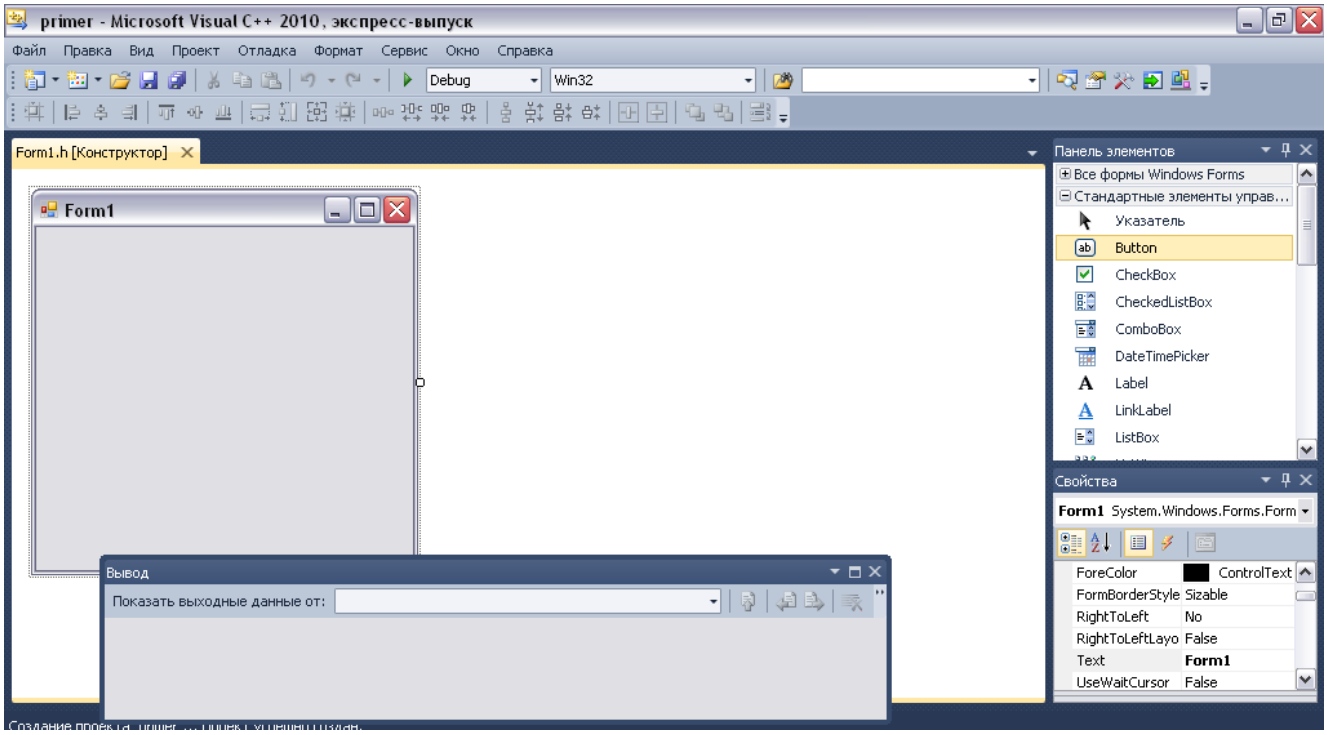
Теоретичні відомості

1. Після завантаження середовища програмування необхідно виконати команду **«Файл-Создать-Проект».**

2. У вузлі Visual C++ області встановлених шаблонів необхідно вибрати **CLR**, а потім у області шаблонів (у середній колонці) обрати шаблон «**Приложения Windows Forms (Visual C++)».** Після чого необхідно ввести ім'я проекту в поле «**Имя»** і шлях до файлів проекту.



3. У вікні додатку що створився зображена *екранна форма* — **Form1**. Форма відобразиться у вкладці F**orm1.h [Конструктор].** У формі розташовують різні компоненти графічного інтерфейсу користувача.



4. Слід пам'ятати, що об'єкти не тільки мають властивості, але і обробляються подіями. Подією, наприклад, є клацання на кнопці **Click**.

Управляють тим або іншою подією за допомогою написання процедури обробки події в програмному коді. Для цього спочатку потрібно одержати «порожній» обробник події. Для отримання порожнього обробника цієї події необхідно у властивостях елемента клацнути на зображенні блискавки «**События**» і в списку всіх можливих подій вибрати подвійним клацанням необхідну подію. Після цього активізується вкладка програмного коду **Form1.h**. Перемикатися між вкладками форми і програмного коду можна за допомогою миші або за допомогою комбінації клавіш **Ctrl+Tab**, як це прийнято звичайно при перемиканні між вкладками в Windows.

5. Запуск проекту здійснюється командою «**Отладка-Начать отладку**» або клавіша **F5** або кнопка із стрілкою на панелі інструметів.

***

Для виведення повідомлень у окремому вікні можна використати метод **Show**.

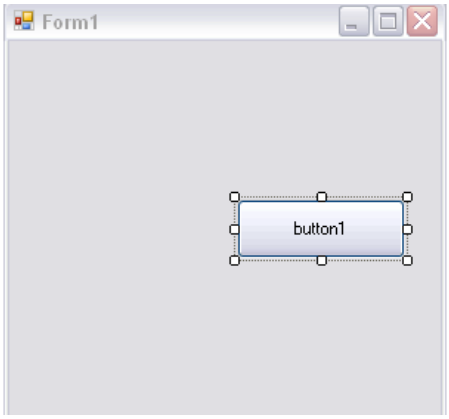
MessageBox::Show("Текст");

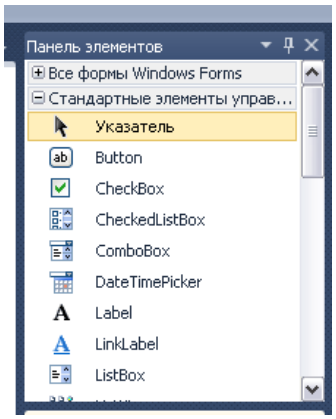
Тут викликається метод **Show** об'єкту **MessageBox** з текстом Оператор розширення області дії (::) указує системі знайти метод **Show** серед методів об'єкту **Messagebox**. Таким чином, об'єкти окрім властивостей мають також і ***методи***, тобто програми, які обробляють об'єкти

Завдання для самостійного виконання

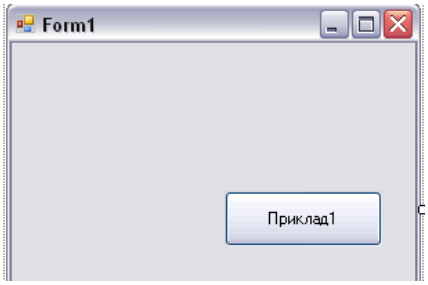
Завдання 1. Виведення повідомлення.

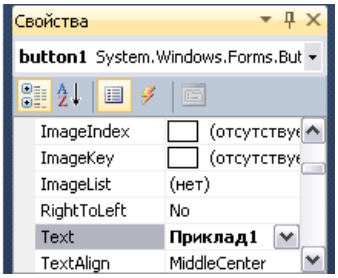
1. Створіть проект **Lesson1**. На формі розмістіть кнопку





2. Додайте на кнопку надпис **Приклад1** скориставшись властивістю **Text**.

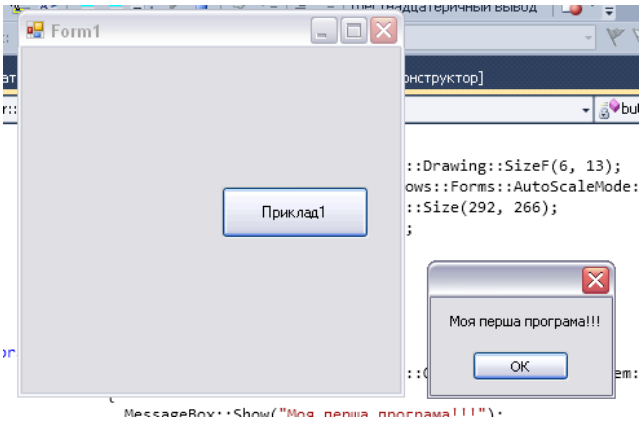




3. Двічі клікніть по кнопці «**Приклад1**». Відкриється вкладка **Form1.h** при цьому сформується порожній обробник подій (**button1_Click**). Між фігурними дужками { } пропишіть програмний код **MessageBox::Show("Моя перша програма!!!");**

```
Form1.h*  Form1.h [Конструктор]*
primer::Form1                    button1_Click(System::Object ^ sender, System::EventArgs ^ e)
            //
            this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
            this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
            this->ClientSize = System::Drawing::Size(292, 266);
            this->Controls->Add(this->button1);
            this->Name = L"Form1";
            this->Text = L"Form1";
            this->ResumeLayout(false);

        }
#pragma endregion
    private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
             {
               MessageBox::Show("Моя перша програма!!!");
             }
```

4. Відкомпілюйте проект та запустіть на виконання «**Отладка-Начать отладку».**



5. Додайте на форму другу кнопку **ВИХІД** та пропишіть у обробнику подій для елементу **button2** метод **Close()**. Ви створили кнопку закриття проекту.

Лабораторна робота №11.
Інструменти візуальної розробки додатків.

Теоретичні відомості

Форма є основою додатку. У формі розташовують різні компоненти графічного інтерфейсу користувача або, як їх інакше називають, елементи управління.

Це поля для введення тексту **TextBox,** командні кнопки **Button,** мітки **Label,** які не можуть бути відредаговані користувачем, і інші елементи управління. Візуальне програмування, припускає просте перетягування елементів за допомогою миші з **панелі елементів,** де розташовані компоненти користувача, у форму. Це допомагає звести до мінімуму безпосереднє написання програмного коду.

Властивостями кнопки є:

- ім'я кнопки (**Name**) — **button**1;
- напис на кнопці (**Text**);
- розташування кнопки (**Location**) в системі координат форми X, Y;
- розмір кнопки (**Size**);
- активність (**Enabled**);
- видимість (**Visible**);

Властивості можна побачити, якщо клацнути правою кнопкою миші в межах форми і вибрати в контекстному меню команду **«Свойства»**, при цьому з'явиться панель **«Свойства» (***якщо вона не відображається на екрані***).**

Слід пам'ятати, що об'єкти не тільки мають властивості, але і обробляються подіями. Подією, наприклад, є клацання на кнопці **Click**, клацання в межах форми, завантаження (**Load**) форми в оперативну пам'ять при старті програми.

Управляють тим або іншою подією за допомогою написання процедури обробки події в програмному коді. Для цього спочатку потрібно одержати «порожній» обробник події. Для отримання порожнього обробника цієї події необхідно у панелі **«Свойства»** клацнути на значку блискавки **«События»** і

в списку всіх можливих подій вибрати подвійним клацанням необхідна подія. Після цього активізується вкладка програмного коду **Form1.h**. Перемикатися між вкладками форми і програмного коду можна за допомогою миші або за допомогою комбінації клавіш **Ctrl+Tab**, як це прийнято звичайно при перемиканні між вкладками в Windows.

Запуск проекту здійснюється командою «**Отладка-Начать отладку»** або клавіша **F5** або кнопка із стрілкою на панелі інструметів.

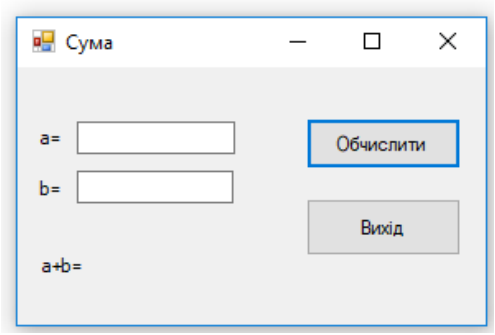
У бібліотеці візуальних компонентів існує множина компонентів, що дозволяють відображати, вводити та редагувати текстову інформацію

Завдання для самостійного виконання

УВАГА! При використанні математичних функцій необхідно вказувати ім'я класу **Math**. *Наприклад*: **s=Math::Sqrt(m);**

Завдання 1.

1. Написати програму знаходження суми двох чисел. Напишіть коментарі до кожного рядка програми.




```
private: System::Void button1_Click(…)
{
        double a,s,b;
        a=Convert::ToDouble(textBox1->Text);
        b=Convert::ToDouble(textBox2->Text);
        s=a+b;
        label1->Text=Convert::ToString(s);
}
```

//**Convert::ToDouble** – перетворення змінної до дійсного типу

//**Convert::ToString** - перетворення змінної до рядкового типу

2. Напишіть програму, яка буде за введеним роком вашого народження виводитиме ваш вік.

3. Задано два різних дійсних числа визначте максимальне з них.

4. Написати програму, яка визначає з двох дійсних чисел додатне та збільшує його значення на одиницю (*операція інкременту*).

5. Визначить кількість коренів квадратного рівняння.

Завдання 2. (Оператор IF)

>> Перейти до теми «Програмування розгалужень у мові С++»

1. Маємо три дійсних числа x, y, z. Знайдіть найменше з них.
2. Маємо дійсні додатні числа x, y, z. Складіть програму, яка буде перевіряти існування трикутника з довжинами сторін **x, y, z**. Визначити периметр трикутника.
3. Серед значень функції y= sin(x) у трьох заданих точках **a, b, c** визначити кількість різних.
4. Складіть програму, яка перевірятиме введене з клавіатури число на парність.
5. Перевірити чи є введене з клавіатури число додатнім, від'ємним, нулем.
6. Визначити, у якій з трьох даних точок *a, b, c* функція y=cos(x) набуває найбільшого значення.
7. Дано три різних дійсних числа *a, b, c*. Знайти добуток двох менших чисел.
8. Дано три різних дійсних числа *a, b, c*. Замінити найменше число добутком двох більших.

Завдання 3. Розробіть додаток використовуючи компоненти введення та виведення даних (**Label, TextBox**). Врахувати перевірку чи заповнено всі необхідні поля.

📖 *Методичні вказівки:*

If (textBox1->Text=="") textBox1->Focus(); *//перевірка умови - якщо поле TextBox не заповнено то курсор встановлюється у поле TextBox*
If (textBox1->Text->Length != 0) button1->Enabled = true; *// якщо довжина тексту у компоненті TextBox не дорівнює нулю то кнопка стає активною*
If (textBox1->Text->Length == 0) button1-> Visible = false; *// якщо довжина тексту у компоненті TextBox дорівнює нулю (тобто поле порожне) то кнопка стає невидимою.*
УВАГА! Активація видимості та активності кнопки реалізується у події **TextChanged** компоненту **textBox**

1. Написати програму визначення вартості придбаного товару. У поле форми ввести кількість одиниць товару, вартість товару - константа. (*Кнопка "обчислити" активна лише за умови якщо введено всі необхідні дані.*)

2. Обчислити периметр трикутника. (*Кнопка "обчислити" видима лише за умови якщо введено всі необхідні дані.*)

3. Написати програму переведення метрів у сантиметри. (*Якщо одне з полів не заповнено, то виводиться повідомлення про помилку і курсор встановлюється у порожнє поле.*)

4. Написати програму знаходження значення виразу p =y*x. (*Якщо одне з полів не заповнено, то виводиться повідомлення про помилку і курсор встановлюється у порожнє поле.*)

📖 *Методичні вказівки до задачі 3:*

```
{
   int m, s;
   if (textBox1->Text=="") textBox1->Focus();
   else
   {
     m=Convert::ToInt32(textBox1->Text);
     s=100*m;
     label1->Text=Convert::ToString(s);
   }
 }
```

Завдання 4. (Оператор SWITCH)

>> Перейти до теми «Оператор вибору Switch»

1. Написати програму, яка за введеним символом математичної операції здійснює відповідну дію над двома числами. При помилковому введенні числа виводиться повідомлення про помилку.

📖 *Методичні вказівки до задачі 1:*

```
…
a=Convert::ToDouble(textBox1->Text);
b=Convert::ToDouble(textBox2->Text);
char c = textBox3->Text[0]; //визначаємо символ операції
switch(c)
{
   case '+':s=a+b;break;
   …
}
…
```

2. Написати програму, яка за введеним номером дня тижня виводить його назву. При помилковому введенні числа виводиться повідомлення про помилку.

3. Написати програму переведення оцінки з числового формату в текстовий.(Наприклад 2 – "незадовільно").

4. Написати програму, яка за введеною відповіддю користувача виводить повідомлення про завершення роботи програми. Врахувати регістр літер та мову введення відповіді („y”- продовжити роботу, „n” – завершити.)

Завдання 5. (Оператор циклу)

>> Перейти до теми «Цикли в мові С++»

📖 *Методичні вказівки:*

***Приклад**:Підрахувати кількість від'ємних чисел із послідовності введених з клавіатури. (Для введення використати компонент* **TextBox**. Властивість **Multiline = True**)

```
int x,k=0;
for(int i=0;textBox2->Lines[i]!="";i++)
{
   x=Convert::ToInt32(textBox2->Lines[i]);
   if(x<0)k++;
}
MessageBox::Show("кількість елементів<0 ="+k);
```

***Приклад**: Підрахувати суму парних чисел із послідовності введених з клавіатури, поки не введено нуль. (Для введення використати компонент* **TextBox**. Властивість **Multiline = True**)

```
int x,s=0;
for(int i=0;textBox2->Lines[i]!="";i++)
{
   x=Convert::ToInt32(textBox2->Lines[i]);
   if(x= =0)break;
   if(x%2= =0) s+=x;
}
MessageBox::Show("сума парних чисел ="+s);
```

****

1. Написати програму, яка підраховує суму N непарних чисел введених з клавіатури.

2. Написати програму для знаходження суми N цілих додатних парних чисел введених з клавіатури.

3. Написати програму для знаходження кількість додатних чисел введених з клавіатури поки не введено нуль.

4. Написати програму, яка обчислює суму парних чисел введених з клавіатури поки не введено нуль.

5. Написати програму, яка обчислює суму перших N елементів ряду 1+1/2+1/3+.... (кількість вводиться під час роботи програми).

6. Написати програму, яка обчислює суму перших N елементів ряду 1/2+1/4+1/6... (кількість вводиться під час роботи програми).

7. Написати програму, яка обчислює суму перших N елементів ряду 1+1/3+1/5+.... (кількість вводиться під час роботи програми).

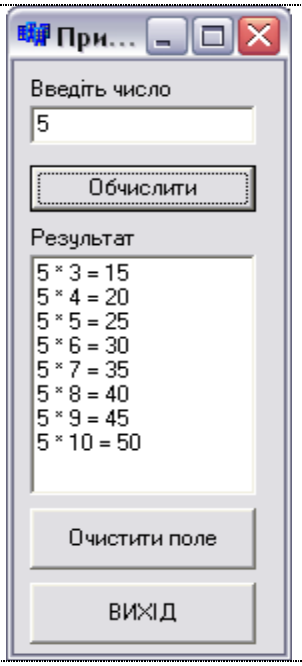
Завдання 6. Виведіть таблицю множення на довільне число.

📖 *Методичні вказівки:*

```
{
   int r;
   int x = (Convert::ToInt32(textBox1->Text));
   for(int i=1;i<=10;i++)
   {
      r=i*x;
     textBox2->AppendText(x+"*"+i+"="+r+ "\r\n");
    }
}
```

Очистка поля **TextBox** здійснюється методом **Clear()**
textBox2->Clear();

"\r\n" – перейти на новий рядок та повернути коретку



1. Написати програму, яка виводить значення кубів чисел, що належать проміжку [1, 9] з кроком 2. Оформити виведення у вигляді: 1^3=1;3^3=9;5^3=125 і т.д.
2. Написати програму, яка виводить таблицю значень функції y = 3x+2 в діапазоні (-2;2) з кроком 0,5.

3. Написати програму, яка виводить на екран таблицю вартості яблук в діапазоні від 100 г до 1 кг. з кроком 100г.(вартість вводиться під час роботи програми).
4. Написати програму, яка заміняє всі парні елементи масиву на „0”.

Завдання 7. Обробка масивів

>> Перейти до теми «Масиви у мові С++»

📖 *Методичні вказівки:*

***Приклад**: Вивести у текстовому полі згенерований двовимірний масив.*

```
//textBox1->Multiline=true
 int i,j, a[5][5];
 Random^r=gcnew Random;
for(i=0;i<5;i++)
 {
  for(j=0;j<5;j++)
  {
   a[i][j]=r->Next(10);
   textBox1->AppendText(a[i][j]+"  ");
  }
  textBox1->AppendText("\r\n");
 }
```

В динамічні памяті створено об’єкт типу **Random**, який володіє функціями генерації випадкових чисел.

NextDouble() – генерує випадкові числа в діапазоні 0.0 – 1.0

a[i][j]=100.0*r->NextDouble() - генерує випадкові дійсні числа в діапазоні 0-100

***Приклад**: Підрахувати кількість нульових елементів двовимірного масиву та замінити всі від’ємні елементи нулями. Введення даних з клавіатури реалізується у компонент* **dataGridView1**. *Результат заміни у компонент* **dataGridView2.**

Після розташування на формі компонентів необхідно встановити для них такі властивості:

dataGridView1	AllowUserToAddRows	true
dataGridView1	Columns /Width	Column1/50 Column2/50 Column3/50 Column4/50 Column5/50
dataGridView1	ColomnHeadersVisible	false
dataGridView1	RowHeadersVisible	false
dataGridView2	AllowUserToAddRows	false
dataGridView2	AllowUserToDeleteRows	false
dataGridView2	ReadOnly	true
dataGridView2	Columns /Width	Column1/50 Column2/50 Column3/50 Column4/50 Column5/50
dataGridView2	ColomnHeadersVisible	false
dataGridView2	RowHeadersVisible	false

```
int i,j, a[3][3], k=0;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
{
    a[i][j]= Convert::ToInt32(dataGridView1->Rows[i]->Cells[j]->Value);
    if(a[i][j]==0)k++;
}
MessageBox::Show("кількість елементів= 0  = "+k);
// заміна та виведення елементів масиву
for(i=0;i<3;i++)
{
    dataGridView2->Rows->Add();
    for(j=0;j<3;j++)
    {
       if (a[i][j]<0)a[i][j]=0;
       dataGridView2->Rows[i]->Cells[j]->Value =Convert::ToString(a[i][j]);
    }
}
```

****

1. Написати програму, яка заміняє всі парні елементи масиву на „0”.

2. Написати програму, яка визначає, скільки раз у масиві зустрічається введене з клавіатури число.

3. Написати програму, яка визначає суму елементів, які мають непарні порядкові номери.

4. Написати програму, яка визначає кількість учнів у класі чий зріст вище середнього.

5. Написати програму, яка визначає мінімальний елемент масиву та його порядковий номер.

Завдання 8. (Обробка рядків)

>> Перейти до теми «Стандартний клас String»

📖 *Методичні вказівки:*

***Приклад**: Замінити кожен 2-й символ рядка на «,» підрахуйте кількість замін*

```
String^ s=textBox1->Text;
String^s1=s1+=s[0];
int k=0;
for(int i=1;i<s->Length;i++)
{
   if(i%2!=0) s1+=",";
   else s1+=s[i];
   if (s1[i]==',')  k++;
}
label1->Text=s1;
MessageBox::Show("кількість замін = "+k);
```

***Приклад**: Замінити всі літери «а» на прописні «А»*

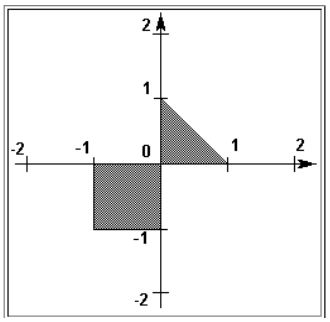
```
String^ s=textBox1->Text;
String^s1=" ";
for(int i=0; i<s->Length; i++)
if(s[i]=='a') s1+="A";
else s1+=s[i];
label1->Text=s1;
```

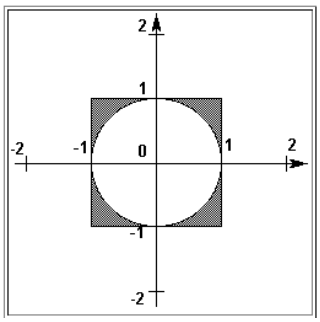
****

1. Написати програму, яка підраховує кількість ком у рядку.
2. Написати програму, яка заміняє всі літера „а” на „1”.
3. Написати програму, яка підраховує кількість слів у рядку.
4. Написати програму, яка визначає, скільки раз у рядку зустрічається літера введена з клавіатури.
5. Написати програму, яка заміняє кожен другий символ рядку на „1”.

Додаткові завдання

1. Написати програму яка визначає, чи потрапляє точка з координатами (x, y) у зафарбовану область.





2. Дано три різних дійсних числа. Визначити, яке з них найбільше, найменше та середнє.

3. Відомі рік, номер місяця та день народження двох людей. Визначити хто з них найстарший.

4. Дано три дійсні додатні числа. Якщо існує трикутник з такими сторонами, то визначити його вид (рівносторонній, рівнобедрений, різносторонній)

Лабораторна робота №12
Розробка додатків з використанням елементів управління.

Теоретичні відомості

Радіокнопки утворюють групи взаємозв'язаних індикаторів, з яких звичайно може бути вибраний тільки один. Вони використовуються для вибору користувачем однієї з декількох взаємовиключних альтернатив

Окрема радіокнопка **RadioButton** особливого сенсу не має. А радіокнопки мають сенс, коли вони взаємодіють один з одним в групі.

Компонент **GroupBox** (*вкладка «**Контейнери**» панелі елементів*) – є контейнером, об'єднуючим групу зв'язаних елементів управління, таких, як радіокнопки **RadioButton** або індикатори **Checkbox** і т.д.

Індикатори з прапорцем **CheckBox** використовуються у додатках для того, щоб вмикати та вимикати опції, або для індикації стану.

Основні властивості компоненту **CheckBox**

Checked – властивість, яка визначає чи включено прапорець.

CheckState – встановлює 3вида станів компоненту: **Checked** - прапорець увімкнуто; **Uncheked** - прапорець вимкнуто; **Indeterminate** - невизначений стан (сірий колір прапорця).

CheckAlign – властивість, що дозволяє відкривати розгортаючийся список, для вибору розташування прапорця у полі компоненту (9 позицій).

AutoCheck - властивість визначає, чи повинно автоматично змінюватися стан прапорця в результаті клацання на його зображенні. За умовчанням значення рівне True.

Стан прапорця змінюється в результаті клацання на його зображенні (якщо значення властивості **AutoCheck = True**). При цьому виникає подія **CheckedChanged**, а потім подія **Click**. Якщо значення властивості **AutoCheck = False**, то в результаті клацання на прапорці виникає подія **Click**, а

потім, якщо процедура обробки цієї події змінить стан кнопки, виникає подія **CheckedChanged**.

Завдання для самостійного виконання

Завдання 1. Розробіть додаток використовуючи елементи управління

1.Розробити програму обчислення вартості розмови по мобільному телефону використовуючи компонент **RadioButton**, за допомогою якого визначається вартість розмови (в середині мережі, інший оператор).

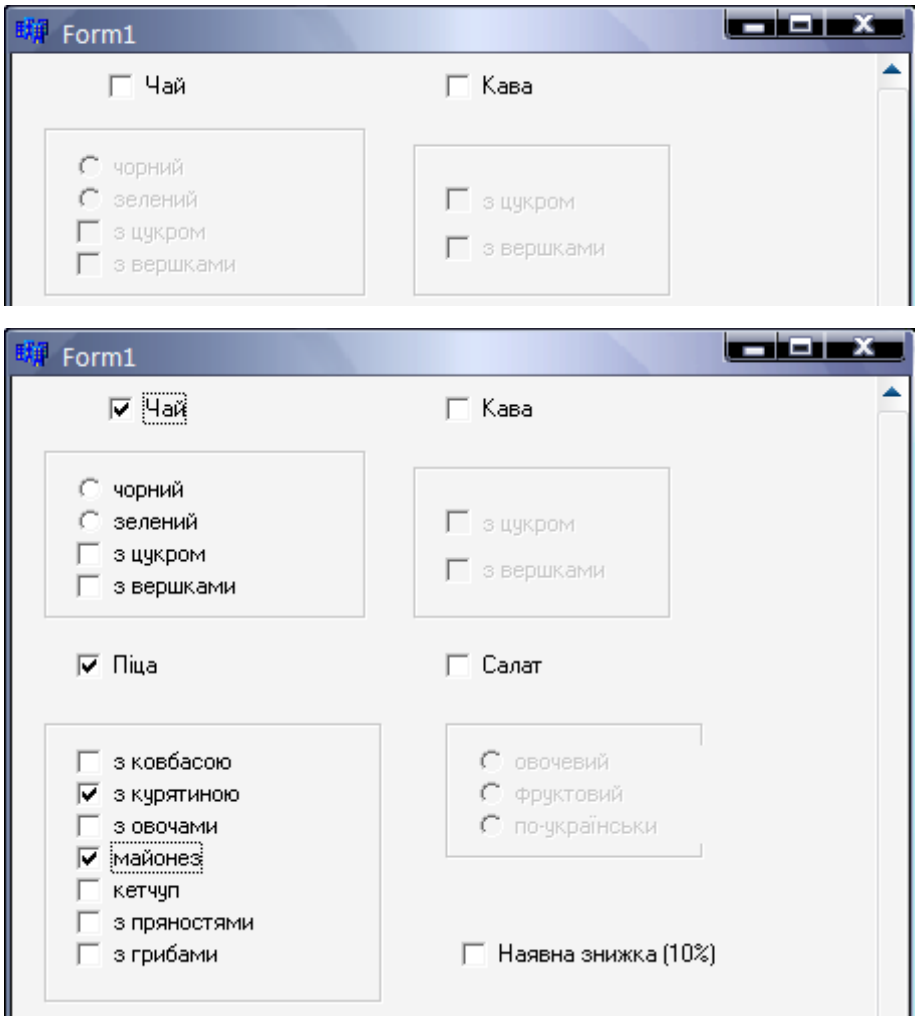
2. Написати програму обчислення вартості вікна за введеною висотою та шириною. Програма повинна містити вибір опції (однокамерний чи двокамерний пакет) та додаткова можливість замовити підвіконня (**CheckBox** і **RadioButton**).

📖 *Методичні вказівки*

```
// Використання компоненту radioButton. Подія CheckedChanged
private: System::Void radioButton1_CheckedChanged(...)
{
   int a,b,s;
   a=Convert::ToInt32(textBox1->Text);
   b=Convert::ToInt32(textBox2->Text);
   if (radioButton1->Checked)
  {
     s=a+b;
     label1->Text=Convert::ToString(s);
  }
}

// Використання компоненту checkBox. Подія CheckedChanged
private: System::Void checkBox1_CheckedChanged(        )
 {
  if (checkBox1->Checked)
  label2->Text="прапорець встановлено";
  else
  label2->Text="прапорець знято";
 }
```

Завдання 2. Розробити програму обчислення вартості замовлення в кафе використовуючи компонент **CheckBox** і **RadioButton**. Одна зі страв повинна бути складною (*наповнення якої обирає клієнт*), компоненти якої не активні (або не видимі) поки не вибрано назву страви. Врахувати можливість відмови від замовлення на певну страву. Враховувати можливість знижки.

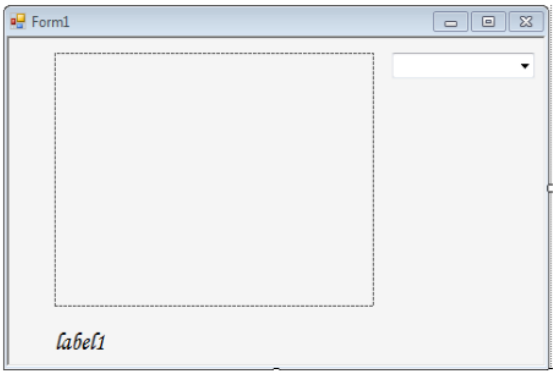
У програмі можна передбачити текстове поле для введеня кількості страв у замовленні.



Лабораторна робота №13
Розробка інтерфейсу користувача прикладних додатків

Завдання 1. Завантаження зображення в **PictureBox** за допомогою **ComboBox**.

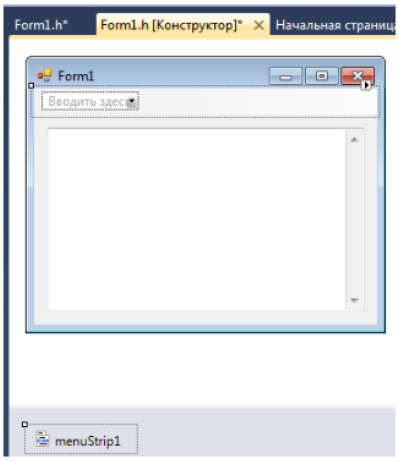
При виборі одного із слів в списку повинна з'являтися картинка, а в "**label**" її назва. Для того що б занести в "**comboBox**" деякий список, потрібно знайти в панелі властивостей - властивість "**Items**" і написати список назв зображень (*кожна назва з нового рядка*).



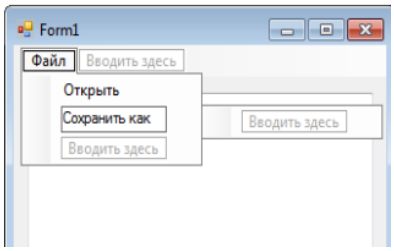

```
private: System::Void Form1_Load(…)
{
 this->Text = "Фотогалерея";
 label1->Text = "";
 comboBox1->Text = "Список";
 }

private: System::Void comboBox1_SelectedIndexChanged(…)
{
    switch (comboBox1->SelectedIndex)
    {
        case 0: pictureBox1->Image=Image::FromFile("d:\\7.0.png");
        label1->Text = "Літо"; break;
        case 1: pictureBox1->Image=Image::FromFile("d:\\7.1.png");
        label1->Text = "Сонце"; break;
        case 2: pictureBox1->Image=Image::FromFile("d:\\7.2.png");
        label1->Text = "Море"; break;
        case 3: pictureBox1->Image=Image::FromFile("d:\\7.3.png");
        label1->Text = "Пляж"; break;
    }
}
```

Завдання 2. Елемент **MenuStrip** і властивість **Anchor в MVS C++**

Елемент «**MenuStrip**» – це головне меню. Властивість «**Anchor**» - це властивість, при якій визначається до яких сторін форми буде прив'язаний елемент, якщо до правої, то при збільшенні або зменшенні розміру форми - права сторона «**textBox**», так само буде збільшуватися або зменшуватися. Якщо поставлена кнопка «**button**», то вона переміщатиметься за стороною форми, до якої прив'язана.

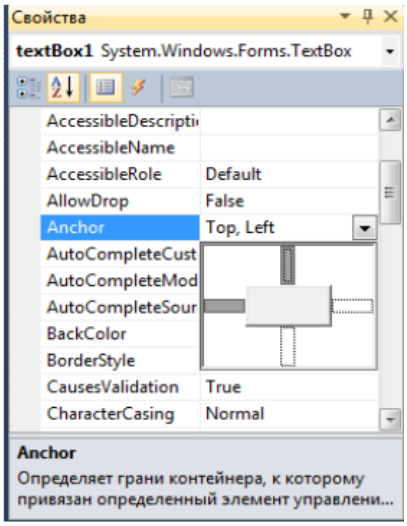


Перенесіть на форму елемент «**MenuStrip**» і елемент «**textBox**». У елементу «**textBox**» застосуйте властивості «**Multeline**» і «**ScrollBars -> Vertical**».

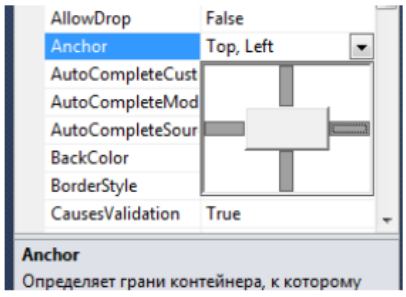


Далі натисніть на текстове поле «**Вводить здесь**» і напишіть – «**Файл**»:

Після цього в нижньому, новому текстовому полі, що з'явилося, напишіть «**Відкрити**», після чого з'явитися ще одне, в якому напишіть «**Зберегти як**»:



Тепер займемося властивістю «**Achor**» - для цього натисніть на «**textBox**». Після чого на панелі властивостей, розташованій зліва, виберіть властивість «**Achor**»:



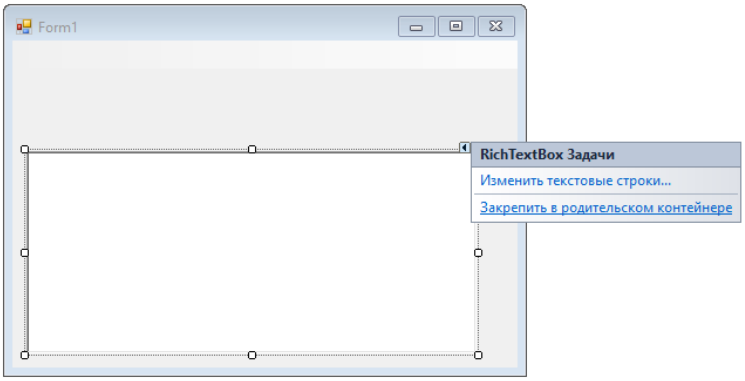
Компонент «**textBox**» прив'язаний до лівої і верхньої частини форми. Тепер виділите так само праву і нижню частину, після чого розмір елементу «**textBox**» змінюватиметься пропорційно розміру форми:

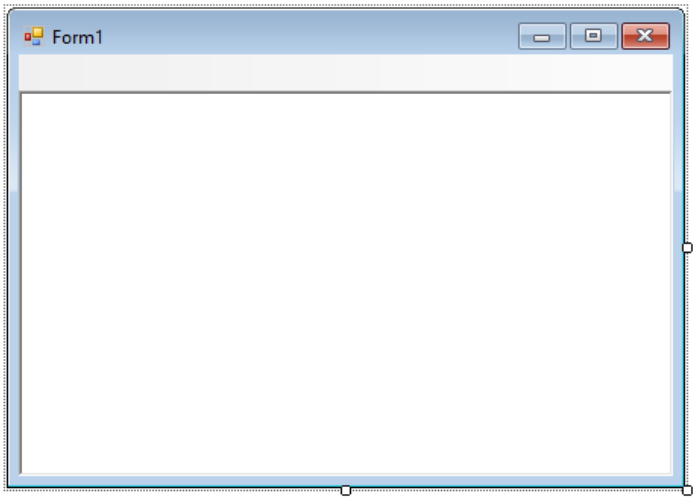
Запустіть програму і перевірте змінені властивості:

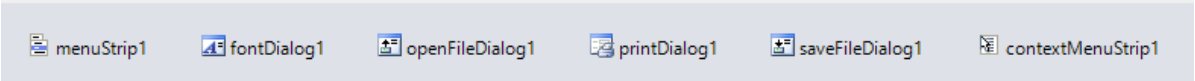
Завдання для самостійного виконання

Завдання 3. Створіть програму «Текстовий редактор». Проаналізуйте представлений програмний код.

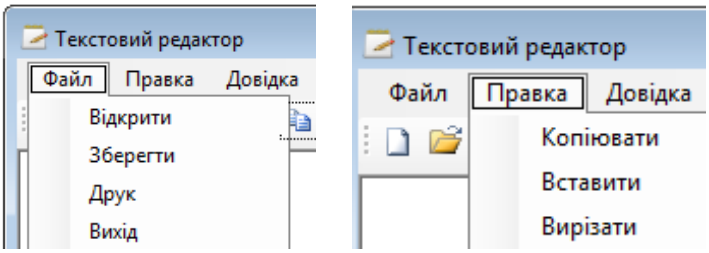
1. Додайте на нову форму компоненти

OpenFileDialog	openFileDialog не візуальний компонент представляє собою діалогове вікно «**Открыть**».
SaveFileDialog	Компонент для відображення діалогового вікна «**Сохранить**» та збереження файлу. Але компонент дає лише шлях до місця розташування файлу, написання коду, який відповідає за збереження даних відводиться програмісту.
FontDialog	Компонент **FontDialog** – виклик вікна налаштування параметрів шрифту. Обрані параметри присвоюються властивості компоненту **Font**.
PrintDialog	Компонент **PrintDialog** – відкриває діалогове вікно налаштування параметрів друку
RichTextBox	 Компонент необхідно закріпити у батьківському контейнері
MenuStrip	Головне меню
ContextMenuStrip	Контекстне меню (застосувати до компоненту richTextBox)





2. Використовуючи компонент **MenuStrip** створіть головне меню програми.



3. Функція збереження тексту створюється програмістом.

```
void save()
{
    auto read = gcnew IO::StreamWriter(saveFileDialog1->FileName, false,
    System::Text::Encoding::GetEncoding(1251));
    read->Write(richTextBox1->Text);
    read->Close();
    richTextBox1->Modified = false;
}
```

4. Пункт меню «**Зберегти**»

```
saveFileDialog1->FileName = openFileDialog1->FileName;
saveFileDialog1->Filter = "txt files (*.txt)|*.txt|rtf files (*.rtf)|*.rtf|All files (*.*)|*.*";
if (saveFileDialog1->ShowDialog() == Windows::Forms::DialogResult::OK) save();
```

5. Пункт меню «**Відкрити**»

```
openFileDialog1->Filter = "txt files (*.txt)|*.txt|rtf files (*.rtf)|*.rtf|All files (*.*)|*.*";
if( openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK){
System::Text::Encoding^kod=System::Text::Encoding::GetEncoding(1251);
auto read= gcnew IO::StreamReader(openFileDialog1->FileName,kod);
richTextBox1->Text=read->ReadToEnd();}
this->Text+=(openFileDialog1->FileName);
```

6. Пункт меню «**Друк**»

```
printDialog1->ShowDialog();
```

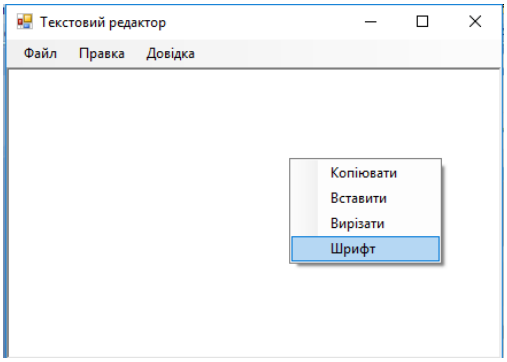
7. Пункти меню «**Копіювати**», «**Вирізати**», «**Вставити**».

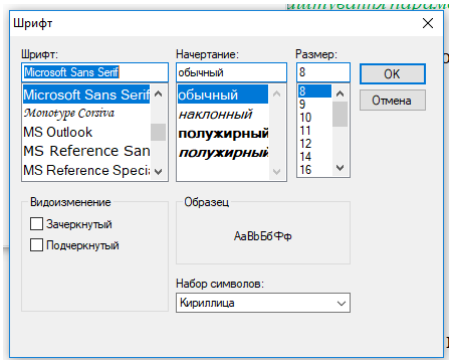
```
richTextBox1->Copy();
richTextBox1->Cut();
richTextBox1->Paste();
```

8. Додайте пункт меню «**Довідка**» з інформацією про розробника програмного продукту.

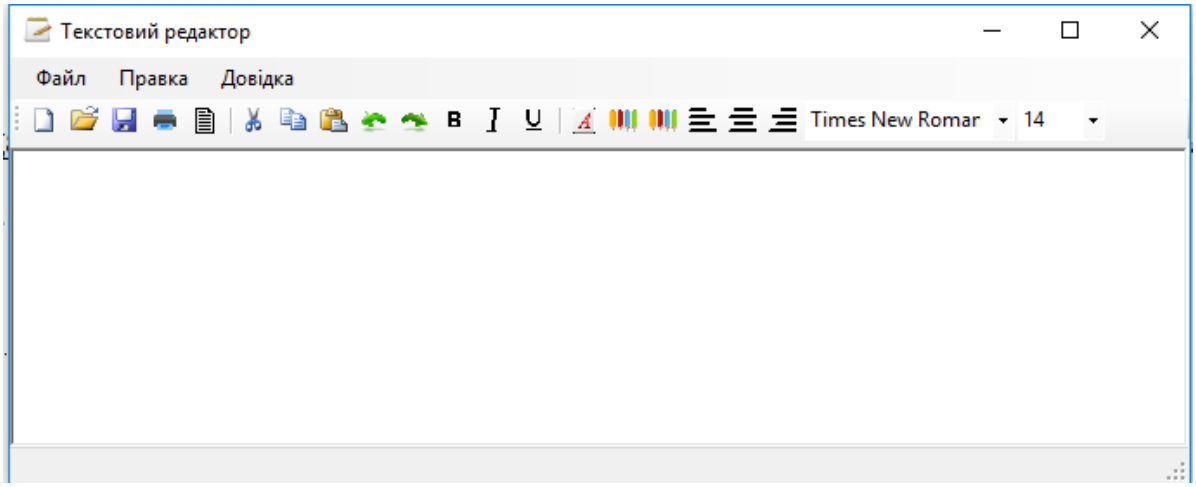
9. Додайте контекстне меню. Пункти меню «**Копіювати**», «**Вирізати**», «**Вставити**», «**Шрифт**».

```
//відкриває діалогове вікно налаштування параметрів шрифта
this->fontDialog1->ShowDialog();
richTextBox1->SelectionFont=fontDialog1->Font;
```





10. Додаткове завдання. Додати панель інструментів зі стандартним набором кнопок. ToolStrip



Лабораторна робота №14
Особливості роботи з графічними примітивами

Теоретичні відомості

Відображення графіки забезпечує компонент `PictureBox`. Графічна поверхня компоненту `PictureBox` є об'єктом `Graphics`, методи якого і забезпечують виведення графіки.

Методи малювання графічних примітивів використовують *олівці* і *пензлі*. Олівець (об'єкт `Pen`) визначає вид лінії, пензель (об'єкт `Brush`) - вид заливки області.

Побудова графічних примітивів на графічній поверхні (`Graphics`) виконують відповідні методи.

Метод	Дія
DrawLine(Pen, x1, y1, x2, y2), DrawLine(Pen, p1, p2)	Пряма
DrawRectangle(Pen, x, y, w, h)	Прямокутник
FillRectangle(Brush, x, y, w, h)	Зафарбований прямокуник
DrawEllipse(Pen, x, y, w, h)	Еліпс
FillEllipse(Brush, x, y, w, h)	Зафарбований еліпс
DrawPolygon(Pen, P)	Багатокутник
FillPolygon(Brush, P)	Зафарбований багатокутник
DrawString(str, Font, Brush, x, y)	Текст

Приклад. Зображення прапора України.

```
private: System::Void pictureBox1_Paint(…)
{
   int x =10, y=10; // лівий верхній кут
   int w=100, h=30; // ширина та висота смуги
   e->Graphics->FillRectangle(Brushes::Blue,x,y,w,h);
   y += h;
   e->Graphics->FillRectangle(Brushes::Yellow,x,y,w,h);
   // Підпис. Використовується шрифт, заданий властивістю Font форми.
   e->Graphics->DrawString("УКРАЇНА",this->Font, Brushes::Black, 10, 80);
}
```

Під анімацією звичайно розуміється рухомий малюнок Підбором часу між виведенням і видаленням малюнка, а також відстані між старим і новим положенням малюнка (крок переміщення), можна досягти того, що у спостерігача складатиметься враження, що малюнок рівномірно рухається по екрану.

Компонент **Timer**, який використовується для генерації послідовності подій, функція обробки яких забезпечує виведення і видалення малюнка.

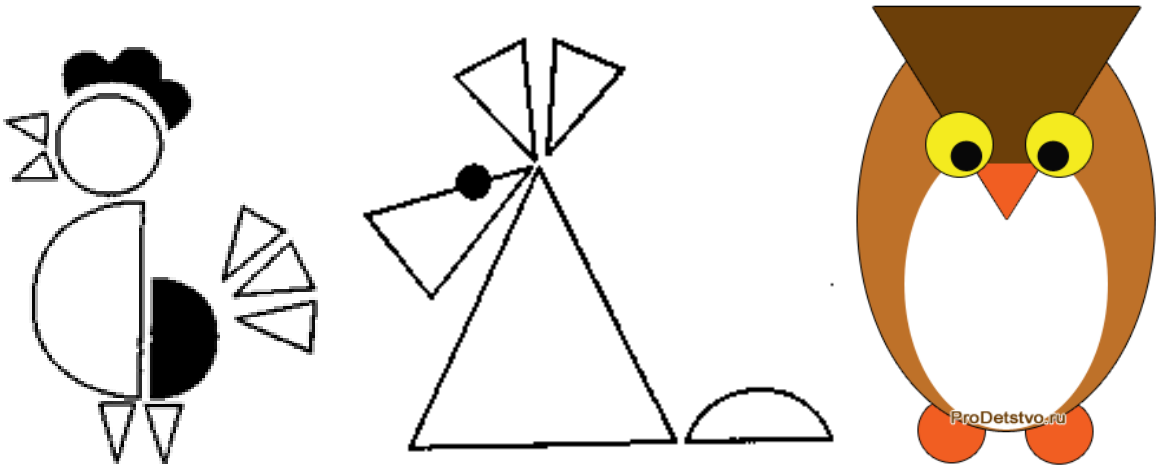
Компонент **Timer** генерує подію **Tick**. Період виникнення події **Tick** вимірюється в мілісекундах і визначається значенням властивості **Interval**. Слід звернути увагу на властивість **Enabled**. Воно дає можливість програмі "запустити" або "зупинити" таймер. Якщо значення властивості **Enabled** = **False**, та подія **Tick** не виникає.

Приклад програми, яка імітує рух геометричної фігури (червоий квадрат)

```
namespace anima {
…
using namespace System::Drawing;
int x=50;  // оголошення глобальної змінної
…
private: System::Void pictureBox1_Paint (…)
{
   e->Graphics->FillRectangle(Brushes::Red,x,70,50,50);
}

private: System::Void timer1_Tick(…)
{
  x+=3;
  pictureBox1->Refresh();
}
```

Завдання для самостійного виконання

Завдання 1. Побудуйте малюнки.



📖 *Методичні вказівки:*

Метод DrawPolygon креслить багатокутник (контур).

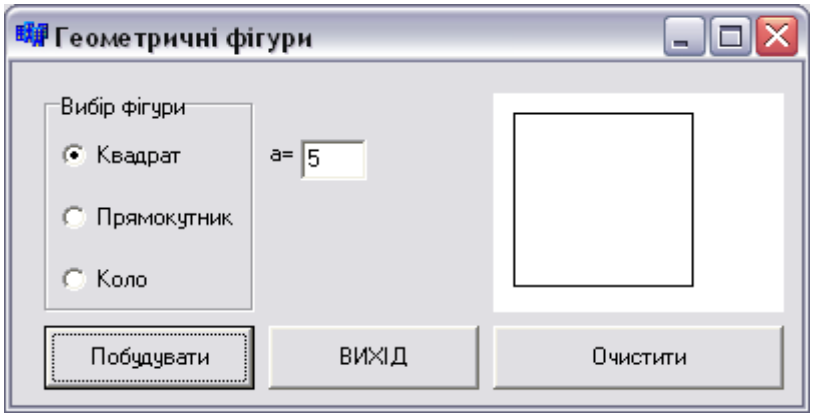
DrawPolygon(aPen, p)

Зафарбований багатокутник малює метод FillPolygon.

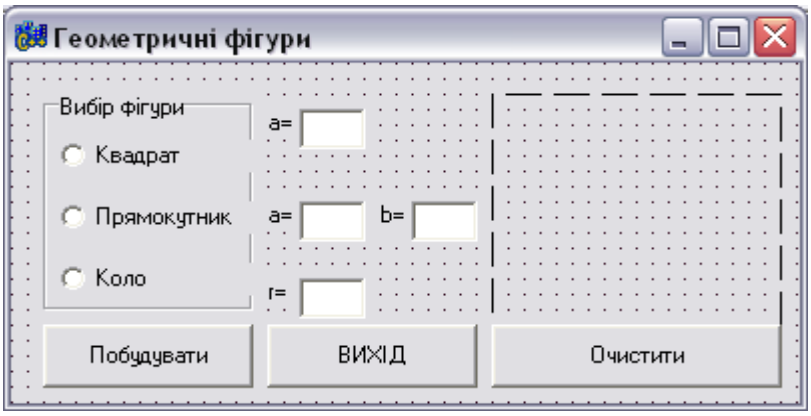
FillPolygon(aBrush, p)

```
array<Point>^ p;
p = gcnew array<Point>(5);
p[0].X = 10; p[0].Y =30;
p[1].X = 10; p[1].Y =10;
p[2].X = 30; p[2].Y =20;
p[3].X = 50; p[3].Y =10;
p[4].X = 50; p[4].Y =30;
e->Graphics->FillPolygon(Brushes::Gold, p);
e->Graphics->DrawPolygon(Pens::Black,p);
```

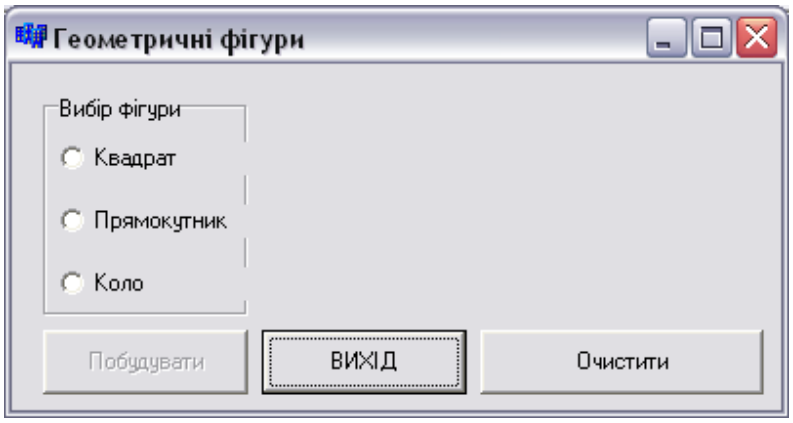
Завдання 2. Написати програму для демонстрації побудови геометричних фігур. Фігура виводиться у компоненті **PictureBox**.



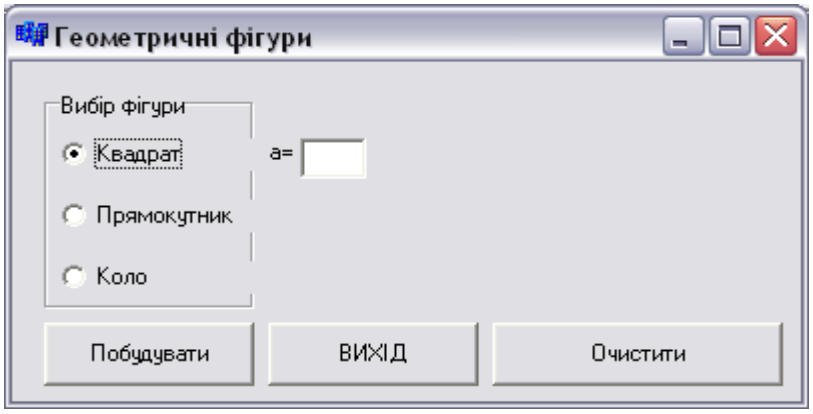
📖 *Методичні вказівки:*



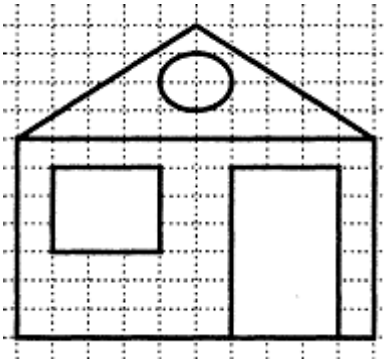
1. При завантаженні кнопка «**Побудувати»** – неактивна (**Enabled = False**) а поля для введення параметрів – невидимі (**Visible = False**).

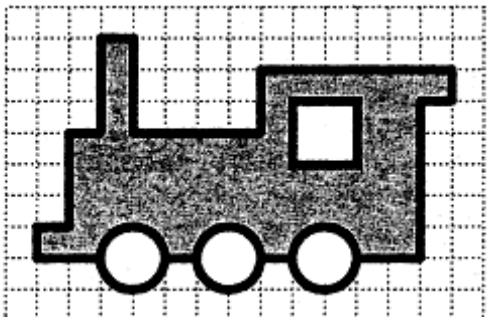


2. Після вибору геометричної фігури стають видимими поля для введення відповідних параметрів.



Завдання 3. Побудуйте малюнки методом базової точки.





🕮 *Методичні вказівки:*

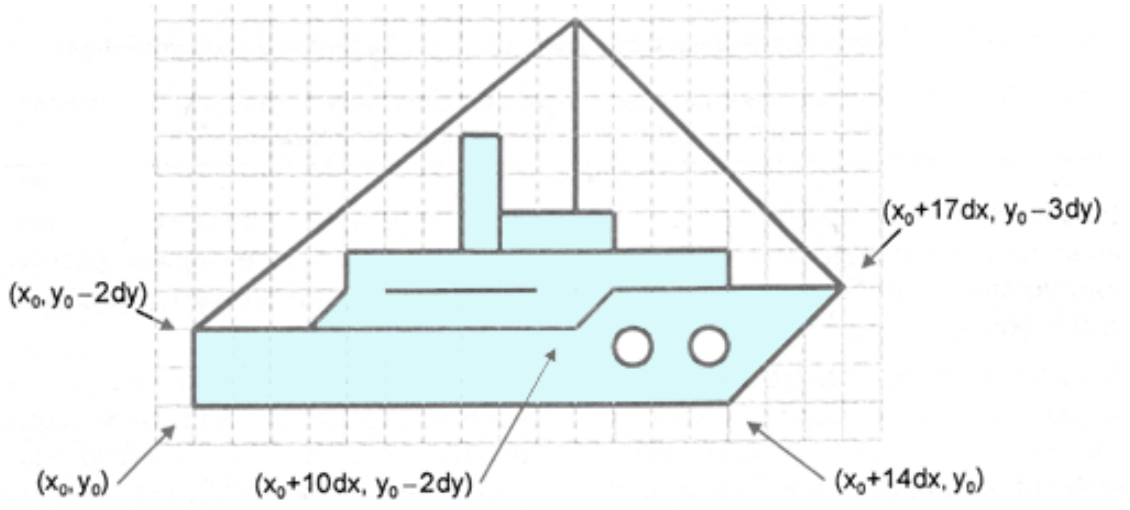
При програмуванні складних зображень, що складаються з безлічі елементів, використовується метод, який називається методом базової точки. Суть цього методу полягає в наступному:

1. Вибирається деяка точка зображення, яка береться за базову.

2. Координати інших точок відлічуються від базової точки.

3. Якщо координати точок зображення відлічувати від базової у відносних одиницях, а не у пікселах, то забезпечується можливість масштабування зображення.

Наприклад зображення кораблика. Базовою - є точка з координатами (X_0 Y_0). Координати інших точок відлічуються саме від цієї точки.

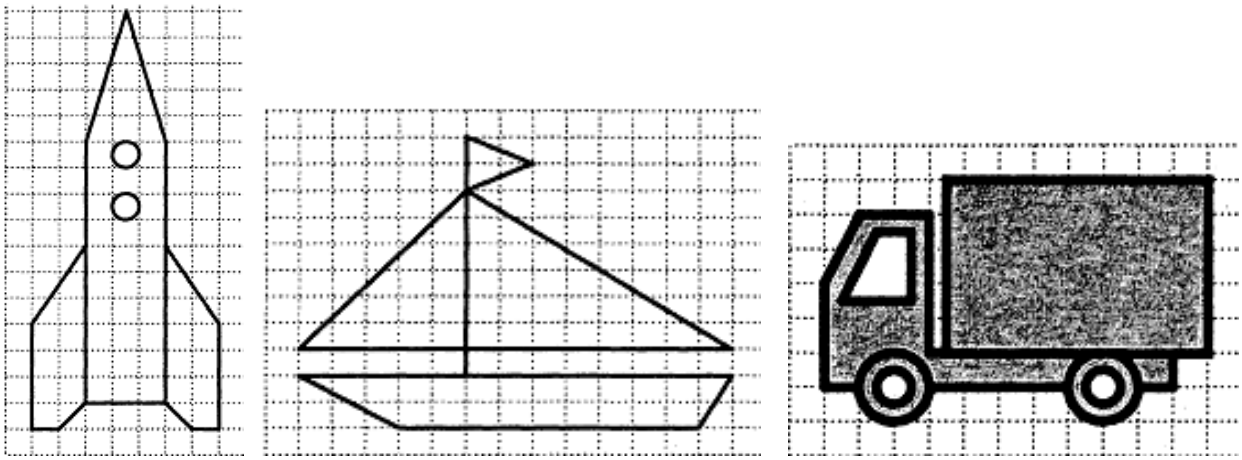


dx, dy – маштабування зображення (задається у пікселах.)

Приклад програми пободови фрагменту кораблика методом базової точки

```
int X = 70, Y = 100; //базова точка X0 Y0
int dx=10,dy=10; // крок сітки
array<Point>^ p1;
array<Point>^ p2;
p1 = gcnew array<Point>(7);
p2 = gcnew array<Point>(8);
// координати точок корпусу
p1[0].X = X; p1[0].Y = Y;
p1[1].X = X; p1[1].Y = Y-2*dy;
p1[2].X = X+10*dx; p1[2].Y = Y-2*dy;
p1[3].X = X+11*dx; p1[3].Y = Y-3*dy;
p1[4].X = X+17*dx; p1[4].Y =Y-3*dy;
p1[5].X = X+14*dx; p1[5].Y =Y;
p1[6].X = X; p1[6].Y =Y;
e->Graphics->DrawPolygon(Pens::Black, p1);
// координати точок надбудови
p2[0].X = X+3*dx; p2[0].Y = Y-2*dy;
p2[1].X = X+4*dx; p2[1].Y = Y-3*dy;
p2[2].X = X+4*dx; p2[2].Y = Y-4*dy;
p2[3].X = X+13*dx; p2[3].Y = Y-4*dy;
p2[4].X = X+13*dx; p2[4].Y = Y-3*dy;
p2[5].X = X+11*dx; p2[5].Y = Y-3*dy;
p2[6].X = X+10*dx; p2[6].Y = Y-2*dy;
p2[7].X = X+3*dx; p2[7] .Y = Y-2*dy;
e->Graphics->DrawPolygon(Pens::Black, p2);
```

Завдання 4. Створіть анімацію використавши кольорові малюнки створені методом базової точки (*один на вибір*).



СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вирт Н. Алгоритмы и структуры данных / Н.Вирт -- М.: Мир,1989. – 360 с.

2. Войтенко В.В. С/С++: Теорія та практика. Навчально-методичний посібник / В.В. Войтенко, А.В. Морозов – Житомир:ЖДТУ, 2004. – 324с.

3. Глушаков С.В. Практикум по С++/ С.В. Глушаков, А.В. Коваль, С.В. Смиронов,.– Харьков: Фолио, 2006. – 525 с.

4. Златопольский Д.М. Сборник задач по програмированию /Д.М. Златопольский– 2-е изд., перераб. и доп. – СПб.:БХВ – Петербург, 2007. – 240с.

5. Кнут Д. Искусство программирования. Т. 2. Получисленные алгоритмы / Д.Кнут. – М.: Вильямс, 2007. — 832 с.

6. Кнут Д. Искусство программирования. Т. 3. Сортировка и поиск / Д.Кнут. – М.: Вильямс, 2007. — 824с.

7. Кривцова О.П. Програмування мовою С++. Розробка консольних додатків у середовищі Microsoft Visual C++Express [Електронний ресурс] : навч. посіб. / О.П. Кривцова. – Полтава: ПНПУ імені В.Г. Короленка, 2017. – 99с.

8. Культин Н. Б. С/С++ в примерах и задачах /Н.Б. Культин - СПб.: БХВ-Петербург, 2001. - 288с.

9. Пахомов Б.И. С.С++ и VC++2010 для начинающих /Б.И. Пахомов.- СПб.: БХВ-Петербург, 2011. - 736с.

10. Хортон А. VC++2010:полный курс /Айвор Хортон. – М.: ООО «И.Д. Вильямс», 2011. – 1216с.