Лєхан С.А.

Інформатика

Мова програмування С++

Спецкурс

10-12 класи

Навчальний посібник

Шепетівка
«Аспект»
2007

ББК 22.18
Л-52

Технічний редактор, вчитель інформатики:
Пилипчук О.П. – с. Гаврилівка, Хмельницька обл.

Рецензенти, вчителі інформатики:
Буз І.А. – ОІППО, м. Одеса;
Ніколаєнко М.С. – ОІППО, м. Суми;
Прокоп Ю.В. – м. Одеса;
Ребрина В.А. – ОІППО, м. Хмельницький;
Ткачук Д.В. – м. Одеса.

Коректор Слободян В.В.

Л-52 Інформатика. Мова програмування С++. Спецкурс. 10-12 класи. Навчальний посібник / Лєхан С.А. – Шепетівка, «Аспект», 2007 – 160 с.

ISBN 978-966-2017-11-3

Рекомендується як навчальний посібник для 10-12 класів загальноосвітніх навчальних закладів різних профілів з 12-річним терміном навчання. Просто і доступно описані основні відомості про алгоритми та базові алгоритмічні структури, основи програмування мовою С++. До кожної теми додаються питання для самоконтролю і вправи для закріплення набутих знань. Відповідає вимогам діючих програм з інформатики та 12-бальній системі оцінювання знань учнів.

Для широкого кола читачів, а також для учнів, студентів, вчителів та викладачів загальноосвітніх навчальних закладів

ББК 22.18

ISBN 978-966-2017-11-3

Передмова

Рекомендується для учнів 10-12 класів загальноосвітніх навчальних закладів з 12-річним терміном навчання. Тривалість курсу – 45 годин.

Навчальний посібник відповідає вимогам діючих програм МОН України з інформатики та 12-бальній шкалі оцінювання знань учнів.

Назва, нумерація, зміст розділів і уроків відповідають рекомендованому поурочному плануванню і календарному плану.

Навчальний посібник орієнтовано на практичне використання комп'ютерів, починаючи з першого уроку. Кожна тема дозована на один урок, має структуру, що відповідає санітарним нормам: теоретичний матеріал без використання комп'ютерів – 15 хв.; закріплення нового матеріалу, комп'ютерне тестування домашнього завдання і виконання вправи – 30 хв.

Для практичної роботи або тематичного оцінювання відводиться окремий урок.

Для виконання насиченого поурочного планування діючих програм доводиться шукати засоби інтенсифікації проведення уроку: наприклад, до дошки одного учня не викликати і з місця не питати, не витрачати час на диктування конспектів, давати завдання додому.

Для роботи одночасно з усіма учнями рекомендується використовувати тренажери та комп'ютерне тестування за допомогою контрольно-діагностичної системи, виконувати вправи та практичні роботи.

Тест служить для самоконтролю набутих на уроці теоретичних знань. Вправа служить для формування і закріплення набутих на уроці практичних навичок і може служити для поточного оцінювання навчальних досягнень учнів на уроці.

Практичні роботи розроблені згідно з вимогами діючих програми для формування і закріплення практичних навичок, здобутих протягом кількох уроків, і служать основним засобом оцінювання знань та навичок учнів. Тематична робота служить для оцінювання знань та навичок учнів з однієї або кількох тем.

Майже кожна вправа має 3 дії, кожна практична або тематична робота мають від 3-х до 6-ти послідовних дій, виконання яких оцінюється від 2 до 4 балів, щоб оцінювати роботу за принципом «скільки дій зробив – таку оцінку й заробив».

Автор висловлює вдячність технічному редактору Олександру Павловичу Пилипчуку за професійну допомогу, а також учасникам семінару вчителів інформатики Хмельницької області, що проходив 18-19 квітня 2007 року, зауваження та пропозиціїяких допомогли покращити текст посібника. За корисні консультації з окремих питань автор дякує Кухару А.В., викладачу Хмельницького національного університету.

Зміст

1. Лінійні алгоритми

1.1. Вступ до мови програмування С++

Програмування: основні поняття

Програмування *– це процес визначення послідовності інструкцій, які повинен виконати комп'ютер для розв'язання певного завдання. Для запису цих інструкцій використовують мову програмування, наприклад С++.*

Ви починаєте вивчати **мову програмування** С++ (читається «Сі-плюс-плюс»). С++ є сучасною, однією з найпоширеніших у середовищі професійних програмістів мов програмування.

Акредитований комітет стандартів, що діє під керівництвом Американського національного інституту стандартів (American National Standards Institute – ANSI), створив міжнародний стандарт для мови С++. Стандарт ANSI – це спроба гарантувати, що програма мовою С++ буде працювати однаково при переносі з одного комп'ютера на інший. Таким чином, програма повинна виконуватися без помилок на комп'ютерах різних архітектур, що працюють під керуванням операційних систем Windows, MacOS, Linux та інших.

Мова програмування С++, як і будь-яка інша мова, містить чотири основних елементи: символи, слова, словосполучення, речення. Але «слова» – елементарні конструкції, що мають самостійний зміст – називають **лексемами**. «Словосполучення», що задають правило обчислення деякого значення, називають **виразами**. «Речення», які задають закінчений опис деякої дії – **операторами**.

Для опису складної дії потрібна послідовність операторів, які можуть бути об'єднані у **складений оператор**, і він може розглядатися як один оператор. Оператори можуть бути виконуваними, тобто такими, що задають дії над даними, та невиконуваними, які служать для опису даних (їх часто називають операторами опису або просто описами).

Об'єднана єдиним алгоритмом сукупність описів й операторів утворює **програму**. Але перед створенням діючої програми має сенс на папері скласти схему, за якою буде виконуватися програма, аби не заплутатися при складанні алгоритму. Для цього будують блок-схему

алгоритму, на якій зображують елементарні (базові) конструкції, як, наприклад, просте слідування чи лінійний алгоритм.

***Блок-схема** алгоритму – графічне зображення алгоритму у вигляді організованої послідовності блоків.*

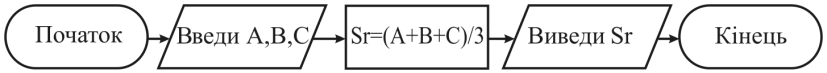
Графічне зображення базових алгоритмічних структур

Назва блоку	Опис дії
початок / кінець	Позначає початок та завершення алгоритму
Модифікація	Позначає дію, яку потрібно виконати. Це може бути вказівка зробити окрему дію (обчислити математичний вираз, накреслити прямокутник), так і послідовність дій (виконати обчислення за заданими формулами, намалювати малюнок).
Уведення / Виведення	Позначає уведення вхідної інформації і виведення проміжної і результуючої інформації.
Перевірка умови	Позначає перевірку значення логічного виразу деякої умови. Логічний вираз може набувати або значення ІСТИНА (1) або значення ХИБНІСТЬ (0).

Просте слідування команд означає, що дії повинні виконуватись послідовно одна за одною:

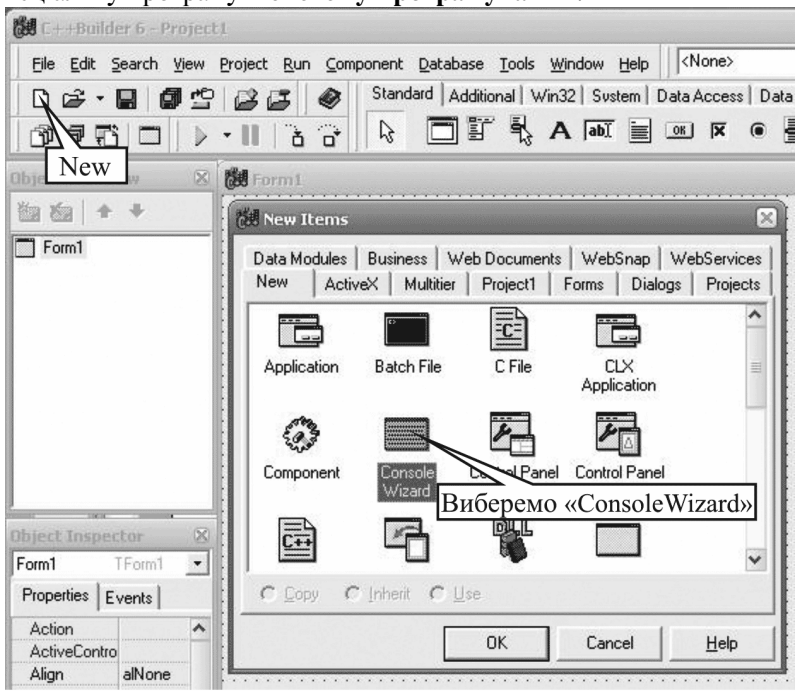
Лінійний алгоритм – алгоритм, який містить лише вказівки про безумовне виконання деяких операцій, без повторень або розгалужень (просте слідування).

Для прикладу наведемо алгоритм знаходження середнього арифметичного трьох чисел:



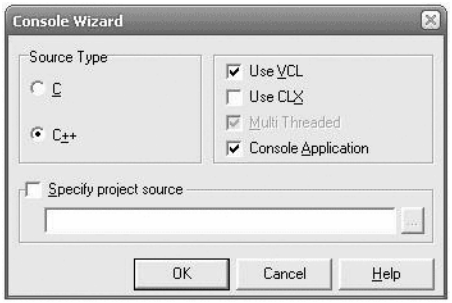
Перша програма

Для створення своєї першої програми ви повинні запустити спеціальну програму – **систему програмування**.

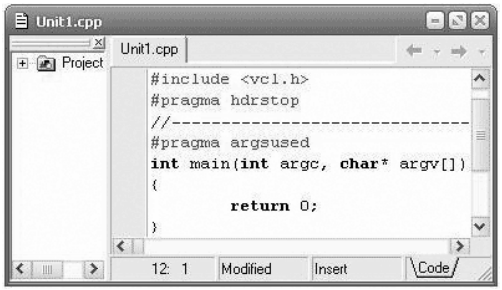


Після цього на екрані з'являться засоби для уведення, подальшої обробки і запуску вашої програми – **середовище програмування**. Однією з систем програмування є програма Borland C++Builder6. Для запуску програми натисніть: «Пуск» → «Програми» → «Borland C++Builder6» → «C++Builder6».

Для уведення вашої програми виберіть на панелі кнопку «New» (див. мал.), що означає створення нового проекту. Відкриється вікно «New items». Виберемо «ConsoleWizard». У діалоговому вікні «Console-Wizard» виберемо мову



програмування C++ та натиснемо «ОК». В результаті C++ Builder створить проект **консольної** програми, тобто такої, що виконуватиметься без використання графічного інтерфейсу Windows. На екрані з'явиться вікно редактора коду, в якому знаходиться шаблон програми – функція main, детальніше про яку ви дізнаєтесь пізніше. Це вікно далі можна використовувати для уведення та редагування програми.



Уведіть вашу першу програму, яка є лінійною програмою. Для цього допишіть виділені грубим шрифтом рядки, не заглиблюючись поки що в їхній зміст:

```
#include <vcl.h>                          //Програма 1.1
#include<iostream.h>
#include<conio.h>
#pragma hdrstop
#pragma argsused
int main()
{
 cout << "My first program!";
 getch();
 return 0;
}
```

Для того щоб програма була виконана і видала результат, необхідно перетворити її у послідовність команд процесора. Цю функцію виконує **компілятор** – програма-перекладач з мови програмування (в нашому випадку C++) на мову машинних кодів, зрозумілих процесору. Щоб відкомпілювати вашу програму і запустити на виконання, виберіть пункт меню «Run» чи натисніть клавішу **F9**. Якщо ви не припустилися синтаксичних помилок, про що вкаже компілятор у спеціальному вікні, результат роботи програми буде таким:

```
My first program!
```

Поекспериментуйте із цією програмою, щоразу компілюючи і запускаючи після внесення змін до тексту:


```
cout << "My first program!" << endl;
```

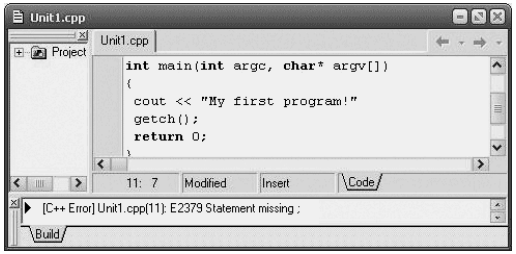
Потім так:

```
cout << "Programuvati prosto!";
cout << "Duge prosto!" << endl;
```

Ім'ям **endl** позначено спеціальний символ – кінець рядка (англ. **end l***ine* – кінець рядка; читається «енд-ел»). При виведенні він спричиняє перехід на новий рядок.

Програмуючи у Borland C++Builder6 уводьте текст в лапках латинськими буквами замість українських. Це пов'язано з тим, що в DOS і Windows для російських та українських літер використовують різні кодування (DOS – ASCII, а Windows – ANSI). Це призводить до того, що при виконанні програм замість очікуваного тексту на екран виводиться якась нісенітниця. Для коректного виведення літер кирилиці ми потім напишемо спеціальну функцію.

Спробуйте не брати текст у рядку з **cout** у лапки – компілятор виявить синтаксичну помилку, про що буде вказано у рядку з відповідним повідомленням (див. мал.). Клацніть по ньому мишкою й компілятор вкаже вам на рядок з помилкою.



На малюнку видно, що програміст забув поставити крапку з комою в кінці оператора. Компілятор знайшов помилку і вивів повідомлення: «**Statement missing;**», вказавши також ім'я файлу та номер рядка з помилкою (**Unit1.cpp(11)**).

При створенні програми мовою C++ ви повинні дотримуватися певних правил. Наприклад, треба брати текстові повідомлення в лапки

й ставити крапку з комою в кінці більшості операторів C++. Тільки виправивши всі помилки ви одержите результат виконання програми.

Оператори можна записувати в один рядок, але краще, коли кожен оператор буде починатися з нового рядка. Така програма буде зрозуміліша і в ній легше буде знайти помилки.

Розглянемо докладніше вашу першу програму. Вона складається із директив препроцесора, декількох операторів, функції і фігурних дужок, які називають групуючими символами:

```
#include <iostream.h>    //директиви препроцесора
#include <conio.h>
int main()                        //заголовок функції
{
cout<<"My first program" //продовження оператора
<<" C++!";        // у новому рядку не є помилкою
getch();
return 0;
}
```

Для кращого розуміння програм, у них корисно вставляти коментарі, перед якими ставляться дві косі риски («подвійний слеш» **//**) (див. текст програми). Компілятор ніяк не обробляє інформацію, розміщену після цих символів до кінця рядка. Якщо коментар займає кілька рядків, то на його початку ставлять символи **/***, а в кінці – ***/**.

Заголовкові файли (бібліотеки)

Препроцесором називається перша фаза компілятора. Інструкції препроцесора називаються **директивами** (розпорядженнями). Рядки, що їх містять, починаються зі знака **#**. Практично кожна програма на C++ містить директиви препроцесора **#include** (читати «паунд інклуд»). Вони необхідні для включення у програму певних файлів під час компіляції.

Наприклад, для виведення на екран тексту за допомогою об'єкта **cout** (читати «сі-аут») та операції перенаправлення вихідного потоку **<<** (будемо далі називати операцією виведення), необхідно підключити файл <*iostream.h*> (читати «ай-оустрім крапка ейч» від англійських: input-output streams – вхідні-вихідні потоки). Для застосування різних математичних функцій слід підключити файл <*math.h*>; для затримки зображення на екрані використовуємо функцію **getch()**, яка повертає код символу

натиснутої клавіші, а для її роботи підключаємо бібліотеку <*conio.h*> і т.д.

Файли з розширенням *h* називаються **заголовковими**. Розташовуються заголовкові файли в підкаталозі *INCLUDE*, і ви можете переглянути вміст цих файлів, але змінювати їх забороняється!

Головна функція

Кожна програма на C++ має один вхід, з якого починається виконання програми, – головну функцію, до складу якої входить перший виконуваний оператор.

Запис

`int main (int argc, char* argv[])`, чи

`int main ()`.

визначає головну функцію. Ваші програми повинні завжди включати одну і тільки одну функцію з ім'ям **main**.

Слово int перед назвою функції означає, що дана функція повертає значення типу **int**. В дужках після **main** у першому прикладі перелічені аргументи командного рядка. Такий варіант запису (з аргументами командного рядка **(int argc, char* argv[])**) слід використовувати, якщо передбачається обробка програмою вхідних аргументів. Зараз на аргументи командного рядка ми уваги не звертатимемо. Можна просто вилучити їх опис, якщо система згенерувала його автоматично, і використовувати другий варіант заголовка функції.

Крім головної функції програма може містити інші функції.

За правилами C++, до функції **main** висуваються особливі вимоги. Зокрема, її слід оголошувати такою, що повертає значення типу **int**. Повернення нею значення 0 (оператор **return 0;**) означає, що виконання функції **main**, а отже й всієї програми, завершилось успішно. Інші значення є кодами повідомлень про помилки, які трапились в ході виконання програми.

Групуючі символи {}

Служать для групування зв'язаних операторів. У простих програмах, які ми надалі будемо складати, ці символи будуть групувати оператори, які відповідають операторам вашої головної функції, чи допоміжної, якщо така буде використовуватися. Пізніше ви вивчите інші випадки застосування групуючих символів.

Виведення повідомлень на екран

Як було сказано вище, для виведення даних на екран використовується об'єкт **cout** (детальне вивчення якого не входить у наш курс) та операція виведення **<<**.

Розглянемо приклад виведення чисел на екран:

```
#include<iostream.h>                    //Програма 1.2
#include<conio.h>
int main()
{
cout << 1001;
getch();
return 0;
}
```

Спробуйте вивести дійсне число (їх часто називають числами із плаваючою крапкою):

```
cout<<0.8976;
```

Потім направте у вихідний потік числа в такий спосіб:

```
cout<<1<<2<<0<<0<<1;
```

Перевірте, як спрацюють такі рядки:

```
cout<<"Vvedi ocinku: "<<12<<endl;
cout<<"Ocinka "<<12<<" - uljublena."<<endl;
```

Отже, у вихідний потік можна виводити як числа, так і текстові повідомлення. Текст при цьому слід брати в лапки. В одному операторі можна вивести послідовно декілька числових та текстових значень.

Спеціальні символи виведення

Випробуйте таку програму:

```
#include<iostream.h>                    //Програма 1.3
#include<conio.h>
int main()
{
cout<<"Ryad 1\nRyad 2";
getch();
return 0;
}
```

А тепер змініть програму так:

```
cout<<1<<'\n'<<0<<'\n'<<3;
```

Послідовності символів, що починаються із символу '\' («зворотний слеш») називають **керуючими послідовностями** або **спеціальними символами**. Вони зображаються на екрані двома символами, але компілятором сприймаються як один символ.

У наведених прикладах використана керуюча послідовність **\n** – символ нового рядка, яка при виведенні поміщає курсор у початок наступного рядка, аналогічно до **endl** (кінець рядка).

Символ «зворотний слеш» використовується для включення в рядок:

- кодів, що не мають графічного зображення (наприклад, **\a** – звуковий сигнал, **\n** – переведення курсору в початок наступного рядка);
- символів апострофа (**'**), зворотного слеша (****), знака питання (**?**) і лапок (**"**).

Спеціальні символи розташовуються або у одинарних лапках, якщо використовуються окремо, або у подвійних, разом з іншими символами рядка.

Призначення спеціальних символів

Символ	Призначення
\a	Звуковий сигнал (дзвінок)
\b	Крок назад (зворотний пропуск)
\f	Перехід на нову сторінку
\n	Перехід на новий рядок
\r	Повернення каретки (не перехід на новий рядок!)
\t	Символ горизонтальної табуляції
\v	Символ вертикальної табуляції
****	Символ «зворотний слеш»
\?	Знак питання
\'	Одинарні лапки
\"	Подвійні лапки
\0	Нульовий символ

Наприклад, якщо всередині рядка потрібно записати лапки, їх випереджають косою рискою, за якою компілятор відрізняє їх від лапок, що обмежують рядок:

```
"Видавництво\"Аспект\" "
```

Коли ми вивчатимемо рядки, то побачимо, що у кінець рядкової константи компілятор додає спеціальний символ NULL (тобто '**\0**').

Спробуйте дослідити цю програму:

```
#include<iostream.h>                    //Програма 1.4
#include<conio.h>
int main()
{
cout<<"Dzvon!\a\t Dzvon!\a\t";
getch();return 0;
}
```

Ширина виведення

Модифікатори формату використовуються для керування шириною поля, що відводиться для розміщення значення, яке виводиться. Модифікатор **setw** дозволить вам регулювати кількість символів, займаних виведеним числом. Але при цьому ви повинні включити в програму заголовковий файл <*iomanip.h*>:

```
#include<iostream.h>                    //Програма 1.5
#include<iomanip.h>
#include<conio.h>
int main()
{
cout << "Druk:" << setw(3) << 1012 << endl;
cout << "Druk:" << setw(4) << 1012 << endl;
cout << "Druk:" << setw(5) << 1012 << endl;
cout << "Druk:" << setw(6) << 1012 << endl;
getch();return 0;
}
```

При використанні **setw** *ви вказуєте* ***мінімальну*** *кількість символьних позицій, займаних числом!*

У програмі 1.5 модифікатор **setw(3)** вказує мінімум 3 символи, однак у зв'язку з тим, що число 1012 потребує більше трьох символів, об'єкт **cout** з операцією виведення **<<** використовує реально необхідну кількість символів. Для кожного виведеного значення використовується окремий модифікатор **setw**.

Збереження програм у Borland C++Builder6

Для збереження вашої програми (проекту) виконайте такі дії: «File» → «Save Project As…». У вікні «Save Unit1 As» виберіть папку, наприклад: *E:\10kl\Petrenko* та натисніть «Создание новой папки». Назвіть її *wpr1-1*. Потім відкрийте папку *wpr1-1* і для першого завдання знову створіть папку, наприклад *1*. Відкрийте її і натисніть «Сохранить» для файлу з ім'ям *Unit1.cpp* а також для файлу *Project1.bpr*.

Зверніть увагу: програма, яку ви випробовували збережена у файлі *Unit1.cpp*, а файл *Project1.bpr* з описом загальних властивостей проекту був створений автоматично.

Щоб відкрити проект, виберіть «File» → «Open Project» і у діалоговому вікні, що з'явиться, виберіть потрібну папку та відкрийте файл *Project1.bpr*.

Питання для самоконтролю:

1. *Назвіть основні елементи мови програмування?*
2. *Що таке «оператор», «складений оператор»?*
3. *Назвіть основні елементи, з яких складаються блок-схеми алгоритмів?*
4. *Що таке «лінійний» алгоритм?*
5. *Як створити проект консольної програми?*
6. *Що таке «компілятор»?*
7. *Як компілятор повідомляє про помилку у програмі?*
8. *Що називають «групуючими символами»?*
9. *Які файли називаються заголовковими?*
10. *Що таке «головна функція»?*
11. *Як вивести повідомлення на екран?*
12. *Назвіть спеціальні символи виведення. Для чого вони призначені?*
13. *Як керувати шириною поля виведення?*

Вправа 1-1.

1) Випробувати в процесі уроку програми 1.1–1.5 з усіма вказаними доповненнями.

Збережіть програми, створивши у власній папці нову папку *wpr1-1*.

Примітка. Якщо власна папка ще не створена, її назву та розміщення слід уточнити у вчителя. Повний шлях, орієнтовно, може відповідати такому шаблону:

<диск>:\<клас>\<прізвище>\<вправа>.

Наприклад, *D:\10kl\petrenko\wpr1-1*.

1.2. Типи даних. Змінні в С++. Присвоєння значень змінним

Програми пишуть для того, щоб обробляти дані. Дані різних типів по-різному зберігаються у пам'яті комп'ютера і обробляються теж по-різному. Від типу даних залежить:

1. внутрішнє подання даних у пам'яті комп'ютера;
2. множина значень, яких можуть набувати величини цього типу;
3. операції й функції, які можна застосовувати до величин цього типу.

Виходячи із цих характеристик, програміст вибирає тип кожної величини, використовуваної в його програмі. Обов'язковий опис типу дозволяє компілятору робити перевірку допустимості різних конструкцій програми.

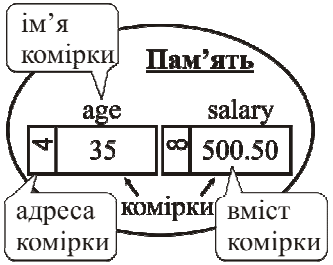
Основні типи даних часто називають **арифметичними**, оскільки їх можна використовувати в арифметичних операціях.

Типи даних у С++

Назва	Позначення	Діапазон значень	Розмір, байт
Байт	`char`	від –128 до +127	1
Байт без знака	`unsigned char`	від 0 до 255	1
Ціле число	`int`	від – 2147483648 до + 2147483647	4
Логічне значення	`bool`	значення true (істина) або false (неправда)	1
Дійсне число одинарної точності	`float`	від ±3.4e-38 до ±3.4e+38 (7 значущих цифр)	4
Дійсне число подвійної точності	`double`	від ±1.7e-308 до ±1.7e+308 (15 значущих цифр)	8
Дійсне число збільшеної точності	`long double`	від ±1.2e-4932 до ±1.2e+4932	10

Перші чотири типи називають **цілочисельними** (цілими), останні три – типами **з плаваючою крапкою**.

В останній колонці позначено, який об’єм пам'яті займає елемент даних відповідного типу. Наприклад, дані типу `int` та `float` займають по 4 байти. На малюнку умовно зображено розміщення у пам’яті комп’ютера двох величин: `age` типу `int`, та `salary` типу `float`. Зверніть увагу, що адреса комірки `age` відрізняється від адреси комірки `salary` на чотири байти.



Дані, які можуть змінюватися в ході виконання програми, називають **змінними**. Змінна – це іменована область пам'яті, яка має свою адресу і в якій зберігаються дані певного типу. У змінної є ім'я й значення. Ім'я служить для звертання до області пам'яті, у якій зберігається значення.

Перш ніж використати змінну, її потрібно оголосити, вказавши **ім'я й тип**:

```
int age; float salary;
```

Імена змінних бажано вибирати змістовні, щоб потім не ламати голову над тим, що ж позначає ця змінна. В іменах змінних можна використовувати букви латинського алфавіту, цифри, знак підкреслення (_).

На відміну від деяких інших мов програмування, у мові С++ букви нижнього й верхнього регістрів розрізняються, тому, наприклад, ***`salary`*** *і* ***`Salary`*** *– це імена двох різних змінних!*

Ім’я змінної не може починатися з цифри, але може зі знака підкреслення. Як імена змінних не можна використовувати ключові слова С++ (тобто зарезервовані слова, які мають спеціальне значення для компілятора. Наприклад: `void`, `if`, `for` …).

Дані, значення яких не змінюється в процесі виконання програми, називають **константами**.

При оголошенні константи вказують її тип, ім’я та значення. Модифікатор `const` показує, що вказане значення змінювати не можна:

```
const float pi=3.14159
```

Приклади констант, якими ми користуємось у житті: число днів у тижні чи місяців у році. Ці дані не змінюються за жодних обставин, тому ці значення – константи.

Приклад 1.

```
#include<iostream.h>                    //Програма 1.6
#include<conio.h>
int main()
{
int age=35;          //оголошення та ініціалізація
float salary=500.50;                        // змінних
cout<<"Robitnyky "<<age<<" rokiv "<<endl;
cout<<"Oklad: "<<salary<<" grn"<<endl;
getch();
return 0;
}
```

Приклад 2.

Опишемо та ініціалізуємо змінні по-іншому:

```
…
int age;                          //оголошення
age=35;                           //ініціалізація
float salary;                     //оголошення
salary=500.50;                    //ініціалізація
…
```

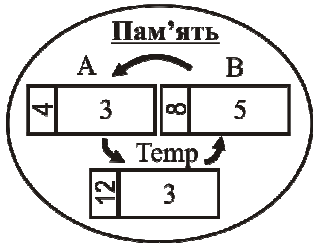
Ці дві програми цілком ідентичні.

Як ми бачимо, тут використовується знак «**=**», але цей знак читається не «дорівнює», як в математиці, а «присвоїти». Це означає, що змінній цілого типу **age** присвоюється значення 35, а десятковій змінній **salary** присвоюється значення 500.50. Після виконання присвоєння, значення змінних будуть поміщені у комірки пам'яті комп'ютера з відповідними адресами (див. малюнок вище).

У математиці рівняння **n=n+100** не має сенсу, а в мові С++ означає, що значення змінної з комірки пам'яті з ім'ям **n** після виконання присвоєння збільшилося на 100.

Приклад 3*. Обмін значень двох змінних.*

Багато задач з програмування потребує переставляння значень двох змінних. При цьому змінній **A** треба присвоїти значення змінної **B** та навпаки, змінній **B** присвоїти значення змінної **A**. Але якщо ми запишемо **A = B**, то значення, яке до цього було у змінній **A**, буде втрачене. Тому треба використати ще одну змінну, наприклад **Temp**, для тимчасового збереження в ній значення змінної **A**. Переставляння виконується трьома присвоюваннями (див. малюнок):




```
Temp = A;
A = B;
B = Temp;
```

Випробуйте програму з обміном значень змінних:

```
#include<iostream.h>                  //Програма 1.7
#include<conio.h>
int main()
{
int A=3;
int B=5;
cout<<"A= "<<A<<" B= "<<B<<endl;
Temp = A;
A = B;
B = Temp;
cout<<"A= "<<A<<" B= "<<B<<endl;
getch();
return 0;
}
```

Питання для самоконтролю:

1. *Для чого використовують типи даних?*
2. *Для чого описувати дані? Чи обов'язково це робити?*
3. *Що таке «змінна»?*
4. *Що таке числа з «плаваючою крапкою»?*
5. *Як змінні зберігаються у пам'яті комп'ютера?*
6. *Назвіть основні правила написання імен змінних?*
7. *Що таке «константа»?*
8. *Як здійснюєтья перестановка значень двох змінних?*

Вправа 1-2.

1) Випробувати в процесі уроку програми 1.6–1.7 з усіма вказаними доповненнями.
2) Випробуйте програму 1.7 для змінних A і B типу **double**. Чим відрізнятиметься малюнок з прикладу 3 для цих змінних?

Збережіть програми, створивши у власній папці нову папку *wpr1-2*.

1.3. Практична робота №1 «Робота в середовищі програмування Borland C++»

Напишемо програму, при виконанні якої на екран виводитиметься резюме консультанта фірми «Сервер» Петра Іванова, день народження у якого 25 квітня. На екрані ми повинні побачити:

```
Resume
Surname:        Ivanov
Name:           Petro
Work at:        "Server"
Date of birth:  25.04
```

Запускаємо Borland C++. Створюємо проект з іменем *Resume*:

```
#include<iostream.h>
#include<conio.h>
int main()
{
// Виведення через 3 табуляції тексту:Resume
// та переведення рядка
    cout<<"\t\t\tResume\n";
// Виведення порожнього рядка
    cout<<"\n";
//  Через 2 табуляції виведення тексту: Surname,
//  та ще через 1 табуляцію – Ivanov
    cout<<"\t\tSurname:"<<"\tIvanov\n";
    cout<<"\n";
// Виведення через 2 табуляції імені
    cout<<"\t\tName:"<<"\t\tPetro\n";
    cout<<"\n";
// Назва фірми «Сервер» буде вказана в лапках
    cout<<"\t\tWork at:"<<"\t\"Server\"\n";
    cout<<"\n";
    cout<<"\t\tDate of birth: "<<"\t25.04";
    cout<<"\n";
    getch();return 0;
}
```

Питання для самоконтролю:

1. *Назвіть математичні операції у С++.*
2. *Як працює операція % – остача від ділення?*
3. *Як у С++ виконуються префіксна і постфіксна операції збільшення й зменшення?*
4. *Які скорочення у виразах дозволяє робити мова С++?*
5. *Як увести дані з клавіатури?*

Вправа 1-3.

1) Переробіть цю програму для виведення власного резюме (наприклад, учня ЗОШ №3 м. Васюки Степанюка Івана).
2) Уведіть змінну **age** – вік, та виведіть на екран через 1 табуляцію.
3) Уведіть змінну **srball** – середній бал атестату, та виведіть на екран.
4) Виділіть під змінну **srball** п'ять позицій на екрані. Яку бібліотеку при цьому потрібно підключити?

5) Уведіть змінні **little** = 1.535×10^{250} та **big** = 2.2×10^{-50}.
6) Виконайте перестановку значень цих двох змінних.

Збережіть програму, створивши у власній папці нову папку *wpr1-3*.

1.4. Виконання простих операцій. Математичні операції мови С++. Уведення даних

Оскільки інформація в комп'ютері кодується числами, то в більшості програм є потреба виконувати певні обчислення. Для обчислення значень використовують вирази, які складаються з операндів, знаків операцій та дужок. Результат обчислення можна вивести на екран або зберегти у змінній відповідного типу.

Математичні операції у С++:

+ - додавання;
– - віднімання;
***** - множення;
/ - ділення;
% - остача від ділення.

Операція ділення **/** виконується тільки для операндів арифметичного типу. Якщо обидва операнди – цілі числа, то результат ділення округлюється до цілого числа, інакше тип результату відповідатиме правилам перетворень.

Операція **%** – остача від ділення – застосовується тільки до цілих операндів (наприклад, якщо **x=11; y=4;**, то **x%y** дорівнює 3).

Розглянемо приклади програм з обчисленнями:

```
#include<iostream.h>                    //Програма 1.8
#include<conio.h>
int main()
{
cout<<5+4<<endl;              //Виведення результатів
cout<<"21.678/3="<<21.678/3<<endl;     //на екран
const float pi=3.14159;
//Обчислимо довжину кола, радіус якого 3,5
float r=3.5;
float l=2*pi*r;          //Запис результату у змінну
cout<<"L="<<l<<endl;
getch();
return 0;
}
```

Зверніть увагу, що результати перших двох обчислень виводяться безпосередньо на екран. Третій результат зберігається у змінній **l**, а її значення потім виводиться. Виведення другого і третього результатів супроводжується текстовими коментарями (”21.678/3=”, ”L=”).

Мова С++ дозволяє робити такі скорочення:

x+=y рівносильно **x=x+y**	**x*=y** рівносильно **x=x*y**
x-=y рівносильно **x=x-y**	**x/=y** рівносильно **x=x/y**

Зауважимо, що порядок виконання операцій також залежить від звичайних дужок ( ). Вирази в дужках, як у математиці, обчислюються першими. Крім того, вибираючи типи змінних, які ви будете використовувати в програмах, не забувайте про можливість помилок. Щоб уникнути їх, користуйтеся таблицею «Типи величин».

Операції збільшення й зменшення

Доволі часто в програмуванні доводиться збільшувати значення змінної на одиницю: **i = i + 1**. Мова С++ дозволяє цей запис скоротити, завдяки операції збільшення: **i++**.

Знак операції збільшення ви можете розміщувати до або після змінної:

```
++i;                    i++;
```

У першому виразі знак операції записаний перед змінною, тому це – **префіксна операція збільшення**. Аналогічно, у другому виразі використана **постфіксна операція збільшення**. С++ трактує ці дві операції по-різному. Наприклад, розглянемо такий вираз з операцією присвоювання:

```
number=i++;
```

При обчисленні цього виразу поточне значення **i** буде присвоєне змінній **number**. Після цього постфіксна операція **++** призведе до збільшення поточного значення **i**. Використання постфіксної операції в цьому випадку робить наведений вище оператор еквівалентним таким двом операторам:

```
number = i;
i = i + 1;
```

Тепер розглянемо наступний вираз, що містить операцію присвоювання та префіксну операцію збільшення:

```
number=++i;
```

У цьому випадку спочатку буде збільшене значення **i**, а потім результат, присвоєний змінній **number**. Використання префіксної

операції збільшення робить показаний вище оператор, еквівалентний таким двом операторам:

```
i = i + 1;
number = i;
```

Уважно дослідіть програму:

```
#include<iostream.h>                    //Програма 1.9
#include<conio.h>
int main()
{
int i,number;
i=10;
cout << "i="<<i<<endl;
number=i++;
cout << "number=" << number << endl;
getch();
return 0;
}
```

А тепер проаналізуйте роботу цієї програми:

```
#include<iostream.h>                    //Програма 1.10
#include<conio.h>
int main()
{
int i,number;
i=10;
cout << "i="<<i<<endl;
number=++i;
cout << "number=" << number << endl;
getch();
return 0;
}
```

Отже, у випадку застосування префіксної форми, значення обох змінних стають рівними між собою і дорівнюють **i+1**. Постфіксна ж операція виконується після присвоєння, тому збільшиться на одиницю лише значення змінної **i**.

Дослідіть самостійно, яким буде результат застосування операції зменшення **--** (**i--**, **--i**).

Уведення даних з клавіатури

На минулих заняттях ви ознайомилися з вихідним потоком – об’єктом **cout** та операцією **<<**, що здійснює виведення

інформації на екран. Для уведення даних у С++ використовується вхідний потік – об'єкт **cin** та операція уведення **>>**:

```
#include<iostream.h>                    //Програма 1.11
#include<conio.h>
int main()
{
int num;
cout<<" Ocinka:";
cin >>num;  //Уведення значення з вхідного потоку
cout<<"Vasha ocinka - "<<num<<" baliv"<<endl;
getch();
return 0;
}
```

а тепер змінимо програму так:

```
…                                      //Програма 1.12
int num1, num2;
cout<<"Vvedit 2 ocinki:";
cin>>num1>>num2;
cout<<"Vasha ocinka - "<<num1<<" baliv"<<endl;
cout<<"Vasha ocinka - "<<num2<<" baliv"<<endl;
…
```

В цьому випадку з вхідного потоку спочатку уводиться змінна **num1**, а потім **num2**. Числа, запитувані програмою, потрібно уводити, відокремивши пропуском:

```
Vvedite 2 ocinki: 12  11
```

По закінченні уведення натиснути Enter.

Також можна після кожного числа натискати клавішу Enter або Tab. Крім того, зверніть увагу, що кілька змінних одного типу можна описати так:

```
int num1, num2;
```

При уведенні даних із клавіатури будьте особливо уважні: уникайте розбіжності типів і пов'язаних із цим помилок (наприклад, тип **char** може набувати значень –128...+127; **char A='5'** – це символ, **int A=5** – число і т.п.).

Тепер спробуємо уводити текстові дані:

```
#include<iostream.h>                    //Програма 1.13
#include<conio.h>
int main()
{
char letter;
```



```
cout<<"Vvedite imya:";cin>>letter;
cout<<"Hello "<<letter<<endl;
getch();
return 0;
}
```

Як ви помітили після випробування програми, відбувається уведення тільки одного, першого символу. Це пояснюється тим, що для збереження даних типу **char** відводиться всього один байт (8 біт), і в ньому може зберігатися тільки один символ (див. табл. «Типи величин»). Пізніше ви навчитеся уводити в програму із клавіатури цілі рядки.

Вправа 1-4.

1) Випробуйте за допомогою комп'ютера програми 1.8–1.13 з усіма вказаними доповненнями та проаналізуйте отримані результати.
2) За аналогією з операцією збільшення (програми 1.9 та 1.10) перевірте, як буде працювати операція зменшення **--** (**i--**, **--i**).
3) Напишіть програму й дослідіть можливості спрощеного запису присвоєння з обчисленням у С++ (**x+=y**; **x*=y**; **x-=y**; **x/=y**).
4) Складіть програму для обчислення площі прямокутника за уведеними з клавіатури довжинами сторін, вираженими цілими числами, меншими ніж 150.

Збережіть програми, створивши у власній папці нову папку *wpr1-4*.

1.5. Практична робота №2 «Уведення та виведення даних»

1) Скласти програму для обчислення швидкості, з якою бігун долає дистанцію. Довжина дистанції L=1000 м, T=3.25 с. Знайти швидкість за формулою: V=L/T, та вивести на екран результат.
2) Скласти програму, для переведення відстані, заданої у верстах, у кілометри (1 верста – 1066,8 м). Вигляд екрану після виконання програми приблизно такий:

```
Versta->km
Vidstan u verstah: 100
100 verst – 106.68 km
```

3) Скласти програму, для переведення дробового числа у грошовий формат. Наприклад, число 12,5 повинно набути вигляду: **12 grn 50 kop**.

Збережіть програми, створивши у власній папці нову папку *pr2*.

1.6. Математичні функції в С++. Запис математичних виразів мовою С++

Для використання у програмі математичних функцій потрібно приєднати до програми заголовковий файл <*math.h*>.

Основні математичні функції мови С++

ceil	округлення вгору	ceil(2.5)=3
floor	округлення вниз	floor(2.5)=2
sin	синус	Кут задається у радіанах
cos	косинус	
tan	тангенс	
abs	модуль цілого числа	
fabs	модуль дробового числа	
fmod(x,y)	остача від ділення x на y	fmod(5,2)=1
pow(x,y)	піднесення до степеня x^y	pow(2,3)=8
sqrt	квадратний корінь	

M_PI – число π = 3.14159... також можна використовувати, підключивши заголовковий файл *math.h*.

Приклад 1. *Записати мовою С++:*

а) $z = \frac{x+y}{3} \cdot \frac{7}{x-y}$

Відповідь: **z=(x+y)/3*7/(x-y);**

б) $s = \frac{-b+\frac{1}{a}}{\frac{2}{c}}$

Відповідь: **s=(-b+1/a)/(2/c);**

Приклад 2. *Записати мовою С++. Чому дорівнюють значення z1 і z2 при b=2?*

$$z_1 = \frac{\sqrt{2b+2\sqrt{b^2-4}}}{\sqrt{b^2-4}+b+2}; \qquad z_2 = \frac{2}{\sqrt{b+2}}$$

Відповідь: z1=2, z2=0.5

```
z1=sqrt(2*b+2*sqrt(b*b-4))/(sqrt(b*b-4)+b+2);
z2=2/sqrt(b+2);
```

Приклад 3. *Записати мовою С++. Чому дорівнюють значення z1 і z2 при α=0, β=π.*

$$z_1 = (\cos\alpha - \cos\beta)^2 - (\sin\alpha - \sin\beta)^2;$$

$$z_2 = -4\sin^2\frac{\alpha-\beta}{2}\cdot\cos(\alpha+\beta)$$

Відповідь: z1= z2=4

```
z1 = pow(cos(a)-cos(b),2)-pow(sin(a)-sin(b),2);
z2 = -4*pow(sin((a-b)/2),2)*cos(a+b);
```

Питання для самоконтролю:

1. *Які ви знаєте математичні функції мови С++?*
2. *Який заголовковий файл треба підключити для використання математичних функцій?*
3. *Знайдіть помилку:* $\frac{a+b}{c*d} = (a+b)/c*d$

Вправа 1-6.

1) Поясніть, чому дана програма написана неправильно:

```
main ()
{
cout << "My name is Sasha!";
}
```

2) Складіть програму для знаходження периметра квадрата, якщо задано його площу.
3) Знайти площу кільця за заданими зовнішнім та внутрішнім радіусами.
4) Дано катети прямокутного трикутника. Знайти його периметр.
5) З клавіатури уводиться двоцифрове число. Знайти:
   а) число десятків у ньому; б) суму його цифр;
   в) число одиниць у ньому; г) добуток його цифр.

Збережіть програми, створивши у власній папці нову папку *wpr1-6*.

1.7. Практична робота № 3 «Створення лінійних програм»

1) Обчислити без використання комп'ютера значення виразів:
   а) `sin(pow(x,2)-1)+2*abs(y)/cos(2+y)` при x = 1, y = –2;
   б) `ceil(6.9)-floor(6.2)-1`;
   в) `30/6-30%5+2`;
   г) `3*7/2%7/3-ceil(sin(1))`.
2) Обчислити значення виразів (значення невідомих ввести із клавіатури):
   а) $\frac{a+b}{c}+\frac{c^2}{a-b}$; б) $3+\frac{1}{2}+\frac{c}{(a+b)^2}$; в) $m\,g\cos\left(a^2\right)$.
3) Увести значення змінних й обчислити 2 вирази. Чому дорівнюють z1 та z2, при α=π, a=4, x= π, y=0:

а) $z_1=\frac{\sin 2\alpha+\sin 5\alpha-\sin 3\alpha}{\cos\alpha+1-2\sin^2 2\alpha}$; $z_2=2\sin\alpha$;

б) $z_1=\cos^4 x+\sin^2 y+\frac{1}{4}\sin^2 2x-1$; $z_2=\sin(y+x)\cdot\sin(y-x)$;

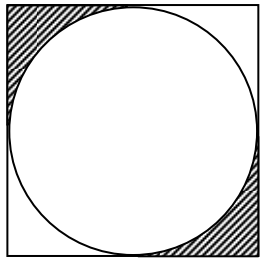
в) $z_1=1-\frac{1}{4}\sin^2 2\alpha+\cos 2\alpha$; $z_2=\cos^2\alpha+\cos^4\alpha$;

г) $z_1=\left(\frac{a+2}{\sqrt{2a}}-\frac{a}{\sqrt{2a}+2}+\frac{2}{a-\sqrt{2a}}\right)\cdot\frac{\sqrt{a}-\sqrt{2}}{a+2}$; $z_2=\frac{1}{\sqrt{a}+\sqrt{2}}$.

Збережіть програми, створивши у власній папці нову папку *pr3*.

1.8. Тематичне оцінювання з теми «Програма. Мова програмування»

1) У квадрат вписане коло (див. мал.).
   а) Визначити площу заштрихованої частини фігури. Довжину сторони квадрата увести із клавіатури.
   б) Визначити площу не заштрихованої частини фігури. Радіус кола увести із клавіатури.



2) Дано трицифрове число.
   а) Знайти число, отримане при прочитанні його цифр у зворотному напрямку.
   б) У ньому закреслили першу ліворуч цифру й приписали її наприкінці. Знайти отримане число.
3) Увести значення змінних й обчислити 2 вирази (значення першого та другого виразів повинні збігатися):

а) $z_1=\cos\alpha+\cos 2\alpha+\cos 6\alpha+\cos 7\alpha$; $z_2=4\cos\frac{\alpha}{2}\cdot\cos\frac{5}{2}\alpha\cdot\cos 4\alpha$.

б) $z_1=\frac{1-2\sin^2\alpha}{1+\sin 2\alpha}$; $z_2=\frac{1-tg\,\alpha}{1+tg\,\alpha}$.

Збережіть програми, створивши у власній папці нову папку *ta1*.

Питання для самоконтролю:

1. *Для чого в С++ служить крапка з комою?*
2. *Чим* `'7'` *відрізняється від* `7`*?*
3. *Який тип даних потрібно використати для запису вартості товару в гривнях?*
4. *У чому полягає відмінність між константою й змінною?*
5. *Чи завжди зберігає змінна своє значення в ході виконання програми?*
6. *Чи правильний запис:*
```
const int a=25;
…
a=a+1;
```
7. *Як вивести на екран символ «лапки»?*
8. *Чи можна використати різні типи даних в одній операції? Якщо так, то як це відіб'ється на результаті операції?*
9. *Навіщо при записі операцій використовують круглі дужки?*
10. *Чи правильно зроблено перестановку значень двох змінних:*
```
…
int a=12;
int b=21;
a=b;
b=a;
…
```

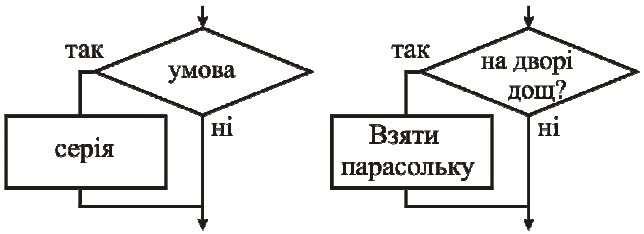
2. Алгоритми з розгалуженням

2.1. Оператор розгалуження if

Поняття розгалуження

Людині досить часто доводиться приймати рішення, як вчинити, які дії виконати. Розглянемо приклад: **якщо** на дворі йде дощ (відповідь – «так»), **то** виконуємо дії: «Взяти парасольку», інакше (відповідь – «ні») пропускаємо ці дії і виконуємо подальшу «програму».

Якщо відповідь на питання-умову «На дворі йде дощ?» є позитивною, то виконується серія (одна або кілька) команд (гілка «так»). Після виконання серії команд виконавець переходить до наступної після розгалуження команди.



Будь-яка команда серії може бути командою розгалуження. У цьому випадку кажуть, що команди розгалуження вкладені одна в одну.

Взагалі, умова — це будь-яке запитання такого типу, що допускає лише дві можливі відповіді: «так» або «ні». Перевірка умови повинна бути допустимою дією виконавця. У програмі на С++ використовуються умови, що стосуються певних програмних об'єктів (наприклад, змінних).

Оператор розгалуження

Оператор розгалуження (або умовний оператор) дозволяє здійснювати під час виконання програми перевірку умови, та на основі перевірки виконувати той або інший оператор. Загальний вигляд оператора розгалуження такий:

```
if (умова, яку слід перевірити)
        оператор;
```

Оператор розгалуження **if** виконує перевірку, використовуючи операції порівняння С++. Якщо результат перевірки є істиною, то буде виконаний оператор, записаний після умови.

Наведені блок-схема та запис умовного оператора є скороченою його формою, на відміну від повної форми, яка буде розглянута нижче. Слід пам'ятати, що в кінці оператора-виразу ставиться крапка з комою.

А зараз розглянемо операції порівняння, які використовуються в мові програмування С++.

Операції порівняння в С++

Операція	Питання	Приклад
==	Два значення рівні?	(i == 100)
!=	Два значення не рівні?	(a != b)
>	Перше значення більше, ніж друге?	(var > num)
<	Перше значення менше, ніж друге?	(x < 345)
>=	Перше значення більше або дорівнює другому?	(s >=30)
<=	Перше значення менше або дорівнює другому?	(age3 <= 5.0)

Розглянемо приклад:

```
#include<iostream.h>                    //Програма 2.1
#include<conio.h>
int main()
{
float x,y=6.8;
cout<<"Vvedite x:";
cin>>x;
if(x>=y)                                 //Перевірка умови
   cout<<"x>=y!!!"<<endl;               //Простий оператор
getch();
return 0;
}
```

У цьому прикладі оператор розгалуження перевіряє умову і, якщо вона істинна, виводить на екран: **x>=y!!!**. В інших випадках, якщо умова не виконується, програма закінчується не виводячи ніяких повідомлень на екран.

Якщо необхідно виконати кілька інструкцій, коли результат порівняння – істина, вони повинні бути згруповані всередині фігурних дужок **{…}**.

Декілька операторів, обмежені фігурними дужками, називають складеним оператором або блоком.

Отже, оператор, який буде виконано у випадку істинності умови, може бути складеним.

***Приклад**. Фрагмент ігрової програми, в якому приймається рішення про продовження гри:*

```
#include<iostream.h>                    //Програма 2.2
#include<conio.h>
int main()
{
char game_end;
cout<<"Prodovgimo gru?(t/n):";
cin>>game_end;
 if(game_end=='t')                  // Перевірка умови
 {
  cout<<"Pravila gri:" << endl;        //Складений
  cout<<"Opis pravil gri...";           //оператор
 }
getch();
return 0;
}
```

Як бачимо, в цьому випадку дія оператора **if** поширюється на обидві команди, що входять до складеного оператора.

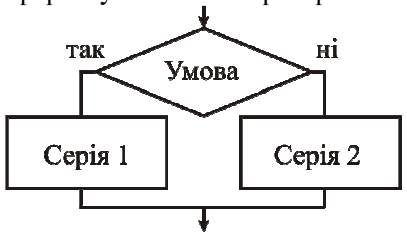
Також з цього прикладу видно, що можна порівнювати не лише числа, але й символи (тобто дані типу **char**). Кожен символ має числовий код, і саме ці коди порівнюються як звичайні числа. Наприклад, код символу ‘A’ дорівнює 65, а код символу ‘Z’ дорівнює 90.

Повна форма оператора розгалуження

Дуже часто виникає потреба при виконанні умови в операторі розгалуження **if** виконувати одну серію команд, а при не виконанні – іншу. В такому випадку використовують повну форму оператора розгалуження:

```
if (умова, яку слід перевірити)
  Оператор 1;
else
  Оператор 2;
```


Блок-схема повної форми умовного оператора має такий вигляд:



Якщо відповідь на питання-умову є позитивною, то виконується серія команд «серія 1» (гілка «так»), якщо ж відповідь негативна – серія команд «серія 2» (гілка «ні»). Після виконання однієї з серій команд виконавець переходить до наступної після розгалуження команди.

Оператор розгалуження **if** у цьому випадку діє так: якщо результат перевірки є істиною, **if** виконує оператори, записані після умови, інакше – оператори після ключового слова **else**.

***Приклад** застосування повної форми умовного оператора:*

```
#include<iostream.h>                    //Програма 2.3
#include<conio.h>
int main()
{
int oc;
cout<<"Vvedit' ocinky:";cin>>oc;
if (oc >=5)
{
  cout<<"Vasha ocinka "<<oc<<" baliv"<<endl;
  cout<<"Zalik!"<<endl;
}
else
{
  cout<<" Vasha ocinka "<<oc<<" baliv "<<endl;
  cout<<"Nezalik!"<<endl;
}
getch();return 0;
}
```

У наведеній програмі складені оператори застосовуються як для **if**, так і для **else**. Але виконано буде лише одну серію операторів.

У ЖОДНОМУ РАЗІ не будуть виконані ОБИДВА складені оператори.

Питання для самоконтролю:

1. *Для чого використовують умовний оператор?*
2. *Назвіть дві форми умовного оператора?*
3. *Наведіть приклад з використанням розгалуження?*
4. *Назвіть операції порівняння в С++?*
5. *Для чого використовують ключове слово «else»?*
6. *Чи може хоч одна з команд серії бути командою розгалуження?*
7. *Чи можуть виконатися обидві серії команд в умовному операторі?*

Вправа 2-1.

1) Випробуйте програму 2.1. Уведіть значення x=5; x=7.2. Проаналізуйте результати.
2) Випробуйте програму 2.2. Уведіть значення змінної **game_end** рівним «**t**», потім «**n**». Уведіть значення «**T**», чи «**N**» (великі літери), або українською «**т**» чи «**н**». Поясніть результати роботи програми.
3) Випробуйте програму 2.3. Уведіть значення змінної **oc** рівним 4, потім 5 та 10. Поясніть результати. Спробуйте ввести **oc** = -5, а потім **oc** =13. Що вийшло?

Збережіть програми, створивши у власній папці нову папку *wpr2-1*.

2.2. Практична робота № 4 «Програми з оператором розгалуження»

Варіант 1

1) Дано два числа. Виведіть перше з них, якщо воно більше від другого, і обидва числа, якщо це не так.
2) Обчислити площу кільця, перевіривши перед цим правильність уведених даних: радіус отвору не може бути більшим, ніж радіус кільця.
3) Обчислити вартість покупки з урахуванням знижки. Знижка 3% надається, якщо сума покупки перевищує 500 грн., 5% – якщо сума більша, ніж 1000 грн.

Варіант 2

1) Обчислити частку двох чисел. Програма повинна перевіряти правильність уведених даних і, якщо вони неправильні (дільник дорівнює нулю), видавати повідомлення про помилку.
2) Увести радіус кола й сторону квадрата. У якої фігури більша площа?
3) Обчислити вартість покупки з урахуванням знижки. Знижка 10% надається, якщо сума покупки перевищує 1000 грн.

Збережіть програми, створивши у власній папці нову папку *pr4*.

2.3. Логічні операції «І», «АБО», «НЕ». Оператор-перемикач

Дуже часто в програмах доводиться перевіряти відразу кілька умов. Наприклад, у прикладі 2.2 при відповіді на питання "**Prodovgimo gru?(t/n):**" неуважний гравець може увести: «**T**» або «**N**», тобто символи верхнього регістру, що спричинить неправильну реакцію програми. Отут нам і допоможуть логічні операції.

Логічні операції «АБО» та «І»

Змінимо умову в програмі так:

```
if ((game_end=='t')||(game_end=='T'))
```

|| – логічна операція «АБО».

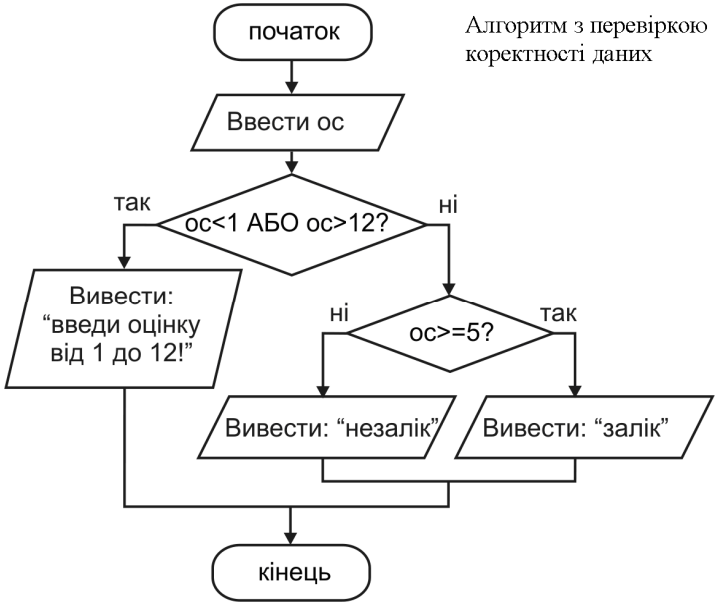
Умова читається так: «якщо змінна дорівнює символу '**t**' АБО дорівнює символу '**T**'».

У наступному прикладі показано, як уникнути некоректного уведення оцінок, наприклад, менших від 1 бала або більших, ніж 12 балів. Скориставшись умовними операторами та логічними операціями, можна здійснити перевірку коректності даних.

Складемо блок-схему та запишемо алгоритм:

```
#include<iostream.h>                    //Програма 2.4
#include<conio.h>
int main()
{
int oc;
cout<<"Vvedi ocinky:";
cin>>oc;
```

```
if ((oc<1)||(oc>12))     //Перевірка правильності
{                                          //оцінки
cout<<"Vvedi ocinky vid 1 do 12 baliv"<<endl;
}
else                  //Реакція на правильну оцінку
{
  if (oc>=5)
  {
   cout<<"Vasha ocinka "<<oc<<" baliv"<<endl;
   cout<<"Zalik"<<endl;
  }
  else
  {
   cout<<"Vasha ocinka "<<oc<<" baliv"<<endl;
   cout<<"Nezalik"<<endl;
  }
}
getch(); return 0;
```



Алгоритм з перевіркою коректності даних

```
}
```

У цьому прикладі серія команд після **else** являє собою також оператор розгалуження.

Переробимо приклад 2.4 так:

```
…
if ((oc>=1)&&(oc<=12))
{
  if (oc>=5)
  {
   cout<<"Vasha ocinka "<<oc<<" baliv"<<endl;
```

&& – логічна операція «І»

```
   cout<<"Zalik"<<endl;
  }
  else
  {
   cout<<"Vasha ocinka "<<oc<<" baliv"<<endl;
   cout<<"Nezalik"<<endl;
  }
}
else
{
cout<<"Vvedi ocinky vid 1 do 12 baliv"<<endl;
}
…
```

У цьому випадку перевірка коректності читається так: «якщо **oc** більше або дорівнює 1 балу **I** менше або дорівнює 12 балам, то виконаються оператори в дужках після **if**, в іншому випадку виконаються оператори після **else**».

Умову, записану з використанням логічних операцій, називають **складеною умовою**.

Уважно дослідіть роботу наведених програм і запам'ятайте таке:

Нехай A і B – умови.

Складена умова A||B виконується, якщо виконується хоча б одна із умов: ***або*** *умова A,* ***або*** *умова B.*

Складена умова A&&B виконується лише тоді, коли виконуються обидві умови: ***і*** *умова A,* ***і*** *умова B.*

Подання логічних значень у С++

У С++ істина подається як ненульове значення, а хибність – як 0.

Припустимо, ваша програма використовує змінну **velo**, щоб зберегти інформацію про те, чи є у користувача велосипед, чи ні. Якщо немає велосипеда, ви можете надати цій змінній значення 0 (хибність):

```
velo=0;
```

Якщо ж є, то можна надати змінній будь-яке ненульове значення, наприклад 1:

```
velo=1;
```

Потім програма може використати цю змінну в якості умови:

```
if (velo) ...
```

Якщо значення змінної ненульове, умова вважається істинною; у іншому випадку (тобто, якщо змінна дорівнює 0), умова хибна. Виходячи з вищесказаного, попередній оператор рівносильний такому:

```
if (velo != 0) ...
```

Наступна програма моделює діалог про те, чи є у користувача велосипед або автомобіль:

```
#include<iostream.h>                    //Програма 2.5
#include<conio.h>
int main()
{
int velo, avto;
cout<<"Maete avtomobil? (1-tak,0-ni):";
cin>>avto;
cout<<"A velosiped? (1-tak,0-ni):";
cin>>velo;
if(velo)
 cout<<"Velo – ce zdorov'ya!\n";
if(avto)
 cout<<"Avto – ce shvydkist!\n";
if(velo && avto)
 cout<<"Avto i velo–ce shvydkist i zdorov'ya\n";
if(velo || avto)
 cout<<"Avto i velo – chudo-tehnika!\n";
getch();
return 0;
}
```


Уважно проаналізуйте цю програму, уводячи різні значення змінних **velo** й **avto**.

Операція заперечення «НЕ»

У наведених вище прикладах, як правило, виконувалася серія операторів, якщо умова істинна. Але іноді необхідно виконати серію операторів, якщо умова хибна. У цьому випадку знадобиться логічна операція «НЕ», позначувана в С++ знаком оклику (**!**):

```
#include<iostream.h>                    //Програма 2.6
#include<conio.h>
int main()
{
int a,b,c;
cout<<"Analiz rivnjannja vidu: a*x*x+b*x+c=0"
<<endl;
cout<<"Vvedit' a,b,c (cherez TAB!):";
cin>>a>>b>>c;
if(!a)
   cout<<"Ce linijne rivnjannja!"<<endl;
else
   cout<<"Ce kvadratne rivnjannja!"<<endl;
getch();return 0;
}
```

! – логічна операція «НЕ»

Операція «НЕ» перетворює хибність в істину, а істину в хибність. Надалі ви часто будете використовувати операцію «НЕ». Наприклад, ваша програма може продовжувати повторювати обробку рядкової константи, поки не досягне її кінця, про що вказуватиме символ \0. Але про це поговоримо пізніше.

Питання для самоконтролю:

1. *Що означає логічна операція «АБО»?*
2. *Що означає логічна операція «І»?*
3. *Що означає логічна операція «НЕ»?*
4. *Як називають умову, записану з використанням логічних операцій?*
5. *Як у С++ представляється істина? А хибність?*
6. *У що операція «НЕ» перетворює хибність?*

Вправа 2-3.

1) Випробуйте програму 2.4. Для цього відкрийте файл завдання 3 з попередньої вправи 2.1. та доопрацюйте програму. Проаналізуйте результати.
2) Випробуйте програму 2.5. Уведіть значення змінної **velo**=0, **avto**=1; **velo**=1, **avto**=0; **velo**=**avto**=1; **velo**=**avto**=0. Поясніть результати роботи програми. Доопрацюйте програму, щоб у останньому випадку на екрані з'являлось: **Kupit' velo abo avto!**
3) Випробуйте програму 2.6. Доопрацюйте її для розв'язування квадратного рівняння.

Збережіть програми, створивши у власній папці нову папку *wpr2-3*.

2.4. Обробка декількох умов

Приклад використання вкладених умов

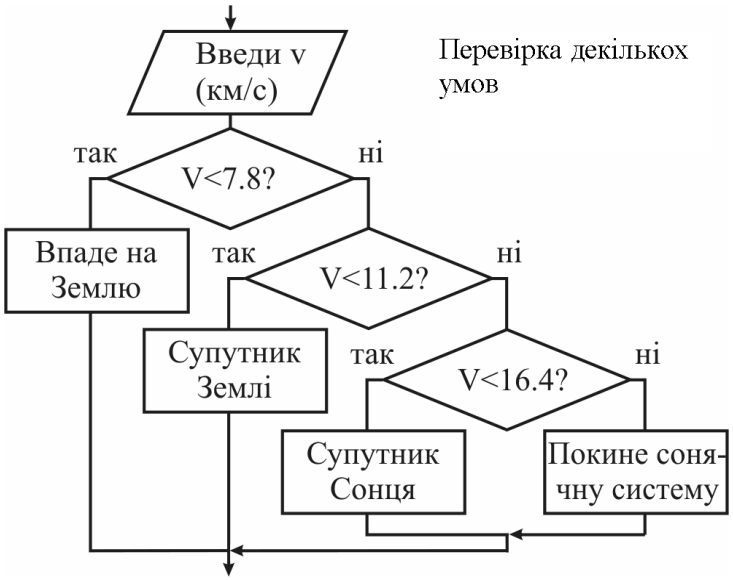
Нехай нам треба визначити поведінку космічного апарата, що стартує на екваторі, залежно від його початкової швидкості V. Як ви знаєте з уроків фізики, тут можливі чотири випадки:

- при V<7,8 км/с апарат впаде на поверхню Землі;
- при 7,8≤V<11,2 км/с апарат стане супутником Землі;
- при 11,2≤V<16,4 км/с апарат стане супутником Сонця;
- при V≥16,4 км/с апарат покине Сонячну систему.

Складемо блок-схему алгоритму (див. мал.).

Як видно з блок-схеми, ми маємо декілька вкладених одна в одну умов. Якщо не виконується перша умова, то буде перевірена друга умова і т. д. Причому, 2-гу й 3-тю умови необов'язково записувати за допомогою подвійної нерівності (наприклад, **(v>=7.8)&&(v<11.2)**). Зрозуміло, якщо перша умова (**v<7.8**) хибна, то при перевірці 2-ї умови **v** вже не може дорівнювати будь-якому значенню, меншому за 7.8:

```
#include<iostream.h>                    //Програма 2.7
#include<conio.h>
int main()
{
float v;
cout<<"Vvedit shvidkist (km/s):";
```


```
cin>>v;
if(v < 7.8)
 cout<<"Korabel upade na Zemlyu"<<endl;
else if(v < 11.2)
 cout<<"Korabel – suputnik Zemli"<<endl;
else if(v < 16.4)
 cout<<"Korabel – suputnik Soncja "<<endl;
else
 cout<<"Korabel pokine sonjachnu sistemu"<<endl;
getch();
return 0;
}
```

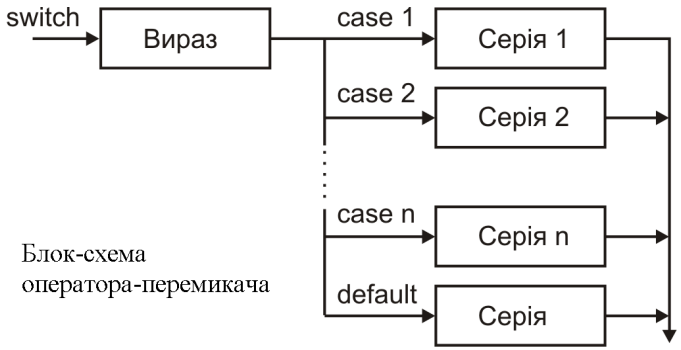
Експериментуючи з цією програмою, уводьте різні значення **v** і аналізуйте отримані результати.

Оператор switch-перемикач

У попередній програмі для з'ясування поведінки космічного корабля ми використали оператори розгалуження. Але, при глибині вкладеності цих операторів понад три, така конструкція робить програму складною для сприйняття людиною. Більш зручним у такій ситуації виявляється застосування оператора **switch**.

Загальна структура оператора така:

```
…
switch (вираз)
{
   case константний вираз 1:оператори 1;
         break;
   case константний вираз 2: оператори 2;
         break;
   …
   case константний вираз n: оператори n;
         break;
default: оператори;
}
…
```



Блок-схема
оператора-перемикача

Виконання оператора починається з обчислення виразу (він повинен бути цілочисельним), а потім керування передається першому оператору зі списку, поміченого константним виразом, значення якого співпало з обчисленим. Вихід із перемикача найчастіше виконується за допомогою оператора **break**.

Розглянемо приклад:

```
#include<iostream.h>                    //Програма 2.8
#include<conio.h>
int main()
{
  int shwid;
  cout<<"Vvedi shvidkist korablya:"<<endl;
  cout<<"0– V< 7.8 km/s"<<endl;
```



```
  cout<<"1- 1 kosmichna V=(7.8-1.2)km/s"<<endl;
  cout<<"2- 2 kosmichna V=(11.2-6.4)km/s"<<endl;
  cout<<"3- 3 kosmichna V>16.4km/s"<<endl;
  cin>>shwid;
  switch (shwid){
   case 0:cout<<"Upade na Zemlyu"<<endl;  break;
   case 1:cout<<"Stane suputnikom Zemli"<<endl;
                                                  break;
   case 2:cout<<"Stane suputnikom Soncja"<<endl;
                                                  break;
default:cout<<"Pokine sonjachnu sistemu"<<endl;
                                                  break;
      }
getch();
return 0;
}
```

До речі, мітка в **case** може бути й типу **char**:

```
…
char scor = 'a';
switch (scor){
 case 'a':cout<<"Korabel upade na Zemlyu"<<endl;
          break;
…
```

Зверніть увагу на використання оператора **break**. Якщо в програмі зустрічається варіант (**case…**), що дорівнює значенню керуючого виразу оператора **switch**, то подальша перевірка умов припиняється і виконуються всі оператори, аж до кінця оператора **switch**. Оператор **break** є вказівкою завершити поточний оператор **switch** і продовжити виконання програми з першого оператора, що є наступним за оператором **switch**.

Питання для самоконтролю:

1. *Чи можливо за допомогою умовного оператора обробляти декілька умов?*
2. *Для чого використовують оператор-перемикач?*
3. *Як працює оператор-перемикач?*
4. *За допомогою якого оператора здійснюється вихід із оператора-перемикача?*
5. *Якого типу може бути мітка в **case**?*

Вправа 2-4.

1) Випробуйте програму 2.7.
2) Випробуйте програму 2.8. Поясніть результати роботи програми. Доопрацюйте програму, щоб мітка в **case** була типу **char**.
3) Напишіть програму, яка запитує у користувача номер дня тижня, а потім виводить назву дня тижня. При введенні неправильних даних програма повинна видати відповідне повідомлення.

Збережіть програми, створивши у власній папці нову папку *wpr2-4*.

2.5. Практична робота № 5 «Використання логічних операцій та оператора-перемикача»

1) Проаналізуйте роботу програми:

```
#include<iostream.h>
#include<conio.h>
int main()
{
float v;
cout<<"Vvedite v (km/s):"; cin>>v;
cout<<"shwid="<<(v<7.8)<<"+"<<(v<11.2)<<"+"
<<(v <16.4)<<endl;
getch();return 0;
}
```

Зробіть висновки і доопрацюйте її так, щоб її взаємодія з користувачем була такою ж, як у програмі 2.7. Для підрахунку значення змінної **shwid** використайте вираз:

```
int shwid=(v < 7.8)+ (v < 11.2)+(v < 16.4);
```

2) Написати програму, що вимагає уведення часу дня і, залежно від уведеного значення, бажає доброго ранку, доброго дня, доброго вечора або спокійної ночі.
3) Написати програму для обчислення вартості розмови по телефону з урахуванням 20% знижки, що надається у суботу і неділю. Увести тариф, тривалість розмови й номер дня тижня. Проаналізувати результати при різних вхідних даних.

Збережіть програми, створивши у власній папці нову папку *pr5*.

3. Оператори для організації циклів

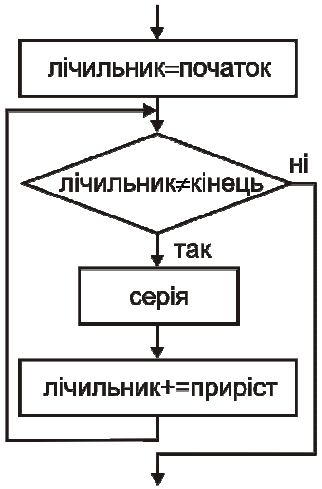
3.1. Цикли. Цикл із лічильником

Види циклів

Часто при виконанні алгоритмів необхідно багаторазово повторювати одні і ті ж дії. Наприклад, щоб зобразити ромашку, потрібно намалювати багато однакових пелюсток; щоб викопати яму, потрібно багато разів повторити подібні рухи лопатою тощо. У таких випадках використовують циклічну структуру або структуру «повторення».

Цикл *– це форма організації дій, при якій одна і та ж послідовність дій виконується кілька разів доти, поки виконується деяка умова.*

Серія команд, що повторюється без змін при кожному проході циклу (ітерації), називається **тілом циклу**.



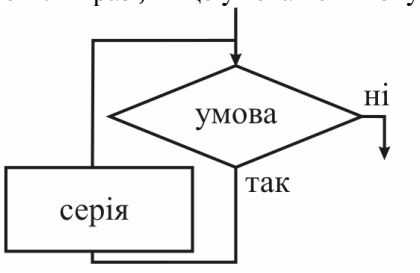
Першим розглянемо **цикл з лічильником** (див. блок-схему). Такий цикл використовується, коли заздалегідь відомо, скільки разів треба виконати тіло циклу. Лічильник набуває початкового значення, перевіряється умова, і якщо вона істинна, то виконується серія операторів, а також лічильник змінюється на величину приросту. Потім все повторюється доти, поки умова не перестане виконуватися.

Є ще два типи повторення: **з передумовою** та **з післяумовою**. Такі цикли зручно використовувати, коли заздалегідь не відомо, скільки разів буде виконуватися тіло циклу.

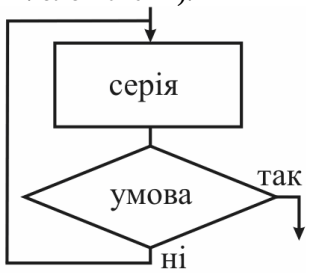
У першому випадку спочатку перевіряється умова, і якщо вона істинна, то тіло циклу виконується черговий раз, якщо ж ні – повторення серії операторів припиняється.

У випадку повторення з післяумовою, спочатку відбувається виконання вказаної дії, а після цього визначається, чи є потреба ви-

конувати її знову. Причому, в цьому випадку повторення відбувається в разі, якщо умова не виконується (див. блок-схеми).



Цикл з передумовою



Цикл з післяумовою

Можливі ситуації, коли тіло циклу з передумовою не виконається жодного разу. Це відбувається в тому випадку, коли при першій перевірці умови значення логічного виразу є хибним.

Якщо при повторенні циклу умова незмінно залишається істинною, то цикл може повторюватися нескінченно.

Цикл з післяумовою обов'язково виконається хоча б один раз, тому що спочатку виконується серія операторів, а потім перевірка умови.

Цикл з лічильником for

Розглянемо синтаксис оператора циклу з лічильником у С++:

```
for (оператор1; вираз1; вираз2)
    оператор_тіла_циклу;
```

Послідовність його виконання така:

крок 1: виконується оператор1;

крок 2: обчислюється вираз1, і якщо він істинний, то виконується оператор_тіла_циклу. Якщо хибний, то виконання циклу припиняється і виконується наступний оператор;

крок 3: обчислюється вираз2;

крок 4: повторюються кроки 2-4.

Наприклад, у програмі може зустрітися такий цикл з лічильником:

```
for (int i = nc; i <= kc; i++)
    cout<<i;
```

for – ключове слово, що означає початок циклу;

i – змінна циклу, якій присвоюється початкове значення лічильника **nc**;

kc – значення лічильника, після досягнення якого цикл завершується;

i++ – зміна лічильника циклу. Якщо крок не дорівнює одиниці, можна, наприклад, написати: **i=i+5**, чи **i=i+0.25**. Можливе скорочення запису: **i+=5** у першому випадку та **i+= 0.25** у другому. Не забувайте, що у другому випадку змінна циклу повинна мати тип **float**!

В даному випадку виконання циклу здійснюється за такою схемою:

1. Параметр **i** одержує значення **nc**.

2. Робиться перевірка, чи не перевищує значення змінної циклу кінцевого значення лічильника **kc**.

3. Якщо не перевищує, то виконуються оператори у фігурних дужках. Якщо оператор у циклі один, його можна у дужки не брати.

4. Цикл закінчує роботу, як тільки умова **i<=kc** стане хибною.

Розглянемо приклад:

```
#include<iostream.h>                    //Програма 3.1
#include<conio.h>
int main()
{
int num;
for (num = 1; num<=100; num++)
 cout<<num<<" ";
getch();return 0;
}
```

Як бачите, у цій програмі в операторі **for** змінна **num** спочатку отримує значення 1. Потім здійснюється перевірка, чи виконується умова **num<=100**. Якщо це так, то виконується відповідний оператор тіла циклу (**cout<<num<<" ";**) і **num** збільшується на 1 (**num++** або **num = num + 1**). Потім перевірка і подальші кроки повторюються.

Складемо програму, при виконанні якої комп'ютер «просить» увести число, при якому цикл повинен завершитися, а потім роздруковує всі числа від нуля до уведеного числа:

```
#include<iostream.h>                    //Програма 3.2
#include<conio.h>
int main()
{
int i,n;
cout<<"Vvedit' chislo: "; cin>>n;
```

```
for (i = 0; i <= n; i++)
  cout<<i<<" ";
getch();return 0;
}
```

Тілом циклу **for** може бути складений оператор. Наступна програма підсумує всі цілі числа від 1 до 10, вивівши при цьому покроковий коментар:

```
#include<iostream.h>                    //Програма 3.3
#include<conio.h>
int main()
{
int i,sum=0;
  for (i = 1; i <= 10; i++)
  {
   cout<<"Dodayu "<<i<<" do "<<sum;  //тіло циклу
   sum = sum + i;                    //- складений
   cout<<"- oderguyu: "<<sum<<endl;   //оператор
  }
getch();return 0;
}
```

Лічильник циклу можна не тільки збільшувати, а й зменшувати. Заголовок циклу може бути, наприклад, таким:

```
for (i = 10; i >= 1; i--)
```

Внесіть зміни у попередню програму і випробуйте її. У цьому випадку змінна циклу з кожним кроком зменшується на одиницю.

Змінимо програму так, щоб підсумовування цілих чисел від 1 до **n** (**n**>1) здійснювалося доти, поки значення суми не перевищить уведене значення **Smax.** У результаті на екран буде виведено кількість проведених операцій додавання:

```
...                                  //Програма 3.3.1
int i,Smax,n,sum=0;
int k=0;//лічильник операцій додавання
cout<<"Vvedit n:";cin>>n;
cout<<"Vvedit Smax:";cin>> Smax;
  for (i = 1; i <= n; i++)
      if(sum<=Smax)
         {
           sum = sum + i;
           k++; //приріст лічильника на одиницю
         }
```



```
cout<<" Wikonano: "<<k<<" operaciy"<<endl;
...
```

*Зверніть увагу, що між заголовком циклу **for** та тілом циклу не ставиться крапка з комою (;)!!!*

Основний цикл програми 3.3.1 можна записати ще коротше:

```
...                                  //Програма 3.3.2
for (i = 1; sum<=Smax; i++)
sum += i;
cout<<" Wikonano: "<<--i<<" operaciy"<<endl;
...
```

Як бачите, перевірка умови **sum<=Smax** тут винесена в заголовок циклу. Крім того, замість змінної **k** в кінці виводиться зменшене на одиницю значення змінної **i**. Отже, такий варіант є кращим з двох причин:

- **економія пам'яті**, завдяки відмові від змінної **k**;
- **більша швидкодія**, за рахунок зменшення кількості операцій, які повторюються в тілі циклу, з 4-х (**<=,=,+,++**) до 2-х (+=).

Особливості використання циклу for

Як вже було сказано, іноді, через помилки програмування, повторення циклу не припиняється зовсім. Тоді кажуть, що програма «зациклилася». Причиною «зациклення» є те, що умова припинення циклу не може стати істинною. Уникайте таких помилок.

Приклад зациклення:

```
for (i = 0; i < 100; value++)
  cout<<i;
```

Цей цикл мав би припинитись при досягненні змінною циклу **i** значення 100. Але ні у виразах заголовку, ні у тілі циклу значення **i** не змінюється. Як наслідок, значення змінної **i** ніколи не стане рівним 100, і програма буде працювати нескінченно!

Як вже було сказано, цикли **for** не обмежуються використанням в якості лічильника циклу змінних типу **int**. Наприклад, наступна програма використовує змінну циклу типу **char** (**letter**) для виведення букв латинського алфавіту у першому

циклі й змінну типу **float** (**value**) для виведення чисел із плаваючою крапкою в іншому циклі:

```
#include<iostream.h>                    //Програма 3.4
#include<conio.h>
int main()
{
char letter;
float value;
      for (letter = 'A'; letter<='Z'; letter++)
         cout<<letter;
  cout<<endl;
      for (value = 0.0; value<=1.0; value+=0.1)
         cout<<value<<" ";
getch();
return 0;
}
```

Питання для самоконтролю:

1. *Що таке «цикл»?*
2. *Як виконується цикл з лічильником?*
3. *Як виконується цикл з передумовою?*
4. *Чи може тіло циклу з передумовою не виконатися жодного разу?*
5. *Як виконується цикл з післяумовою?*
6. *Чи може тіло циклу з післяумовою не виконатися жодного разу?*
7. *Що таке «зациклення»?*
8. *Чи можливо у циклі **for** в якості змінних циклу використовувати змінні типу **char**; типу **float**?*

Вправа 3-1.

1) Випробуйте програму 3.1. Експериментуючи із цією програмою, змініть значення 100 на 10, 30 і навіть 4000.
2) Випробуйте програму 3.2. Що відбудеться при **n** = –1?
3) Випробуйте програму 3.3. Експериментуючи із програмою, замініть 10 іншими значеннями. Потім замість **i++** уведіть **i+=5**.

   Перероблю програму, як у прикладі 3.3.1. Експериментуйте, уводячи різні значення **n** та **Smax**.

Збережіть програми, створивши у власній папці нову папку *wpr3-1*.

3.2. Практична робота № 6 «Програми з циклом із лічильником»

1) Надрукувати всі натуральні числа від 1 до введеного з клавіатури **n** та їх квадрати у вигляді таблиці:

   1 1
   2 4
   3 9 і т.д.
2) Увести ціну 1 кг цукерок та вивести на екран таблицю вартості цукерок від 100 г до 1 кг із кроком 100 г.
3) Капосний папуга навчився висмикувати у дідуся Івана волосся, яке ще залишилось у того на голові. Почавши з однієї волосини, він кожен день збільшував порцію вдвічі. Через скільки днів дідусеві не знадобиться гребінець, якщо на початку в нього було аж N волосин.

Збережіть програми, створивши у власній папці нову папку *pr6*.

3.3. Цикл while

Цикл **while** зручно застосовувати у випадку, якщо в програмі необхідно повторювати дії, поки виконується якась умова. Заздалегідь ми не знаємо, коли вона перестане виконуватися й скільки разів виконається, відповідно, цикл. Наприклад, щоб змоделювати процес завантаження автомобіля за допомогою екскаватора, потрібно послідовно уводити значення маси чергової порції піску і перевіряти, чи не перевищена вантажопідйомність автомобіля. Кількість ітерацій такого циклу невідома, оскільки маса піску в ковші екскаватора щоразу інша, тому слід використати цикл з умовою.

Існує 2 форми написання циклу **while**:

```
while(умова)
 оператор_тіла_циклу;
```

```
do
 оператор_тіла_циклу;
while(умова);
```

У першому випадку, спочатку проводиться перевірка умови, і якщо вона істинна – виконуються оператори, що складають тіло циклу. Цикл буде виконуватися, поки умова залишатиметься істинною. При першому ж порушенні істинності умови цикл припиниться. У такій формі запису циклу **while** можлива ситуація, коли тіло циклу взагалі не буде виконане: якщо умова при першій перевірці виявиться хибною.

У другій формі запису тіло циклу обов'язково буде виконане хоча б один раз, тому що умова перевіряється після першого виконання операторів тіла циклу.

Як у першій, так і в другій формі оператор тіла циклу може бути складеним:

```
while(умова)
  {оператори}
```

```
do
  {оператори}
while(умова);
```

Вдалий вибір виду циклу (**for**, **while** чи **do…while**) робить програму зрозумілішою, а отже зменшує ймовірність помилки.

Розглянемо приклади.

Алгоритм Евкліда. Алгоритм знаходження найбільшого спільного дільника двох натуральних чисел m і n (НСД(m,n)) був описаний в III столітті до н.е. в класичному трактаті «Початки» грецького математика Евкліда.

Розв’язок можна одержати шляхом послідовного віднімання меншого числа від більшого доти, поки m≠n:

крок 1: порівняти m і n; якщо m=n, то НСД(m,n)=n, інакше крок 2;
крок 2: визначити більше з чисел;
крок 3: відняти від більшого числа менше. Отриманою різницею замінити більше число;
крок 4: перейти до кроку 1.

Для пари натуральних чисел, наприклад 39 і 12, це виглядає так:

39,12→27,12→15,12→3,12→3,9→3,6→3,3 ← рівні

39–12 — 12–3

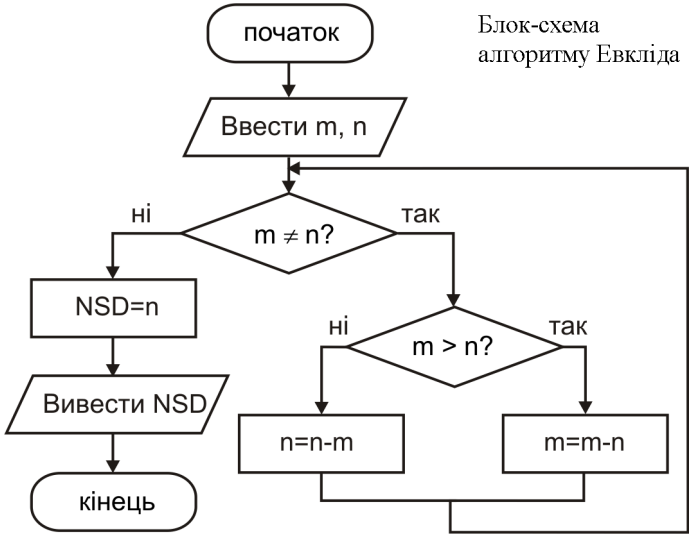
Зі схеми видно, що НСД(39,12)=3.

Використаємо відомі нам оператори С++:

поки числа не стануть рівними,	`while(m!=n)`

більше число замінюватимемо різницею більшого і меншого	`if(m>n)`
	`  m=m-n;`
	`else`
	`  n=n-m;`

У результаті одержимо пару однакових чисел **m** і **n**, які й дорівнюють найбільшому спільному дільнику.



Блок-схема алгоритму Евкліда

```
#include<iostream.h>                    //Програма 3.5
#include<conio.h>
int main()
{
int m,n,NSD;
cout<<"Vvedi m,n:";cin>>m>>n;
   while(m!=n)
   {
     if(m>n)
       m=m-n;
     else
       n=n-m;
   }
   NSD=n;
   cout<<"NSD="<<NSD<<endl;
getch();
return 0;
}
```

***Приклад 2**. Увести послідовність додатних та від'ємних чисел, що закінчується нулем, і визначити суму додатних:*

```
#include<iostream.h>                    //Програма 3.6
#include<conio.h>
int main()
```

```
{
int k, S=0;
   do
   {
      cout<<"Vvedi k:";cin>>k;   //уведення даних
      if(k>0)                   //відбір додатних
           S=S+k;                //підсумовування
   }
   while(k!=0);           //перевірка закінчення
cout<<"Summa dodatnih S="<<S<<endl;
getch();
return 0;
}
```

У цій програмі цикл використовується як для підсумовування даних, так і для перевірки закінчення уведення даних (при **k**=0).

Приклад 3. *Підрахувати* **k** *– кількість цифр у десятковому записі цілого невід'ємного числа* ***n***.

Поділимо шукане число націло на 10, потім поділимо отриманий цілий результат на 10, і так доти, поки не одержимо в результаті нуль. При цьому змінна-лічильник **k**, збільшуючись після кожного ділення, лічить кількість ітерацій, яка й дорівнює розрядності числа **x**. У даній програмі для вирішення поставленого завдання цикл повинен виконатися хоча б один раз, тому скористаємось формою **do…while**.

```
#include<iostream.h>                 //Програма 3.7
#include<conio.h>
int main()
{
int n;                                        //число
int k=0;                                  //лічильник
cout<<"Vvedi cile dodatne chislo:";cin>>n;
int ost=n;                          //цілий результат
   do
   {
     ost=ost/10;
     k=k+1;
   }while(ost!=0);
cout<<k<<"-cifrove"<<endl;
getch();return 0;
}
```


Вкладені цикли

Цикли можуть бути вкладеними один в одний, тобто тіло циклу може включати в собі оператор циклу. Кількість вкладених циклів (ще кажуть: «глибина вкладення») не обмежена.

Приклад. *Вивести на екран прямокутник, заповнений символами «*»:*

```
********
********
******** і т.д.
```

```
#include<iostream.h>                 // Програма 3.8
#include<conio.h>
int main()
{
int i,j,n,m;
cout<<"Rjadkiv: ";cin>>n;
cout<<"Stovpciv: ";cin>>m;
for (i = 0; i < n; i++)
{
   for (j = 0; j < m; j++)
     cout<<"*";
  cout<<endl;
}
getch();return 0;
}
```

У цій програмі зовнішній цикл використовує змінну циклу **i**, а внутрішній – змінну циклу **j**. Для кожного значення **i** внутрішній цикл виконується **m** разів. Таким чином, оператор виведення буде друкувати у рядку **m** символів «*****», а потім, при переході до наступного значення змінної **i**, завдяки оператору **cout<<endl;** буде здійснюватись перехід на новий рядок. Це повторюватиметься доки, поки змінна зовнішнього циклу не досягне значення **n**.

Щоб не заплутатися у вкладених циклах, обов'язково використовуйте табуляцію при написанні програми! Це зробить вашу програму зрозумілішою і спростить пошук помилок.

Питання для самоконтролю:

1. *У яких випадках зручно використовувати цикл* ***while****?*
2. *Назвіть дві форми використання циклу* ***while****?*
3. *Виконайте усно алгоритм Евкліда для пари чисел 24 і 16?*
4. *Для чого використовують змінну-лічильник?*
5. *Чи може використовуватися цикл у якості оператора в тілі іншого циклу?*
6. *Для чого використовують табуляцію при написанні програм?*

Вправа 3-3.

1) Випробуйте програму 3.5. для **m**=39, **n**=12. Доповніть її так, щоб були виведені проміжні результати обчислень (як, наприклад, у програмі 3.3).
2) Випробуйте програми 3.6. та 3.7. Чому при введенні більше ніж 10-цифрових чисел програма 3.7. працює неправильно?
3) Випробуйте програму 3.8. Виведіть замість символу «*» своє ім'я.

Збережіть програми, створивши у власній папці нову папку *wpr3-3*.

3.4. Практична робота № 7 «Програми з повтореннями»

1) На дверях ліфта висіло загрозливе попередження про те, що двері зачиняються самі в той момент, коли зайвий за вагою пасажир переступить поріг ліфта. Який за рахунком пасажир спричинить аварію, якщо ліфт витримує вагу не більше S кг, а вага кожного з n пасажирів, що стоять у черзі до ліфта, дорівнює, відповідно, a_1, a_2, …, a_n ?
2) Обчислити факторіал цілого числа, уведеного з клавіатури. Дослідити, в якому діапазоні вхідних даних програма працює правильно.

Примітка. Факторіалом числа n називається добуток усіх натуральних чисел від 1 до n. Записується це так: n!=1*2*3…* n. Наприклад: 4!=1*2*3*4=24.

3) Увести послідовність чисел, що закінчується нулем, і визначити найбільше число в ній.

Збережіть програми, створивши у власній папці нову папку *pr7*.

3.5. Тематичне оцінювання з теми «Оператори повторення та розгалуження»

1) Вивести на екран заповнений символами «*» трикутник:

```
*
**
***
і т.д.
```

2) Надрукувати на екрані таблицю Піфагора.
3) Задані 3 цілих додатних числа: a, b, c. Визначити, чи можна з відрізків з такими довжинами утворити трикутник.

Збережіть програми, створивши у власній папці нову папку *ta2*.

Питання для самоконтролю:

1. *Яких типів повинні бути вирази, що стоять у круглих дужках операторів* ***if, while, do...while****?*
2. *Як виконати більше одного операторар в циклі або якій-небудь гілці обчислень умовного оператора?*
3. *Якого типу повинні бути вирази, що стоять в дужках після ключового слова* ***switch****, і константні вирази в* ***case****?*
4. *У якому випадку припиняється виконання циклу* ***for****?*
5. *У якому випадку припиняється виконання циклу* ***while****?*
6. *У якому випадку припиняється виконання циклу* ***do… while****?*
7. *Тіло якого з циклів обов'язково виконується хоча б один раз?*
8. *Який оператор використовується для передачі керування на кінець оператора* ***switch****?*
9. *Скільки разів виконається цикл* ***for*** *(i=a; i<=b; i++), якщо:*
   а) b >= a – 1;
   б) b < a.
10. *Скільки разів виконається цикл* ***for*** *(i=a; i<=b; i--), якщо:*
    а) a >= b – 1;
    б) a < b.
11. *Що буде надруковано в результаті виконання такого фрагменту програми:*

```
int y=1;
int i=2;
while (i<=5) {
      y*=i;
      i++;
}
cout<<y;
```

12. *Замінити у цьому фрагменті цикл* ***while*** *циклом:*
    а) for… ;
    б) do… while.

4. Функції

4.1. Функції у С++. Локальні і глобальні змінні

Функція являє собою цілком самостійний блок програми. Як правило, вона одержує певні дані при виклику й повертає якесь значення. Щоб викликати функцію, потрібно вказати її ім'я і, за потреби, в дужках – вхідні дані, відокремлені комами. Це означає, що якщо в програмі зустрівся рядок

a = Name(23,'y');

то буде викликана функція **Name()** з відповідними параметрами (число 23 та символ 'y'), а по завершенні роботи функції змінній **a** буде присвоєне значення, повернуте цією функцією.

Застосування функцій доцільне тоді, коли певна дія, або послідовність дій повинна повторюватися в різних частинах програми. Наприклад, у С++ немає стандартної функції для обчислення площі круга, а розв'язувана задача вимагає кількаразового обчислення цієї площі для різних радіусів. У такому випадку потрібно написати функцію, що одержує один вхідний параметр – радіус круга, а повертає його площу. Після цього, замість повторення послідовності команд у програмі, досить лише викликати функцію.

Іншою причиною використання функцій може стати великий обсяг програми. Тоді, навіть за відсутності повторюваних ділянок коду, використання функцій може стати доцільним з метою поліпшення наочності програми й для спрощення її налагодження.

Розглянемо приклади.

Приклад 1. *Написати функцію, що отримує один параметр – натуральне число, міняє порядок цифр у цьому числі на протилежний і повертає отриманий результат.*

```
#include<iostream.h>                    //Програма 4.1
#include<conio.h>
int Mir(int a)                                  //Функція
{
int res=0;
        while(a>0)
        {
         res=res*10+a%10;
```



```
         a/=10;
        }
return res;
}
//Головна програма
int main()
{
int Num;
        do {
        cout<<"Vvedi naturalne chislo:";
        cin>>Num;
        } while (Num < 0);
int res = Mir(Num);                    //Виклик функції
cout<<"Dzerkalo = "<<res;
getch();return 0;
}
```

Розробку програми зручно почати зі створення необхідної функції, яку назвемо **Mir** (англ. mirror – дзеркало). Яким має бути тип функції, тобто тип значення, яке вона повертає? В умові сказано, що функція отримує натуральне число, міняє порядок цифр у ньому, а потім повертає отриманий результат. Звідси можна зробити висновок, що повертатиме функція ціле число. Отже тип функції буде **int**. З тієї ж причини список параметрів нашої функції буде містити тільки один параметр цілого типу (**int a**): ніяких інших додаткових даних, крім вихідного числа, нам не потрібно. Отже, заголовок нашої функції виглядає так: **int Mir(int a)**.

Далі визначаємо тіло функції, тобто інструкції, які необхідно виконати для досягнення поставленої мети (зміна порядку цифр у заданому числі). Для цього використаємо цикл **while**, який виконується, поки задане число залишається більшим від нуля. У цьому циклі «старе» значення змінної-результату **res** множимо на 10 і додаємо до нього останню цифру заданого числа, знаходячи її як остачу від ділення на 10. А далі ділимо задане число на 10, тобто відкидаємо його останню цифру.

Після першої ітерації циклу змінна-результат стане рівною останній цифрі заданого числа. Під час другої значення результату (тобто остання цифра вихідного числа) множиться на 10 і до неї підсумується передостання цифра, отримана діленням заданого числа на 10. І так далі, поки після ділення на 10 заданого числа не

одержимо 0 (тобто не переберемо всі цифри вихідного числа). На цьому цикл закінчується, а у змінній результату одержимо задане число зі зміненим порядком цифр.

По закінченні циклу виконується інструкція **return**, яка повертає значення отриманого результату в головну функцію.

Як видно з тексту програми, в головній функції **main** пропонуємо користувачеві увести додатне число із клавіатури, (у програмі передбачена обробка тільки додатних чисел) після чого викликаємо функцію **Mir** і передаємо їй параметр – уведене користувачем число. Зверніть увагу, що цикл **do {…} while (Num < 0);** забезпечує перевірку коректності уведених даних. Приймає повернуте функцією значення змінна **res** (тобто ця змінна буде зберігати число із цифрами у зворотному порядку). Наступною командою результат буде виведений на екран. Програма завершується.

Приклад 2. *Написати функцію, що отримує ціле число як параметр, і якщо число дорівнює 1, то повертає символ «**p**», якщо -1 – повертає символ «**n**», в іншому випадку виводить повідомлення про помилку й повертає символ «**0**».*

```
#include<iostream.h>                    //Програма 4.2
#include<conio.h>
char Test (int a)
{
switch (a)
 {
  case 1 :return 'p';
  case -1:return 'n';
  default:cout<<"Error!";
  return '0';
 }
}
int main()
{
int  Num;
cout<<"Vvedi cile chislo(1,-1, abo inshe): ";
cin>>Num;
cout<<"Rezultat: "<<Test(Num);
getch();return 0;
}
```


Розробку почнемо зі створення функції, яку назвемо **Test**. Наша функція у всіх випадках повинна повертати символ (це видно з формулювання завдання), тому обираємо тип функції **char**. Список параметрів функції буде містити тільки один параметр типу **int**, тому що за умовою ніяких інших додаткових вхідних даних для функції не потрібно. Таким чином, заголовок функції буде виглядати так: **char Test (int a)**.

У функції, для перевірки значення змінної **a**, що отримане як параметр, використаємо оператор **switch**. Якщо значення змінної **a** дорівнює 1, то будуть виконані інструкції, що йдуть після першого префікса **case**, тобто за допомогою оператора **return** в головну функцію буде переданий символ «**p**». Зверніть увагу, що наприкінці набору інструкцій немає оператора **break**, який мав би переривати виконання оператора **switch**. Він опущений, тому що оператор **return** перериває роботу функції й, відповідно, виконання оператора **switch** також перерветься.

Аналогічно виконується й частина програми, позначена другим префіксом **case** у випадку, коли значення змінної **a** дорівнює -1. Якщо значення змінної **a** не дорівнює 1 і не дорівнює -1, то будуть виконуватися інструкції після префікса **default**, тобто на екран буде виведене повідомлення про помилку й оператор **return** поверне в головну функцію символ «**0**».

Головна ж функція **main** надає користувачеві можливість уведення із клавіатури цілого числа. Потім викликається функція **Test**, що повертає значення, яке й виводиться на екран з відповідним повідомленням за допомогою **cout**. Після чого виконання програми завершується.

Аргументи та значення функцій

Як ви вже знаєте, функції можуть повертати значення в основну програму. Щоб повернути значення, використовується оператор **return**:

```
return c;
```

В операторі **return** може бути записаний також вираз:

```
return (a+b)/2;
```

Але бувають функції, які не повертають значення. Імена таких функцій позначають типом **void**:

```
void Func(int a, int b)
```

В таких функціях оператор **return** не використовують.

Слід також зауважити, що функція може не використовувати вхідних аргументів. Щоб вказати на це, замість їх переліку теж пишуть **void**:

```
int MyFunct (void)
{…}
```

Локальні і глобальні змінні. Область видимості

У C++ використовують термін **область видимості змінної**, що визначає частину програми, де ім'я змінної має сенс (отже, може бути використане).

Локальна змінна являє собою змінну, оголошену всередині функції до її використання. Область її видимості: від точки оголошення до кінця даної функції. Досі у своїх програмах ви використовували саме локальні змінні.

При оголошенні локальних змінних всередині функції дуже імовірно, що ім'я локальної змінної, оголошеної вами в одній функції, буде таким же, як й ім'я змінної, використовуваної в іншій функції. Але завдяки тому, що області дії цих змінних обмежені, це не призводить до **конфлікту**. C++ трактує ім'я кожної змінної як локальне стосовно відповідної функції.

C++ дозволяє у ваших програмах використовувати **глобальні змінні**, область видимості яких – від точки оголошення до кінця програми (глобально для всіх функцій). Оголошувати глобальну змінну слід поза усіма функціями (зокрема, поза функцією **main**):

```
float global_var;                    //глобальна змінна
int main()
{
// оператори програми
}
```

У наступній програмі використовується глобальна змінна з ім'ям **num**. Кожна функція в програмі, описана нижче від місця оголошення глобальної змінної, може використати (або змінити) її значення:

```
#include<iostream.h>                    //Програма 4.3
#include<conio.h>
int num = 1;                         //Глобальна змінна
void one (void)
{
cout<<"num u one ="<<num<<endl;
num++;
}
void two (void)
{
cout<<"num u two ="<<num<<endl;
num++;
}
int main()
{
cout<<"num u main="<<num<<endl;
num++;
one();
two();
getch();
return 0;
}
```

Результат роботи програми:

```
num u main=1
num u one =2
num u two =3
```

Як правило, у програмах варто уникати використання глобальних змінних.

Оскільки будь-яка функція може змінити значення глобальної змінної, це може призвести до помилок у програмі, які складно потім виявити. Крім того, функцію, яка не посилається на глобальні змінні, зручно використовувати в інших програмах: для цього досить скопіювати її текст.

Якщо ваша програма використовує глобальну змінну і її ім’я співпадає з ім'ям локальної змінної, то C++ надає пріоритет локальній змінній.

Однак можуть бути ситуації, коли вам необхідно звернутися до глобальної змінної, чиє ім'я конфліктує з ім'ям локальної змінної. У таких випадках у програмі застосовують операцію дозволу області видимості (**::**), яка означає, що слід використати глобальну змінну.

Припустимо, що у вас є глобальна й локальна змінні з ім'ям **num**. Якщо ваша функція використовує локальну змінну **num**, вона просто звертається до цієї змінної, як показано нижче:

```
num = 1; // Звертання до локальної змінної
```

З іншого боку, якщо з функції треба звернутися до глобальної змінної, пишемо:

```
::num = 2; // Звертання до глобальної змінної
```

Розглянемо приклад:

```
#include<iostream.h>                    //Програма 4.4
#include<conio.h>
int num = 1;                    //Глобальна змінна num
void show_num(int num)          //Локальна змінна num
{
cout<<"Local num="<<num<<endl;
cout<<"Global num="<<::num<<endl;
}
void main()
{
int val = 2;
show_num(val) ;
getch();
}
```

Результат роботи:

```
Local num=2
Global num=1
```

Як бачите, у програмі можна передбачити вибір глобальної або локальної змінної за допомогою операції дозволу області видимості. Однак, як ви, напевно, помітили, використання глобальних і локальних змінних може викликати плутанину, що у свою чергу може призвести до помилок. Тому в міру можливого уникайте використання глобальних змінних та змінних з однаковими іменами.

Питання для самоконтролю:

1. *Для чого використовують функції?*
2. *Яким може бути ім'я функції?*
3. *Де описуються аргументи функції?*
4. *За допомогою якого оператора функція повертає результати?*
5. *Що таке «локальна змінна»?*
6. *Де описується локальна змінна?*
7. *Що таке «глобальна змінна»?*
8. *Де описується глобальна змінна?*
9. *Чому не рекомендується використовувати глобальні змінні?*
10. *Що таке конфлікт імен?*
11. *Якій змінній надається пріоритет при конфлікті імен? Чи можна засобами C++ змінити ситуацію?*
12. *Які змінні мають більшу область видимості – локальні чи глобальні?*

Вправа 4-1.

1) Випробуйте програму 4.1. Перевірте роботу програми для 2-, 3- та 6-цифрових натуральних чисел.
2) Випробуйте програму 4.2. Перевірте роботу програми для Num=1,-1, та будь-якого іншого.
3) Випробуйте програми 4.3.–4.4. У чому різниця між локальними і глобальними змінними?
4) Напишіть програму для визначення об'єму циліндра, де функція отримує, як параметри, радіус і висоту циліндра ($V=2\pi rh$, $\pi=3.14159$).

Збережіть програми, створивши у власній папці нову папку *wpr4-1*.

4.2. Виведення українських літер. Прототипи функцій. Випадкові числа

Нижче наведена функція, що дозволяє використовувати українську мову у повідомленнях, які виводить програма. Будемо сприймати цю функцію формально, не з'ясовуючи тонкощів її роботи. Для нас важливий кінцевий результат використання цієї функції.

```
char bufUkr[256];
char*Ukr(const char* text)
{
CharToOem(text,bufUkr);
return bufUkr;
}
```

Включивши цю функцію у свою програму і підключивши бібліотеку <*windows.h*>, ви зможете використати український алфавіт. Фактично для обробки символів функція **`Ukr`** викликає функцію **`CharToOem()`** зі згаданої бібліотеки.

```
#include<iostream.h>                    //Програма 4.5
#include<conio.h>
#include<windows.h>
// функція для використання української мови
char bufUkr [256];
char*Ukr(const char* text)
{
CharToOem(text, bufUkr);
```

```
return bufUkr;
}
// кінець функції
int main()
{
int num;
cout<< Ukr("Уведіть улюблену оцінку:");
cin >>num;
cout<< Ukr ("Улюблена оцінка - ")<<num<<endl;
getch();
return 0;
}
```

Як бачимо, для звернення до цієї функції користуються її ім'ям, передаючи текстовий рядок, як аргумент: **Ukr ("текст")**. Для використання української літери «і», увімкніть на клавіатурі англійську розкладку і уведіть англійську літеру «і». З літерами «ї» та «є» проблем нема.

Використання функцій

Приклад 1. *Повернуте значення можна присвоїти змінній.*

Ця програма використовує функцію для знахождения середнього арифметичного двох чисел. Повернуте значенння присвоюється змінній **z**.

```
#include<iostream.h>                    //Програма 4.6
#include<conio.h>
float sred(float a,float b)
{
        float c=(a+b)/2;
        return c;
}
int main()
{
        float x=3;
        float y=5.5;
        float z=sred(x,y);     // Виклик функції
        cout<<"z="<<z;
getch();return 0;
}
```


Приклад 2. *Повернуте значення можна вивести, застосовуючи* ***cout****:*

```
 …
float x=3, y=5.5;
cout<<"z="<<sred(x,y);          // Виклик функції
 …
```

Приклад 3. *Повернуте значення можна використати у формулюванні умови:*

```
if(sred(x,y)>= 10)
```

Прототипи функцій

Перш ніж у програмі з'явиться виклик функції, компілятор повинен «знати» тип значення функції, а також кількість і типи параметрів, використовуваних функцією. У розглянутих дотепер програмах описи функцій передували головній функції, в якій вони використовувались. У більш складних програмах така закономірність може порушитися і виникнуть помилки компіляції.

Щоб цього не трапилося, рекомендується оголошувати функції. Оголошення функції повинне розміщуватися в тексті раніше від її першого виклику для того, щоб компілятор міг здійснювати перевірку правильності виклику. Оголошення здійснюється написанням **прототипу** (заголовку) функції на початку вашої програми, після опису підключених бібліотек. Прототип включає назву функції, тип значення, що повертає функція, та типи даних, які одержує функція при виклику. Завершується написання прототипу крапкою з комою:

```
#include<iostream.h>
float sred(float ,float );

int main ()
{
    float x=3;
    float y=5.5;
    float z=sred(x,y);
    cout<<"z="<<z;
    return 0;
}
```

```
float sred(float a,float b)
{
    float c =(a+b)/2;
    return c;
}
```
функція, оголошена прототипом

Випадкові числа

Заголовковий файл <*stdlib.h*> містить функцію **random(x)**, яка повертає випадкове ціле число в діапазоні від 0 до x. Перед першим звертанням до функції **random** рекомендується викликати функцію **randomize()**, яка ініціалізує генератор випадкових чисел. Якщо цього не зробити, то при кожному запуску програми будемо мати однакову послідовність випадкових чисел:

```
#include<iostream.h>                    //Програма  4.7
#include<conio.h>
#include<stdlib.h>
int main()
{
randomize();
for(int i=0;i<10;i++)
 cout<<random(999)<<"  ";
getch();return 0;
}
```

Випадкові числа широко використовуються при написанні тестуючих та ігрових програм. Наприклад, програма 4.8 може бути використана для вивчення таблиці множення молодшими школярами:

```
#include<iostream.h>                    //Програма  4.8
#include<conio.h>
#include<stdlib.h>
int main()
{
randomize();
int a,b,vidp;
for(int i=0;i<12;i++)
{
  a=random(9);b=random(9);
  cout<<a<<"x"<<b<<"=";cin>>vidp;
  if(vidp==a*b)
        cout<<"dobre!"<<endl;
```



```
  else
        cout<<"pogano!"<<endl;
}
getch();return 0;
}
```

Питання для самоконтролю:

1. *Чи може функція не повертати ніякого значення?*
2. *Що таке прототип функції?*
3. *Чи можливо при використанні прототипу розташовувати функцію у будь-якому місці програми?*
4. *Для чого використовуються випадкові числа?*

Вправа 4-2.

1) Випробуйте програми 4.5. та 4.6.
2) Випробуйте програму 4.7. Запустіть її декілька разів. Порівняйте результати. Вилучіть з програми команду **randomize ()**. Запустіть програму кілька разів і порівняйте результати різних запусків. Зробіть висновок про призначення функції **randomize ()**.
3) Випробуйте програму 4.8. Доопрацюйте її так, щоб кожна правильна відповідь додавала один бал, а результатом тестування була оцінка за 12-бальною шкалою. Українізуйте програму.

Збережіть програми, створивши у власній папці нову папку *wpr4-2*.

4.3. Вказівники. Адреси змінних

Зміна значень параметрів функції

Використання функцій дозволяє розділити ваші програми на невеликі, легко керовані частини. Але при звичайному використанні функція не може змінити значення змінної-параметра у тій функції, яка викликала дану. Розглянемо приклад:

```
#include<iostream.h>                    //Програма  4.9
#include<conio.h>
void fun(int, int);                   //Прототип функції
int main()
{
int big=2002, small=0;
cout<<"Do funkcii:"<<big<<"  "<<small<<endl;
```

```
fun(big,small);
cout<<"Pislya funkcii:"<<big<<"  "<<small<<endl;
getch();return 0;
}
void fun(int a,int b)
{
a=1001;b=1001;
cout<<"U funkcii:"<<a<<"  "<<b<<endl;
}
```

Результат виконання програми:

```
Do funkcii: 2002  0
U funkcii: 1001  1001
Pislya funkcii: 2002   0
```

У результаті ми одержимо в результаті ті ж значення big і small, які вони мали перед викликом функції fun, незважаючи на їх зміну в тілі функції. Це пов'язане з тим, що при виклику функції у

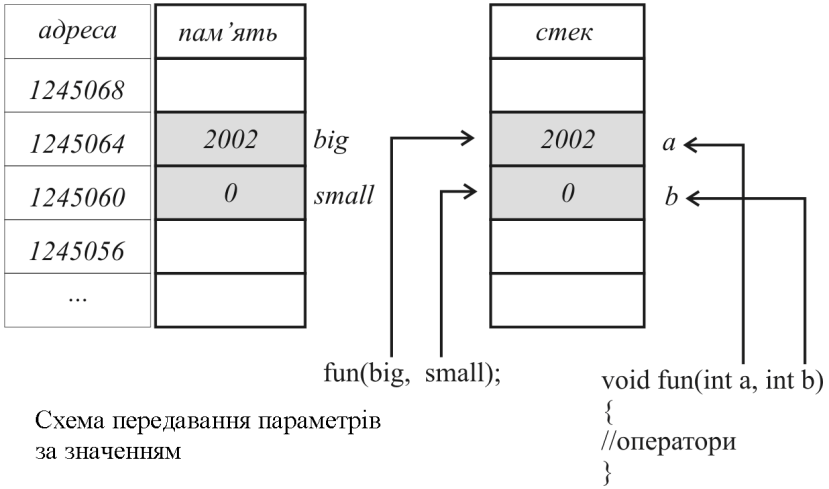


Схема передавання параметрів за значенням

спеціальну ділянку пам'яті – стек – були вміщені копії значень параметрів. Саме з ними (тобто змінними a та b) були виконані операції під час роботи функції. Після закінчення роботи функції їх значення стерлися зі стеку, а значення змінних big і small, які містили вихідні дані, так і не змінилися (дивись малюнок «Схема передавання параметрів за значенням»).

Щоб змінити зовнішній параметр, функція повинна отримати адресу комірки пам'яті, де він розміщений. Для передавання у

функцію адреси змінної-параметра використовується **операція одержання адреси (&).** Наведений нижче виклик функції ілюструє, як програма буде використовувати операцію одержання адреси, щоб передати адреси змінних **big** і **small** у функцію **fun**:

```
fun(&big,&small);
```

У свою чергу в заголовку функції **fun** потрібно вказати, що параметри будуть передані за допомогою адреси. Для цього слід оголосити їх як **змінні-вказівники**, випереджаючи ім'я кожної змінної зірочкою:

```
void fun(int*a,int*b)
```

Змінна-вказівник містить адресу пам'яті відповідної змінної.

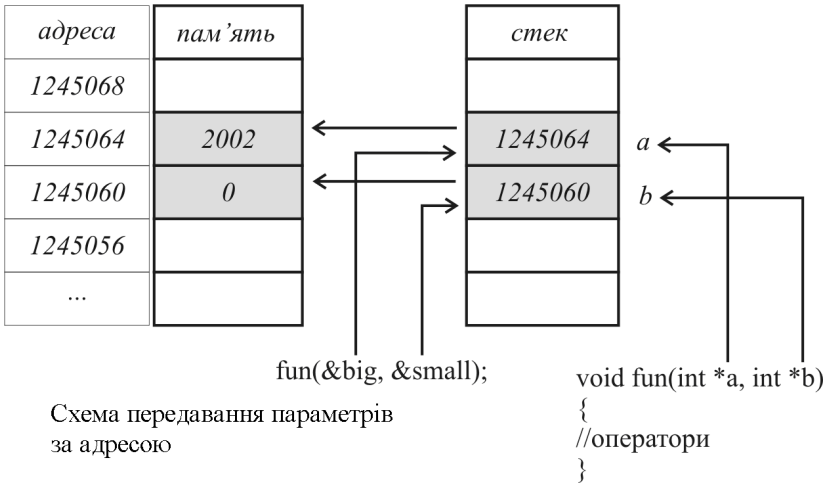


Схема передавання параметрів за адресою

Всередині функції потрібно зазначити C++, що функція працює з адресою параметра:

```
*a=1001; *b=1001;
```

Функція має доступ до комірки пам'яті кожної із змінних. Якщо параметри передаються за адресою, C++ поміщає адресу кожної змінної в стек (дивись малюнок «Схема передавання параметрів за адресою»). Використовуючи вказівники (адреси пам'яті) всередині функції, **fun** може звернутися до пам'яті за адресою кожного параметра, змінюючи значення параметрів, що й потрібно:

```
#include<iostream.h>              //Програма  4.10
#include<conio.h>
void fun(int*, int*);          //Прототип функції
int main()
{
int big=2002, small=0;
cout<<"Do funkcii:"<<big<<"  "<<small<<endl;
fun(&big,&small);//Передача параметра за адресою
cout<<"Pislya funkcii:"<<big<<"  "<<small<<endl;
getch();return 0;
}
void fun(int *a,int *b)
{
*a=1001;*b=1001;
cout<<"U funkcii:"<<"a="<<a<<"
"<<"b="<<b<<endl;
cout<<"U funkcii:"<<"*a="<<*a<<"
"<<"*b="<<*b<<endl;
}
```

Результат виконання програми:

```
Do funkcii: 2002  0
U funkcii: a=1245064   b=1245060
U funkcii: *a=1001  *b=1001
Pislya funkcii: 1001   1001
```

Особливості використання вказівників

Розглянемо вказівники докладніше. На відміну від змінних різних типів, які для зберігання даних мають різну довжину, вказівники на різні типи займають однакову кількість байт – 2 або більше. Скільки саме, залежить від того, з якою областю пам'яті працює програма. У два байти, як відомо, не можна записати число, більше ніж 65535, а отже, всі дані, з якими працює програма, повинні вміститися в область такого розміру. Якщо ця область більша, вказівники будуть займати 4, 8 або 16 байт.

Вказівник не є самостійним типом, він завжди пов'язаний з яким-небудь іншим конкретним типом.

При оголошенні вказівника зірочка (*****) дає компілятору зрозуміти, що це саме вказівник. Наприклад:

```
int *a;            //a - вказівник на цілу змінну
double *t;      //t - вказівник на дійсну змінну
                                  //подвійної точності
```


Зірочка ставиться безпосередньо до імені. Тому, щоб оголосити кілька вказівників, потрібно ставити її перед ім'ям кожного з них. Наприклад:

```
int *a, b, *c;       //a й c - вказівники на цілі
                  //змінні, b - змінна цілого типу.
```

За допомогою операції взяття адреси (**&**) можна одержати адресу будь-якої змінної. Наприклад:

```
int number=88;
int *a=&number; //у вказівнику a зараз перебуває
                          //адреса змінної number.
```

Можна подивитися, що перебуває у змінній, на яку вказує вказівник:

```
int number=5;
int *a = &number;
cout<<number;        //виводить змінну number (5)
cout<<*a;              //значення змінної, на яку
                                  //вказує a - теж 5
int cool=*a*number;      //cool=5 у квадраті(25)
```

Далі ви побачите, що у мові C++ вказівники є важливим інструментом для написання ефективних програм.

Питання для самоконтролю:

1. *Чому при звичайному використанні функція не може змінити значення параметру?*
2. *Де при звичайному використанні функції виконуються дії над змінними?*
3. *Як одержати адресу змінної?*
4. *Що таке «змінна-вказівник»? Як оголошується вказівник?*
5. *Де відбуваються дії над змінними при передачі параметра за адресою? Що при цьому зберігається у стеку?*

Вправа 4-3.

1) Випробуйте програму 4.9.
2) Випробуйте програму 4.10. Зробіть висновки.
3) Переробіть функцію з програми 4.10. так, щоб вона згідно з вивченим у п.1.2. (приклад 3) алгоритмом обмінювала значення змінних **big** і **small**.

Збережіть програми, створивши у власній папці нову папку *wpr4-3*.

4.4. Практична робота № 8 «Складання програм з використанням функцій»

1) Написати функцію для обчислення об'єму циліндра. Параметрами функції повинні бути радіус і висота циліндра.
2) Написати функцію, що повертає більше з двох цілих чисел. Використовуючи цю функцію, написати програму для знаходження більшого з трьох чисел.
3) Написати функцію для обчислення факторіалу (див. практичну роботу № 7) та програму, що використовує цю функцію для виведення таблиці факторіалів чисел заданого інтервалу.

Збережіть програми, створивши у власній папці нову папку *pr8*.

4.5. Тематичне оцінювання з теми «Функції»

1) Баба Яга записалася на курси водіїв літальних апаратів. Але справи в неї були кепські, бо вона ніяк не могла запам'ятати, яким чином визначається тривалість польоту при відомій швидкості і відстані. Довелося їй звернутися до юного програміста Хлопчика-Мізинчика, який швиденько написав їй програму для бортового комп'ютера мітли, куди Бабі Язі треба було лише підставити свої значення. Як виглядала ця програма і як нею користувалася Баба Яга?
2) Задані два натуральних числа. Визначити в якому з них більше цифр, організувавши функцію для підрахунку кількості цифр в будь-якому натуральному числі.

Питання для самоконтролю:

1. *У чому різниця між оголошенням прототипу функції і визначенням функції?*
2. *Як оголосити функцію, яка не повертає значення?*
3. *Що таке локальна змінна? Що таке глобальна змінна?*
4. *Що таке область видимості?*
5. *Коли функція не може змінити значення параметра?*
6. *Що потрібно використовувати, щоб функція змінила значення параметра?*

5. Масиви

5.1. Поняття масиву. Опис та ініціалізація масиву

Масив *– це сукупність елементів* ***одного типу****, звернення до яких здійснюється за допомогою імені масиву та індексу (тобто порядкового номера елемента).*

Елементи масиву пронумеровані. Завдяки нумерації можна звернутися до будь-якого елемента масиву як до простого значення базового типу. У С++ нумерація елементів починається з нуля.

У таблиці (див. малюнок) зображений масив цілих чисел, названий **A**. Цей масив містить 5 елементів. Таким чином, перший елемент масиву **A** указують як **A[0]**, другий елемент – як **A[1]**, п'ятий – як **A[4]** тощо. Тим самим – імена масивів повинні задовольняти тим самим вимогам, які ставляться до інших імен змінних. Отже, зображений масив міг бути заповнений у такий спосіб: **A[0]= -4; A[1]= 3; … A[4]= -39**

i	0	1	2	3	4
A[i]	-4	3	2	7	-39

Як ви вже знаєте, номер позиції, зазначений всередині квадратних дужок, називається індексом. Індекс повинен бути цілим додатним числом або математичним виразом, результатом обчислення якого є ціле додатне число. Якщо в якості індексу записаний математичний вираз, то вираз обчислюється з метою визначення індексу.

Приклад 1*. Якщо змінна* **n** *дорівнює 3, а змінна* **m** *дорівнює 1, то оператор* **A[n + m] += 7;** *додає 7 до елемента масиву* **A[4]**.

Приклад 2*. Надрукувати суму перших трьох елементів масиву* **A** *можна в такий спосіб:*

```
cout << A[0] + A[1] + A[2];
```

Приклад 3*. Щоб поділити значення останнього елемента масиву* **A** *на 3 і присвоїти результат змінній* **y**, *необхідно написати:*

```
y = A[4] / 3;
```

Кожен масив займає певну область у пам'яті комп'ютера. Програміст повинен вказати тип елемента, кількість елементів, необхідних для кожного масиву, тоді компілятор зможе зарезервувати відповідний обсяг пам'яті. Наприклад, щоб компілятор зарезервував пам'ять для 5 елементів масиву цілих чисел **A**, можна використати таке оголошення: **`int A[5]`**;

Пам'ять для декількох масивів може бути зарезервована за допомогою одного оголошення. Наступне оголошення резервує пам'ять для 200 елементів масиву цілих чисел **b** й 15 елементів масиву цілих чисел у: **`int b[200], y[15];`**

Масиви можуть складатися з елементів не лише типу **`int`**, але й інших типів даних. Але всі елементи одного масиву повинні бути одного типу! Наприклад, для зберігання рядка символів можна використати масив типу **`char`**: **`char st[50];`**

Приклад 1. *Використаємо оператори циклу* **`for`** *для присвоєння початкових нульових значень елементам масиву* ***B****, що містить 8 цілих чисел, і для його друкування:*

```
#include<iostream.h>                  //Програма 5.1
#include<conio.h>
int main()
{
int B[8];
for (int i = 0; i < 8; i++)           //Присвоєння
  B[i] = 0;                    // початкових значень
cout<<"Nomer"<<'\t'<<"Znachennja"<<'\n';
for(int i = 0; i < 8; i++)            // Виведення
  cout<<i<<'\t'<<B[i]<<'\n';             // масиву
getch();
return 0;
}
```

Контроль за виходом за межі масиву С++ не здійснює, тому треба бути дуже уважним при використанні масивів!

Щоб переконатися в цьому, спробуйте при випробуванні програми 5.1 у другому циклі замість **`…;i < 8;…`** записати: **`…;i < 20;…`** Компілятор не виявить помилки й видасть на екран вісім нульових значень, а далі будуть виведені випадкові числа, які в даний час знаходяться у відповідних комірках пам'яті.

Перед використанням масиву його елементам необхідно присвоїти певні значення, в іншому випадку будемо мати масив з тими значеннями, які випадково опинилися у відповідних комірках пам'яті. Спробуйте, експериментуючи з програмою 5.1, вилучити рядки з присвоєнням початкових значень.

Зверніть увагу, що не можна безпосередньо вивести масив на екран. Якщо у цій програмі другий цикл замінити командою **`cout<<B`**, то замість елементів масиву на екрані побачимо щось на зразок 1245032. Це – адреса, за якою у пам'яті розміщений початок масиву. Саме тому для виведення всіх елементів масиву було використано цикл.

Оголошуючи масив, його елементам можна присвоїти початкові значення за допомогою вміщеного за оголошенням списку, взятого у фігурні дужки.

Приклад 2. *У даній програмі елементам масиву цілих чисел присвоюються вісім значень, після чого масив друкується.*

```
#include<iostream.h>                  //Програма 5.2
#include<conio.h>
int main()
{
int B[8] = {1, 3, 5, 7, 11, 13, 17, 19};
cout<<"Nomer"<<'\t'<<"Znachennja"<<'\n';
for (int i = 0; i <8; i++)
  cout<<i<<'\t'<<B[i]<<'\n';
getch();return 0;
}
```

Якщо початкових значень менше, ніж елементів у масиві, елементи, що залишилися, автоматично одержують нульові початкові значення. Всім елементам масиву **n** можна надати нульові початкові значення за допомогою оголошення:

`int n[10] = {0};`

яке явно надає нульове початкове значення першому елементу й неявно – нульові початкові значення дев'яти елементам, що залишилися, тому що початкових значень тут менше, ніж елементів масиву.

Якщо розмір масиву не зазначений в оголошенні, то кількість елементів масиву буде дорівнювати кількості елементів у списку

початкових значень. Наприклад, для створення масиву **B** з п'яти елементів запишемо: **int B[] = {1, 2, 3, 4, 5};**

Приклад 3. *У наступній програмі присвоюються початкові цілі значення 1, 2, 3,…,10 елементам масиву* **M** *з десяти елементів і на екран виводяться елементи масиву з парними номерами:*

```
#include<iostream.h>                    //Програма 5.3
#include<conio.h>
int main()
{
const int n = 10;
int M[n];
for (int k = 0; k < n; k++)
  M[k] = k + 1;
cout<<"Element"<<'\t'<<"Znachennja"<<'\n';
for (int k = 0; k < n; k += 2)
  cout<<k<<'\t'<<M[k]<<'\n';
getch();return 0;
}
```

Зверніть увагу, що у прикладі 5.3 для задання розміру масиву **M** в оголошенні **int M[n]** використовується іменована константа **n**.

Використання іменованих констант для задання розмірів масивів робить програму зручною для редагування.

У прикладі 5.3 перший цикл **for** заповнюватиме 200-елементний масив, якщо просто змінити значення **n** на початку програми з 10 на 200. Якби ми не використали іменовану константу **n**, потрібно було б змінити програму в трьох різних місцях, щоб застосувати її для обробки масиву з 200 елементів.

Передача масивів у функції

Програми можуть передавати масиви у функції так само, як і будь-які інші змінні. Передаючи масив у функцію, ви повинні вказати тип масиву. Оскільки в C++ не контролюється розмір масиву, то у функцію слід передати також параметр, що містить кількість елементів у масиві:

```
void  fun (int A[], int n);
```


Такий прийом дозволяє однією функцією обробляти масиви різних розмірів.

У наступній програмі масиви різних розмірів передаються у функцію **show_array**, яка використовує цикл **for** для виведення значень масивів:

```
#include<iostream.h>                    //Програма 5.4
#include<conio.h>
void show_array(int A[], int n)
{
int i;
for (i = 0; i < n; i++)
cout << A[i] << ' ';
cout << endl;
}
int main()
{
int little[5] ={1,2,3,4,5};
int big[3] = { 1000, 2000, 3000 };
show_array(little, 5);
show_array(big, 3);
getch();return 0;
}
```

Як бачите, у функцію передається ім’я масиву, а також вказується параметр, що повідомляє функції кількість елементів у масиві:

```
show_array(little, 5);
```

У наступній програмі для уведення з клавіатури значень у масив створена функція **get_values**:

```
#include<iostream.h>                    //Програма 5.5
#include<conio.h>
void get_values (int A[], int n)
{
int i;
  for (i = 0; i < n ; i++)
  {
    cout << "Vvedit’ " << i << ": ";
    cin >> A[i];
  }
}
```

```
int main()
{
int N[3];
get_values(N, 3);
cout << "Znachennya masivu:" << endl;
  for (int i = 0; i < 3; i++)
    cout << N[i] << endl;
getch();return 0;
}
```

Питання для самоконтролю:

1. *Що таке масив?*
2. *Для чого елементи масиву нумерують?*
3. *Як звернутися до елемента масиву?*
4. *Як нумеруються елементи масиву?*
5. *Яким повинен бути індекс масиву?*
6. *Чи можливо у якості індексу використовувати математичні вирази?*
7. *Які імена можуть мати масиви?*
8. *Для чого описують тип масиву?*
9. *Чи можуть у одному масиві використовуватися дані різних типів?*
10. *Що буде, якщо не присвоїти початкові значення всім елементам масиву?*
11. *Чи здійснюється у С++ контроль за виходом за межі масиву?*
12. *Чи можна таким чином вивести масив В на екран:* ***cout<<B****? Що при цьому ми побачимо на екрані?*
13. *Які способи присвоєння значень елементам масиву ви знаєте?*
14. *Що таке іменована константа? Для чого їх використовують при роботі з масивами?*
15. *Як передати масив у функцію?*

Вправа 5-1.

1) Випробуйте програму 5.1. Зробіть кожен елемент масиву рівним 5.
2) Випробуйте програму 5.2. Як змінити програму, щоб останні 4 елементи дорівнювали нулю? Спробуйте не вказувати розмір масиву в оголошенні.
3) Випробуйте програму 5.3. Задайте розмір масиву рівним 15. Виведіть на екран елементи масиву з непарними номерами.
4) Випробуйте програми 5.4.– 5.5.

Збережіть програми, створивши у власній папці нову папку *wpr5-1*.

5.2. Складання програм із масивами

Сума елементів масиву

Задача 1*. Знайти суму всіх елементів масиву цілих чисел* **A** *розмірністю* **n***. Дані уводяться в програму користувачем із клавіатури.*

```
#include<iostream.h>                    //Програма 5.4
#include<conio.h>
int main()
{
const int n = 10;
int A[n];
cout<<"Vvedi masiv iz "<<n<<" chisel:\n";
  for (int j = 0; j < n; j++)  //Уведення масиву
  {
    cout<<"A["<<j<<"]="; cin >> A[j];
  }
int S=0;   //Знаходження суми й друк результатів
  for (int j = 0; j < n; j++)
    S += A[j];
cout<<"Suma="<<S<<'\n';
getch();return 0;
}
```

Перша частина програми служить для уведення елементів масиву в пам'ять комп'ютера. У С++ не можна вказувати масив безпосередньо у команді уведення даних (наприклад, **cin>>A;**), бо це призводить до помилки компіляції. Тому для заповнення масиву з клавіатури використано цикл **for**, в якому значення кожного з елементів масиву уводиться окремо.

Для знаходження суми елементів масиву виділимо змінну з іменем **S**. Спочатку запишемо в неї значення 0 (додавання ще не здійснювалося). Потім, впродовж циклу, в змінній **S** будемо накопичувати результат підсумовування елементів масиву. Для цього щоразу до значення **S** додаватимемо черговий елемент масиву **A[j]** і результат вміщуватимемо знову в змінну **S**:

```
S += A[j]
```

Нагадуємо, що запис **S += A[j]** рівносильний **S = S + A[j]**.

Це продовжуватимемо доти, поки змінна циклу **j** не стане дорівнювати **n**. Як тільки це станеться – цикл завершиться і у змінній **S** залишиться результат підсумовування.

Пошук елементів із заданою властивістю

Задача 2. *Підрахувати і надрукувати кількість від'ємних елементів у масиві з 10 цілих чисел* ***x***.

```
#include<iostream.h>              //Програма 5.5
#include<conio.h>
int main()
{
const int n = 10;           // Розмірність масиву
int x[n];                    // Оголошення масиву
for(int i = 0; i < n; i++)
{
  cout<<i<<" element:\t";
  cin>>x[i];          //Уведення елементів масиву
}
// Підрахунок від'ємних
int col = 0;
for(int i = 0; i < n; i++)
  if(x[i] < 0)                          // Перевірка
    col++;                              // Лічильник
cout<<endl;
for(int i = 0; i < n; i++)
  cout<<x[i]<<"\t";       //Друк елементів масиву
cout<<"\nvidjemnih elementiv:\t"<<col<<endl;
getch();return 0;
}
```

У цій програмі, на відміну від попередньої, до змінної з ім'ям **col** буде додаватися одиниця, кожного разу, коли буде зустрічатися від'ємний елемент масиву. Відбір відбувається за допомогою умовного оператора **if(x[i] < 0)**.

Знаходження мінімального й максимального елементів

Змінним **Min** та **Max** на початку програми присвоїмо значення першого елементу масиву. Потім у циклі будемо перебирати по черзі всі наступні елементи масиву і виконувати перевірку:

- якщо поточний елемент масиву більший, ніж **Max**, то змінній **Max** присвоюється значення поточного елемента;

- якщо поточний елемент масиву менший, ніж **Min**, то змінній **Min** присвоюється значення поточного елемента.

```
#include<iostream.h>              //Програма 5.6
#include<conio.h>
int main()
{
const int n = 10;           //Розмірність масиву
float a[n];                  //Оголошення масиву
for(int i = 0; i < n; i++)
{
  cout<<"Vvedit' "<<i<<" element:\t";
  cin>>a[i];          //Уведення елементів масиву
}
float Min, Max;
Min=Max=a[0];  //Ініціалізація 1-м ел-том масиву
// Порівняння з поточним елементом
for(int i = 1; i < n; i++)
   if(a[i] > Max)
     Max = a[i];          //Поточний Max елемент
   else
     if(a[i] < Min)
          Min = a[i];   // Поточний Min елемент
cout<<"\nMin=\t"<<Min<<"\nMax=\t"<<Max<<endl;
getch();
return 0;
}
```

Питання для самоконтролю:

1. *Чи можна увести елементи масиву таким чином:* ***cin>>A****?*
2. *Який оператор використовують для послідовної обробки елементів масиву?*
3. *Опишіть алгоритм знаходження суми елементів масиву.*
4. *Опишіть алгоритм підрахунку кількості від'ємних елементів у масиві.*
5. *Опишіть алгоритм знаходження найбільшого елемента в масиві.*

Вправа 5-2.

1) Випробуйте програму 5.4 для n=10 та n=5. Знайдіть добуток всіх елементів таблиці A, яка містить тільки додатні цілі числа.
2) Випробуйте програму 5.5. Підрахуйте і надрукуйте кількість додатних елементів масиву цілих чисел x.

3) Випробуйте програму 5.6. Уважно дослідіть роботу програми. Зменште значення кожного елементу масиву на величину мінімального елемента.

Збережіть програми, створивши у власній папці нову папку *wpr5-2*.

5.3. Практична робота № 9 «Розробка програм із масивами»

1) Випадковим чином заповнити масив з 30 цілих чисел. Визначити, скільки елементів відмінні від останнього.
2) Увести число. Масив з 20 чисел заповнити випадковими числами в межах від 0 до 50. Перевірити, чи зустрічається уведене число в масиві.
3) Випадковим чином задати масив з 25 цілих чисел. Надрукувати спочатку всі від'ємні числа, потім всі інші.

Збережіть програми, створивши у власній папці нову папку *pr9*.

5.4. Алгоритми сортування

Сортування – один із найпоширеніших алгоритмів, що використовується при розв'язуванні завдань на комп'ютері. Під сортуванням мають на увазі упорядкування даних за певною ознакою: списку учнів класу за алфавітом, списку учасників змагань за зменшенням кількості набраних балів тощо. У подібних випадках зручно зберігати дані в масивах. Числові дані упорядковують за зростанням або за спаданням.

Функція sizeof

Але перед розглядом алгоритмів сортування розглянемо ще одну із функцій мови C++ – **sizeof**.

Функція визначення розміру **sizeof** використовується для обчислення розміру значення виразу чи типу в байтах і має дві форми:

sizeof (вираз)
sizeof (тип)

Наведемо приклад:

```
#include<iostream.h>
#include<conio.h>
int main()
```



```
{
double x=1;
cout<< "sizeof(float): "<<sizeof(float)<<endl;
cout<< "sizeof(int): "<<sizeof(int)<<endl;
cout<< "sizeof(x): "<<sizeof(x)<<endl;
cout<< "sizeof (x+1): "<<sizeof (x+1)<<endl;
getch();return 0;
}
```

Результат роботи програми:

```
sizeof(float): 4
sizeof(int): 4
sizeof(x): 8
sizeof (x+1): 8
```

Сортування простим пошуком

Розглянемо один з найпростіших алгоритмів сортування числового масиву за спаданням – метод простого пошуку. Переглянемо спочатку весь масив з 1-го до останнього елемента й знайдемо найбільший із них. Поміняємо місцями знайдений елемент і 1-й елемент масиву. Потім переглянемо всі елементи масиву, починаючи з 2-го, і знову знайдемо максимальний. Поміняємо його місцями з 2-м елементом масиву. Повторюватимемо ці дії до досягнення кінця масиву. Врешті залишиться переглянути тільки 2 останніх елементи, вибрати більший і поставити його на передостаннє місце в масиві. Тепер таблиця впорядкована в порядку спадання.

Отже описаний метод сортування полягає у повторенні двох етапів:

1) пошук серед невпорядкованих елементів масиву елемента з найбільшим значенням;

2) упорядкування таблиці шляхом перестановки знайденого елемента із черговим (1-м, 2-м, 3-м...) елементом таблиці.

Розглянемо ці завдання детальніше. Нехай заданий масив **A**, частину елементів якого від **A[m]** до **A[n]** треба упорядкувати за спаданням.

Побудуємо функцію для пошуку найбільшого серед вказаних елементів масиву **A**. Для цього будуть використовуватися змінні:

i – номер 1-го з невпорядкованих елементів масиву;

max – значення найбільшого із уже переглянутих елементів;

L – номер найбільшого серед переглянутих елементів.

Алгоритм починається із серії команд, у якій змінна **max** отримує значення елемента таблиці з індексом **m**. Цей елемент уже переглянутий і ми беремо **m** як значення для **L**. Переглянутий один елемент **A[m]**, отже він – максимальний серед переглянутих, тому для **i** – номера першого з непереглянутих елементів, встановлюємо значення **m+1**.

Цикл працює в такий спосіб. Якщо **max** виявився меншим, ніж елемент **A[i]**, потрібно змінити **max**, і ми це робимо, присвоюючи йому значення **A[i]**; відповідно з цим і значення **L** міняється, тепер воно дорівнює **i**. Оскільки елемент з номером **i** уже опрацьований, серія команд завершується збільшенням **i** на 1 (**i++**). Робота алгоритму завершується, коли всі елементи переглянуті і номер максимального елемента знайдений. Він і передається як результат в основну програму.

```
#include<iostream.h>                    //Програма 5.7
#include<conio.h>
int MaxElement(int A[],int m,int n)
{
int max=A[m];
int L=m;
int i=m+1;
    while(i<n)
    {
      if(A[i]>=max)
      {
        max=A[i];
        L=i;
      }
      i++;
    }
return L;
}
int main()
{
int C[]={1,6,21,4,3,5,8,9,7,90,123,1,345};
int mc=0;
int nc=sizeof(C)/sizeof(int);
int Temp,Lc;
```



```
  while(mc<nc)
  {
   Lc=MaxElement(C,mc,nc);
   Temp=C[mc];                    //Обмін місцями 2-х
   C[mc]=C[Lc];                    //елементів масиву
   C[Lc]=Temp;              //за допомогою допоміжної
   mc++;                                 //змінної Temp
  }
   for(int i=0;i<nc;i++)
                cout<<" "<<C[i];
getch();return 0;
}
```

Побудуємо тепер алгоритм упорядкування (функцію **main**), використовуючи **MaxElement** як допоміжну функцію.

Нехай заданий масив **C**, елементи якого нумеруються від 0 до nc. Присвоїмо необхідним для роботи змінним початкові значення: **mc=0; nc=sizeof(C)/sizeof(int);**. Завдяки використанню функції **sizeof** буде правильно опрацьований масив будь-якого розміру. Можна написати також:

nc=sizeof(C)/sizeof(C[0]);

– тоді робота цього оператора не залежатиме навіть від типу елементів масиву.

Нам необхідно переставити елементи масиву так, щоб вони йшли в порядку спадання. Застосувавши допоміжну функцію **MaxElement(C,mc,nc)** до масиву **C** ми визначимо номер **Lc** елемента цієї таблиці, що має найбільше значення. Після цього ми поміняємо значення елементів **C[mc]** й **C[Lc]**. Для обміну двох елементів масиву використаємо відомий вам лінійний алгоритм (п. 1.2, прикл. 3).

Тоді на **mc**–му місці масиву **C** виявиться найбільший з елементів. На наступному кроці застосовуємо алгоритм **MaxElement** до частини масиву **C** з номерами від **mc+1** до **nc** і знову визначаємо номер **Lc** максимального елемента. Поміняємо місцями елементи **C[mc+1]** й **C[Lc]**, тоді на **mc+1** місці виявиться найбільший з елементів, що залишилися. Далі будемо застосовувати алгоритм **MaxElement** до частини масиву **C**, що починається з номера **m+2,** потім з **m+3** і т.д. й міняти місцями – відповідно **C[mc+2]** й

C[Lc], **C[mc+3]** й **C[Lc]** і т.д. У результаті масив **C** виявиться впорядкованим за спаданням.

Функцію **MaxElement** можна спростити:

```
…
{
int L=m;
int i=m+1;
    while(i<n)
    {
      if(A[i]>= A[L])
        L=i;
    i++;
    }
return L;
}
…
```

Внесіть зміни та проаналізуйте роботу програми самостійно. Розроблена програма може бути легко перероблена для сортування масиву за зростанням. Спробуйте самостійно з'ясувати, які зміни слід внести для цього в текст програми.

Метод «бульбашки»

Метод бульбашки в більшості випадків є ефективнішим, ніж описаний вище метод простого пошуку. Для сортування, наприклад, за зростанням, з використанням цього алгоритму, весь масив переглядається кілька разів підряд. При кожному такому перегляді порівнюються послідовно тільки сусідні елементи масиву: спочатку перший із другим, потім другий із третім, і наприкінці – передостанній з останнім. Якщо при порівнянні виявиться, що попередній елемент більший від наступного, вони міняються місцями (для цього, як і у попередньому сортуванні використовуємо

	I	II	III	IV	V	VI
a[7]	6	14	14	14	14	14
a[6]	10	6	13	13	13	13
a[5]	14	10	6	10	10	10
a[4]	5	13	10	6	9	9
a[3]	0	5	9	9	6	8
a[2]	8	0	5	8	8	6
a[1]	13	8	0	5	5	5
a[0]	9	9	8	0	0	0

проміжну змінну **Temp**). Так, у результаті одного послідовного перегляду елементи, значення яких більші, ніж у сусідів, переміщуються на 1 або більше елементів ближче до кінця масиву.

Якщо ряд чисел зобразити не горизонтально, а вертикально, щоб елемент **a[7]** був зверху, а **a[0]** – знизу, стає зрозумілою назва – метод «бульбашки»: більші елементи, як бульбашки у воді, «спливають» на відповідні позиції.

```
#include<iostream.h>                    //Програма 5.8
#include<conio.h>
int main()
{
const int n = 8;
int i;
int a[n] = {9, 13, 8, 0, 5, 14, 10, 6};
cout <<"\nVivedennja masivu: \n";
for ( i = 0; i < n; i++)
                    cout << a[i] << '\t';
int Temp, j = 1;
bool prapor = false;
do
{
      prapor = false;
      for(i = 0; i <n-j; i++)
            if (a[i] > a[i+1])
            {
                  Temp = a[i];
                  a[i] = a[i+1];
                  a[i+1] = Temp;
                  prapor = true;
            }
      j++;
}
while (prapor);
cout << "\nPislja sortuvannja:\n";
for (i = 0; i < n; i++)
                    cout << a[i] << '\t';
getch();
return 0;
}
```

Якщо провести такий послідовний перегляд масиву кілька разів, то «важкі» елементи остаточно «спливуть» і масив виявиться відсортованим.

Залишається невирішеним ще одне питання. Скільки разів потрібно виконувати такий перегляд? Адже може виявитися, що одного перегляду буде досить. Відповідь – стільки, скільки потрібно для повного сортування, тобто поки при повному перегляді від початку до кінця масиву жодна пара елементів не поміняється місцями. Для припинення сортування зручно скористатись логічною змінною, котрій спочатку присвоюється **`false`**, а потім присвоюється значення **`true`** при кожному обміні. Якщо при черговому перегляді не відбулося жодного обміну, значенням цієї змінної залишиться **`false`** і перегляди припиняються (цикл **`do...while (prapor);`**).

Сортування виконується за допомогою вкладеного циклу **`for`**.

Питання для самоконтролю:

1. *Що таке сортування?*
2. *Для чого використовують сортування?*
3. *Як визначити розмір виразу чи типу в байтах?*
4. *Як визначити кількість елементів масиву?*
5. *Як здійснити обмін місцями 2-х елементів масиву?*
6. *Чому метод «бульбашки» більш ефективний, ніж метод простого пошуку?*
7. *Завдяки чому виникла назва «метод бульбашки»?*

Вправа 5-4.

1) Випробуйте програму 5.7. Уважно дослідіть роботу програми
2) Додайте елементи у масив. Перевірте роботу програми.
3) Уведіть від'ємні числа. Запустіть програму.
4) Вдоскональте програму 5.8, доповнивши її двома функціями: для виведення масиву і для обміну двох змінних.

Збережіть програми, створивши у власній папці нову папку *wpr5-4*.

5.5. Практична робота № 10 «Впорядкування масивів»

1) З клавіатури уводиться 15 цілих чисел. Вивести їх на екран упорядкованими за спаданням.

2) Увести два впорядкованих за зростанням масиви m1 та m2. Об'єднати їх в один масив так, щоб він виявився впорядкованим за зростанням. Наприклад:

```
m1->1 3 5 7 9;   m2->2 4 6 8 10
Rez->1 2 3 4 5 6 7 8 9 10
```

Збережіть програми, створивши у власній папці нову папку *pr10*.

5.6. Вказівники, динамічні змінні й масиви

Статичні та динамічні дані

При розробці великих програм постає проблема раціонального використання оперативної пам'яті комп'ютера. Її обсяг обмежений, що ускладнює обробку великих об'ємів даних.

У С++ змінні можуть бути розміщені або статично – під час компіляції, або динамічно – під час виконання програми, шляхом виклику відповідних функцій зі стандартної бібліотеки.

Основна різниця цих методів – у їхній ефективності й гнучкості. Статичне розміщення більш ефективне, тому що виділення пам'яті відбувається до виконання програми, однак воно менш гнучке, тому що, складаючи програму, ми повинні заздалегідь знати тип і розмір розташовуваної змінної. Приміром, у подальших темах ми побачимо, що зовсім не просто розмістити вміст деякого текстового файлу в статичному масиві рядків: нам потрібно знати його розмір. Завдання, у яких потрібно зберігати й обробляти заздалегідь невідоме число елементів, зазвичай вимагають динамічного виділення пам'яті.

Після запуску програми в пам'яті розміщується безпосередньо код програми та виділяється місце для статичних даних (змінних, масивів). У пам'яті, що залишилась вільною, можна, під час виконання програми, виділяти місце відповідно до потреб. Виділені таким чином ділянки динамічної пам'яті називаються **динамічними змінними**. Після того як потреба у динамічній змінній зникає, місце, яке вона займає, можна звільнити для інших динамічних об'єктів.

Доступ до динамічної пам'яті можна одержати тільки через вказівники.

Робота з динамічними даними

Виділення динамічної пам'яті здійснюється **операцією new**. За допомогою **new** виділяється пам'ять (скільки потрібно), і адреса, з якої вона починається, заноситься у вказівник на потрібний тип. Наприклад:

```
int* n = new int;                    //зразок 1
int* m = new int (10);               //зразок 2
```

У зразку 1 операція **new** виконує виділення достатньої для розміщення величини типу **int** ділянки динамічної пам'яті й записує адресу початку цієї ділянки в змінну **n**. Пам'ять під саму змінну **n** (розміру, достатнього для розміщення вказівника) виділяється на етапі компіляції.

У зразку 2, крім описаних вище дій, проводиться ініціалізація виділеної динамічної пам'яті значенням 10.

Для звільнення пам'яті, виділеної за допомогою операції **new**, використовують **операцію delete**. Наведені вище динамічні змінні **n** й **m** знищуються в такий спосіб:

```
delete n;
delete m;
```

Виділяють дві основні відмінності між статичним і динамічним виділенням пам'яті.

Статичні об'єкти позначаються іменованими змінними, і дії над цими об'єктами проводяться безпосередньо, з використанням їхніх імен. Динамічні об'єкти не мають власних імен, і дії над ними проводяться опосередковано, за допомогою вказівників.

*Масиви, які розміщуються в динамічній пам'яті, називаються **динамічними**.*

Створюються динамічні масиви також за допомогою операції **new**, при цьому необхідно вказати тип і розмірність:

```
int n = 100;    //змінна n для розмірності масиву
float *p = new float [n];
```

У цьому рядку створюється змінна-вказівник на **float**, у динамічній пам'яті виділяється безперервна область, достатня для розміщення 100 елементів дійсного типу, і адреса її початку записується у вказівник **p**. Динамічні масиви при створенні ініціювати не можна, і вони не заповнюються нулями.

Перевага динамічних масивів полягає в тому, що їх розмірність може бути змінною, тобто обсяг пам'яті, виділений під масив, визначається на етапі виконання програми.

Доступ до елементів динамічного масиву здійснюється так само, як до статичного. Наприклад, до 5-го елемента наведеного вище масиву можна звернутися як **p[5]** або ***(p+5)**.

Пам'ять, зарезервована під динамічний масив за допомогою **new[]**, повинна звільнятися оперцією **delete[]**:

```
delete [] p;
```

***Приклад**. З клавіатури вводиться ціле число **n**, а потім ще **n** цілих чисел. Розмістити уведені **n** чисел у динамічному масиві, після чого вивести їх на екран, відокремивши пропусками.*

```
#include<iostream.h>              //Програма 5.9.1
#include<conio.h>
int main()
{
int n,*a;  //Опис змінної розмірності, та масиву
cout<<"Rozmirnist' masivu: ";
cin>>n;             // Кількість елементів масиву
a=new int[n];            // Виділення пам'яті для
                          // масиву з n елементів
// Уведення елементів масиву
for(int i = 0; i < n; i++)
  {
    cout<<i<<" Element=";
    cin>>a[i];
  }
// Виведення масиву на екран
cout<<"\nMasiv:\n";
for(int i = 0; i < n; i++)
    cout<<a[i]<<"   ";
cout<<endl;
delete[]a;                    //Вивільнення пам'яті
getch();
return 0;
}
```

Операції з вказівниками

Якщо до вказівника додати 1 (інкремент), то зміниться адреса, на яку він вказує. Але це не означає, що він після цього буде

вказувати на наступний байт пам'яті. Адреса збільшиться на розмір типу вказівника. Тобто:

```
char ch;
char* chPtr;
chPtr++;    //Збільшився на розмір char (1 байт)
double db;
double *dbPtr;
dbPtr++;  //Збільшився на розмір double (8 байт)
```

У такий же спосіб до вказівника можна додати будь-яку цілу константу.

Різниця двох вказівників – це різниця їхніх значень, ділена на розмір типу в байтах (у застосуванні до масивів різниця вказівників, наприклад, на третій і шостий елементи дорівнює 3).

Додавання двох вказівників не допускається.

Можна використовувати вказівники на константи – це вказівники, оголошені із ключовим словом **const**:

```
const double *cptr;
```

де **cptr** – вказівник на змінну типу **const double**. Сам вказівник – не константа і його значення можна змінювати. Адреса константного об'єкта присвоюється тільки вказівнику на константу.

Існують і константні вказівники. Вони можуть адресувати як константу, так і змінну. Спроба присвоїти значення константному вказівнику викличе помилку компіляції.

Питання для самоконтролю:

1. *Які методи виділення пам'яті для змінних існують у С++?*
2. *У чому різниця між цими методами?*
3. *За допомогою якої операції здійснюється виділення динамічної пам'яті?*
4. *Як знищити динамічні змінні?*
5. *Яка різниця між статичним і динамічним виділенням пам'яті?*
6. *Як створити динамічний масив?*
7. *У чому перевага динамічних масивів?*
8. *Як здійснюється доступ до елементів динамічного масиву?*
9. *Як вивільняється пам'ять, зарезервована під динамічний масив?*
10. *Які операції використовуються при роботі з вказівниками?*

Вправа 5-6.

1) Випробуйте програму, яка реалізує такі функції для роботи з масивами:

   а) функція уведення елементів масиву;

   б) функція виведення масиву;

   в) функція сортування масиву.

   Уважно дослідіть роботу програми.

```
#include<iostream.h>                  //Програма 5.9
#include<conio.h>
//Прототипи
void _Input(int*,int);          //функції уведення
void _Sort(int*,int);          //функції сортування
void _Print(int*,int);          //функції виведення
int main()
{
int n,*a;
cout<<"Vvedi rozmir masivu:\t";
cin>>n;                 //Кількість елементів масиву
a=new int[n];               //Виділення пам'яті для
                             //масиву з n елементів
_Input(a,n);           //Уведення елементів масиву
cout<<"\nMasiv:\n";
_Print(a,n);           //Виведення масиву на екран
_Sort(a,n);                    //Сортування масиву
cout<<"Vidsortovanij masiv:\n";
_Print(a,n);    //Виведення відсортованого масиву
delete[]a;                    //Вивільнення пам'яті
getch();return 0;
}
//ФУНКЦІЇ
void _Input(int *a, int n)              //Уведення
{
  for(int i = 0; i < n; i++)
  {
    cout<<"Element "<< i + 1 <<"\t";
    cin>>a[i];         //Уведення елементів масиву
  }
}
void _Print(int *a, int n)             //Виведення
{
  for(int i = 0; i < n; i++)
    cout<<a[i]<<"   ";
  cout<<endl;         //Виведення елементів масиву
}
```

```
void _Sort(int *a, int n)              //Сортування
{
int temp;  //Тимчасова змінна для обміну значень
bool flag = true;          //Прапорець закінчення
  for(int j = 1; ; j++)                //сортування
  {
    for(int i = 0; i < n - j; i++)
      if(a[i] > a[i+1])
      {
       temp = a[i];
       a[i] = a[i+1];
       a[i+1] = temp;
       flag = false;
      }
    if(flag == true)
       break;
    flag = true;
  }
}
```

Збережіть програму, створивши у власній папці нову папку *wpr5-6*.

5.7. Багатовимірні масиви

Масиви в С++ можуть бути також двовимірними, тривимірними й більше. Простим прикладом двовимірного масиву є сторінка класного журналу, що містить інформацію в рядках і стовпцях. Такий масив має два індекси: перший указує номер рядка, а другий – номер стовпця.

стовпці (stovp)

рядки (ryad)

i \ j	0	1	2	3
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

На малюнку зображена схема нумерації елементів двовимірного масиву **a**, який містить три рядки й чотири стовпці.

Взагалі, масиви з **ryad** рядками й **stovp** стовпцями називають масивами розміром **ryad** на **stovp** (наприклад, масив 3 на 4).

Кожен елемент у масиві **a** визначається ім'ям елемента у формі **a[i][j]; a** – це ім'я масиву, а **i** та **j** – індекси, які однозначно визначають кожен елемент в **a**. Нумерація рядків і стовпців елементів масиву починається з нуля. Багатовимірні масиви можуть одержувати початкові значення у своїх оголошеннях так само, як масиви з єдиним індексом. Наприклад, двовимірний масив **c[3][2]** можна оголосити й дати йому початкові значення в такий спосіб:

```
int c[3][2] = {{4, 2}, {6, 7},{5, 8}};
```

Значення групуються в рядки, вміщені у фігурні дужки. Таким чином, елементи **c[0][0]** і **c[0][1]** одержують початкові значення 4 й 2, а елементи **c[1][0]** і **c[1][1]** одержують початкові значення 6 й 7 і т.д. Якщо початкових значень у даному рядку не вистачає для їхнього присвоєння всім елементам рядка, то елементам, що залишаються, присвоюються нульові початкові значення. Таким чином, оголошення

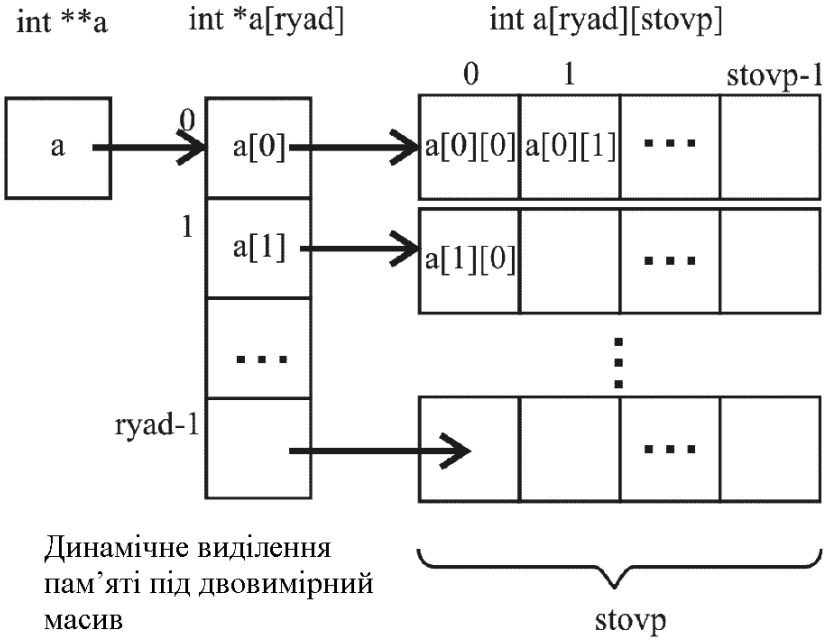
```
int c[2][2] = {{10}, {9, 14}};
```

буде означати що **c[0][0]** одержує початкове значення 10, **c[0][1]** одержує початкове значення 0, **c[1][0]** одержує початкове значення 9 і **c[1][1]** одержує початкове значення 14.

Динамічні багатовимірні масиви

Виділення пам'яті під двовимірний масив цілих чисел, коли обидві його розмірності задаються під час виконання програми, можна забезпечити в такий спосіб:

```
int ryad, stovp;
cout<<"Uvedit' kilkist' ryadkiv:";cin>>ryad;
cout<<"Uvedit' kilkist' stovpciv:";cin>>stovp;
int **a= new int *[ryad];                    //1
for (int i=0; i<ryad; i++)                   //2
     a[i] = new int [stovp];                 //3
```



В операторі 1 оголошується змінна типу «вказівник на вказівник на **int**» і виділяється пам'ять (див. мал.) під масив вказівників на рядки масиву (кількість рядків – **ryad**). В операторі 2 організується цикл для виділення пам'яті під кожен рядок масиву. В операторі 3 кожному елементу масиву вказівників на рядки присвоюється адреса початку ділянки пам'яті, виділеної під рядок двовимірного масиву. Кожен рядок складається з **stovp** елементів типу int.

Якщо при описі змінної використаються вказівник й [] (масив), то змінна інтерпретується як масив вказівників, а не вказівник на масив:
*int *p[10] – масив з 10 вказівників на int.*

Звільнення пам'яті з-під масиву з будь-якою кількістю вимірів виконується за допомогою операції **delete []**. Вказівник на константу видалити не можна.

Приклад 1. *Написати програму, в якій для двовимірного масиву цілих чисел визначається номер крайнього стовпця, розташованого ліворуч, що містить тільки додатні елементи. Якщо такого стовпця немає, виводиться відповідне повідомлення.*

Переглядатимемо масив стовпець за стовпцем. При цьому швидше міняється перший індекс (номер рядка). Зробити висновок про те, що який-небудь стовпець містить тільки додатні елементи, можна тільки після перегляду стовпця цілком. Якщо ж у процесі перегляду зустрівся від'ємний елемент, можна відразу переходити до наступного стовпця.

Цей алгоритм реалізується за допомогою змінної-прапорця **posit**, що перед початком перегляду кожного стовпця встановлюється у значення **true**. При знаходженні у стовпці від'ємного елемента змінна-прапорець отримує значення **false**. Якщо стовпець містить тільки додатні елементи, прапорець залишиться рівним **true**, що буде ознакою наявності в масиві шуканого стовпця. Після виявлення шуканого стовпця, переглядати масив далі не має сенсу, тому виконується вихід із циклу за допомогою **break** й виведення результату.

```
#include<iostream.h>                    //Програма 5.10
#include<iomanip.h>
#include<conio.h>
int main()
{
int ryad, stovp;
// уведення розмірності масиву
cout<<"Vvedi kilkist' ryadkiv:";cin >> ryad;
cout<<"Vvedi kilkist' stovpciv:";cin>>stovp;
// виділення пам'яті під масив
int i, j;
int **a = new int *[ryad];
for( i = 0; i < ryad; i++)
   a[i] = new int [stovp];
// уведення масиву
cout << "Uvedit elementu masivu:" << endl;
for (i = 0; i < ryad; i++)
  for (j = 0; j < stovp; j++)
cin >> a[i][j];
```

```
// виведення масиву
for (i = 0; i < ryad; i++)
{
  for (j = 0; j < stovp; j++)
     cout << setw(4) << a[i][j] <<"  ";
  cout << endl;
}
//аналіз масиву
int n = -1;
bool dodat;
for (j = 0; j < stovp; j++)
{
      //перегляд по стовпцях
      dodat = true;
      //аналіз елементів стовпця
      for (i = 0; i < ryad; i++)
        if (a[i][j] < 0)
        {
          dodat = false;break;
        }
      if (dodat)
      {
        n = j;break;
      }
}
if (n ==-1  ) cout << " Stovpciv nema" <<endl;
else cout<<"Nomer stovpcya: "<<n<<endl;
getch();return 0;
}
```

Можливий випадок, коли жоден стовпець не задовольнить умову. Для виявлення цього змінній **n**, у якій буде зберігатися номер шуканого стовпця, присвоюється початкове значення, наприклад –1, яке, зазвичай, не може бути номером стовпця. Перед виведенням результату аналізується значення змінної **n**: якщо воно після перегляду масиву збереглося незмінним (–1), то стовпців, що задовольняють задану умову, у масиві немає.

Вправа 5-7.

1) Цілочисельний масив з 8 рядків та 15 стовпців заповнити випадковими одноцифровими числами. Скільки разів у масиві зустрічається число 5?
2) Доповнити програму-розв'язок попередньої задачі виведенням таблиці кількостей входжень у масив кожного з чисел, наприклад:
   ```
   0 – 5 raz
   1 – 8 raz
   2 – 2 raz...
   ```
3) Випробуйте програму 5.10. Перероблять її так, щоб визначити номер крайнього стовпця, розташованого ліворуч, що містить тільки від'ємні елементи. Якщо такого стовпця немає, виведіть повідомлення.

Збережіть програму, створивши у власній папці нову папку *wpr5-7*.

Питання для самоконтролю:

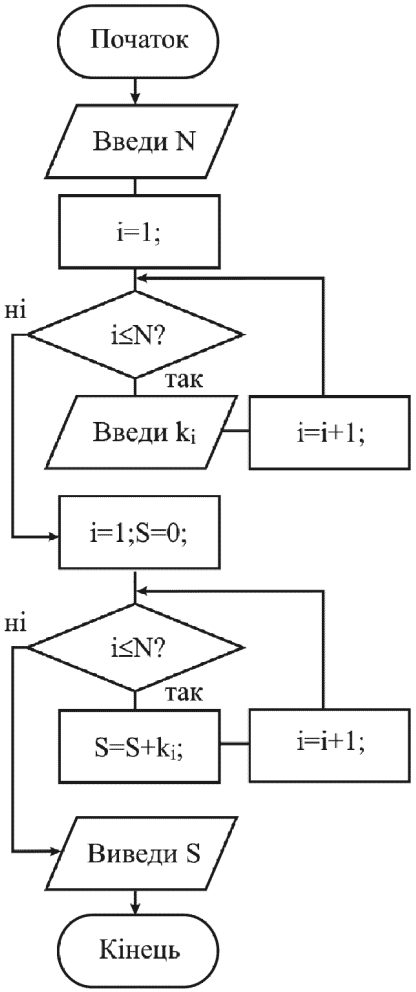
1. *Як нумеруються елементи масиву?*
2. *Чи здійснюється в С++ контроль за виходом за межі масиву?*
3. *Чи можна не вказувати розмір масиву, наприклад:* ***int A[]?***
4. *Скільки елементів містить масив* ***B[9]?***
5. *Чи можуть елементи масиву бути типу* ***char****?*
6. *Для чого використовують таку послідовність операторів:*
   ```
   const int n = 100;
   int c[n];
   ```
7. *Чи можуть в одному масиві зберігатися дані різних типів?*
8. *Чому дорівнює елемент з індексом 2 масиву A:*

A[i]	1	2	3	5	7

9. *Що таке вказівник? Чи можна присвоїти вказівнику число?*
10. *Чи можна присвоїти вказівнику адресу змінної? Якщо так, то як?*
11. *Чи можна ініціалізувати динамічний масив? Якщо так, то як?*
12. *Як звільнити динамічну пам'ять, яку займає масив?*
13. *Як краще оголосити масив, якщо:*
    - *кількість елементів масиву відома до виконання програми;*
    - *кількість елементів не відома до виконання програми, але її можна задати під час виконання програми до уведення елементів масиву;*
    - *кількість елементів масиву буде визначена в ході уведення.*
14. *Наведіть приклади двовимірних масивів.*
15. *Як оголосити двовимірний масив?*
16. *Як надати початкові значення елементам масиву?*
17. *Як відбувається динамічне виділення пам'яті під двовимірний масив?*

5.8. Тематичне оцінювання з теми «Масиви»

1) Барон Мюнхгаузен, вийшовши на екологічно чисте полювання, зарядив свою рушницю кісточками вишень. Після того, як він вдало влучив між роги N оленям, в яких потрапило відповідно k1, k2, …kN кісточок, у них на головах виросли чудові молоді вишеньки. Скільки нових саджанців зміг подарувати барон Мюнхгаузен садівникам-дослідникам? (Користуйтеся наведеною блок-схемою).



2) Заданий одновимірний масив цілих чисел **A[i]**, де **i=1,2,…,n**. Вивести значення елементів масиву:
   а) у зворотному порядку;
   б) з парними індексами;
   в) з непарними індексами;
   г) які є парними числами;
   д) які є непарними числами.

Збережіть програми, створивши у власній папці нову папку *ta3*.

6. Рядкові величини

6.1. Символьні рядки

Комп'ютер використовують для обробки інформації різних видів, зокрема, і текстової. Текст можна розглядати як сукупність символьних рядків. Розглянемо засоби, які мова С++ надає для роботи з ними.

Оголошення символьних рядків у програмі

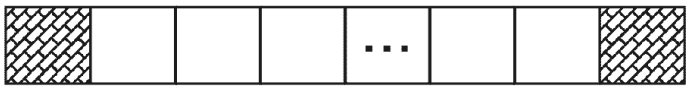
Символьні рядки у С++ являють собою звичайні масиви. Головна відмінність між символьними рядками й іншими типами масивів полягає в тому, що символьний рядок можна виводити, наприклад, командою **cout << abc;**, а не організовувати цикл для поелементного виведення даних.

Кінець символьного рядка позначається символом NULL, що у програмі зображується як спеціальний символ '**\0**'. Можна сказати, що саме NULL перетворює частину символьного масиву в символьний рядок.

Для оголошення символьного рядка усередині програми просто оголосіть масив типу **char** з кількістю елементів, достатньою для зберігання необхідних символів. Наприклад, оголосимо змінну з ім'ям **A**, здатну зберігати 32 символи (не забувайте, що символ NULL є одним із цих 32 символів):

```
char A[32];
```

Як видно з мал. 1, це оголошення створює масив з елементами, пронумерованими від **A[0]** до **A[31]**.

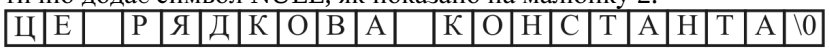


С++ трактує символьний рядок як масив типу char

Розглянуті раніше програми використовують рядкові константи, уміщені у подвійні лапки:

```
"Це рядкова константа"
```

При створенні рядкової константи компілятор С++ автоматично додає символ NULL, як показано на малюнку 2.



Компілятор С++ автоматично додає символ NULL

Коли при виконанні програми за допомогою вихідного потоку **cout** виводиться рядкова константа, **cout** використовує символ '**\0**' як ознаку кінця рядка.

Більшість функцій C++, призначених для роботи з рядками, використовують символ NULL для визначення останнього символу рядка.

У наступній програмі уводиться рядок з використанням циклу **for**, а потім виводиться на екран за допомогою **cout**:

```
#include<iostream.h>                    //Програма 6.1
#include<conio.h>
int main()
{
char abc [27];              // 26 символів плюс NULL
char let; int i;
//Заповнення рядкового масиву латинськими
літерами
for (let = 'A', i = 0; let <= 'Z'; let++, i++)
      abc [i] = let;
abc [i] = '\0';                  //Ознака кінця рядка
cout << abc;          //Виведення рядкової константи
getch();
return 0;
}
```

У програмах на C++ зустрічаються окремі символи, взяті в одинарні лапки (наприклад, **'A'**), і символи у подвійних лапках (**"A"**).

Між цими записами є принципова різниця. Символ в одинарних лапках являє собою **символьну** константу.

Як “A” відрізняється від ‘A’

Компілятор C++ виділяє тільки один байт пам'яті для її зберігання. Однак символ у подвійних лапках являє собою **рядкову** константу – зазначений символ і символ NULL (доданий компілятором). Таким чином, компілятор виділить два байти для символьного рядка. Мал.3 ілюструє, як компілятор C++ зберігає рядкову константу **"A"** і символьну константу **'A'**.

Ініціалізація символьного рядка

Як ви вже знаєте, C++ дозволяє ініціалізувати масиви при оголошенні. Символьні рядки C++ не є винятком. Для ініціалізації символьного рядка при оголошенні вкажіть необхідну послідовність символів усередині подвійних лапок, як показано нижче:

```
char tit [32] = "Вивчаємо мову C++";
```

Якщо кількість символів, що присвоюється рядку, менша від розміру масиву, більшість компіляторів C++ присвоюють символи NULL елементам рядкового масиву, що залишаються. Як і у випадку з масивами інших типів, якщо ви не вказуєте розмір масиву, що ініціюєте при оголошенні, компілятор C++ виділить достатньо пам'яті для розміщення зазначених символів рядка і службового символу NULL:

```
char tit[] = "Вивчаємо мову C++";
```

При компіляції наведеного фрагменту для рядка **tit** буде виділено 18 байтів (15 зображених символів + 2 пропуски + символ '\0').

У наступній програмі символьні рядки ініціалізуються при оголошенні:

```
#include<iostream.h>                    //Програма 6.2
#include<conio.h>
int main()
{
char tit[32] = "Hello, C++";
char les[] = "Simvolni ryadki!";
cout<<"Book: "<<tit<<endl;
cout<<"Lesson: "<<les<<endl;
getch();
return 0;
}
```

Зверніть увагу, що для масиву **tit** виділено 32 байти, хоча рядок, занесений в нього при ініціалізації, – коротший. Тому пізніше, в програмі, у цей масив можна буде вмістити довший рядок, обов'язково помістивши в кінці символ '**\0**'.

Передавання рядків у функції

Передавання символьного рядка у функцію подібне до передавання будь-якого масиву як параметру. Усередині функції потрібно просто вказати тип масиву (**char**) і квадратні дужки масиву. Не треба вказувати розмір рядка. Наприклад, наведена програма використовує функцію **show_ryad** для виведення символьного рядка:

```
#include<iostream.h>                    //Програма 6.3
#include<conio.h>
void show_ryad(char ryad[])
{
cout << ryad << endl;
}
int main()
{
show_ryad("Hello, C++!");
show_ryad("Good Bye, C++!");
getch();
return 0;
}
```

Як бачите, функція **show_ryad** трактує параметр символьного рядка як масив:

void show_ryad(char ryad[])

Оскільки символ NULL вказує кінець рядка, функція не вимагає параметра, що задає кількість елементів у масиві. Замість цього функція може визначити останній елемент, просто знайшовши в масиві символ NULL.

Як ви вже знаєте, функції C++ часто використовують символ NULL для визначення кінця рядка. Наступна програма містить функцію з ім'ям **str_len**, що шукає у рядку символ NULL для визначення кількості символів у ньому. Далі функція використовує оператор **return** для повернення значення довжини рядка до функції, що викликала. При виконанні програми кілька різних символьних рядків передаються у функцію, і на екран виводиться довжина кожного з них:

```
#include<iostream.h>                    //Програма 6.4
#include<conio.h>
int str_len(char ryad[])
{
int i;
for (i = 0; ryad[i] != '\0'; i++);
return(i);
}
int main()
{
char title[] ="Vchimosya programuvannyu";
char lesson[]="Simvolni ryadki";
```



```
cout<<"Ryadok: "<<title<<" mae "<<
str_len(title)<<" simvoliv"<<endl;
cout<<"Ryadok: "<<lesson<<" mae "<<
str_len(lesson)<<" simvoliv"<<endl;
getch();
return 0;
}
```

Як бачите, у функції організовано перегляд рядка, починаючи з першого символу (елемент з індексом 0), і при виявленні NULL перевірка припиняється. На цей момент змінна **i** дорівнює кількості символів у рядку, тому саме її значення повертають оператором **return(i);**.

Далі ви побачите, що наведена функція моделює одну зі стандартних функцій для роботи з рядками.

Питання для самоконтролю:

1. *Що являють собою символьні рядки?*
2. *Чим символьні рядки відрізняються від звичайних масивів?*
3. *Для чого використовується символ **NULL**?*
4. *Чим '**B**' відрізняється від "**B**"?*
5. *Як здійснюється ініціалізація символьного рядка?*
6. *Як здійснюється передавання рядків у функцію?*

Вправа 6-1.

1) Випробуйте програму 6.1. Уважно дослідіть роботу програми. Виведіть маленькі літери латинського алфавіту. (Не виводьте за допомогою цієї програми українських літер!).
2) Випробуйте програми 6.2. та 6.3. Перероблять програму 6.3 так, щоб параметром, переданим функції, була змінна.
3) Випробуйте програму 6.4. Змініть значення рядкових величин.

Збережіть програми, створивши у власній папці нову папку *wpr6-1*.

6.2. Вказівники і символьні рядки

Другий спосіб визначення рядка – це використання вказівника на символ. Оголошення **char *b;** задає змінну **b**, що може містити адресу деякого об'єкта. Однак у цьому випадку компілятор не резервує місце для зберігання символів і не ініціалізує змінну **b** конкретним значенням. Зробити це можна, наприклад, присвоївши **b** вказівник на вже існуючий символьний масив або динамічно виділивши пам'ять під новий масив (див. тему 5.6):

```
#include<iostream.h>                    //Програма 6.5
#include<conio.h>
int main()
{
char Ryad[]="Hello, world!";          //оголошуємо
                                    //символьний масив
char *b;          //оголошуємо вказівник на символ
b = &Ryad[7]; //тепер b вказує на 8-й символ Ryad
*b = 'W';          //присвоюємо першому елементу b
                                          //символ 'W'
cout << b;  //виводимо рядок b на екран (World!)
getch();
return 0;
}
```

Функції для роботи з рядками

Заголовковий файл <*string.h*> містить набір функцій, які забезпечують ефективну роботу з рядками. Розглянемо деякі з них.

`char* strcat(char*Ryad1, const char*Ryad2);`

– об'єднує рядки **Ryad1** і **Ryad2** і записує результат у рядок **Ryad1**.

`char* strcpy(char*Ryad1, const char*Ryad2);`

– копіює рядок **Ryad2** у рядок **Ryad1**.

`int strlen(const char*  Ryad);`

– повертає довжину рядка (кількість символів). Символ '**\0**' не враховується.

`int strcmp(const char*Ryad1,const char*Ryad2);`

– порівнює рядки **Ryad1** і **Ryad2**. Повертає 0, якщо рядки рівні, число менше від нуля, якщо **Ryad1** < **Ryad2** і число більше від нуля, якщо **Ryad1** > **Ryad2**.

`char* strlwr(char*Ryad);`

– перетворює великі літери рядка на малі (обробляє тільки латинські букви).

`char* strupr(char*Ryad);`

– перетворює малі літери рядка на великі (обробляє тільки латинські букви).

`char* strset (char*Ryad, char Simvol);`

– заповнює рядок зазначеним при виклику функції символом.

`char* strrev(char* Ryad);`

– міняє порядок літер у рядку на протилежний.

Приклад 1. *Дослідження функцій для роботи з рядками.*

```
#include<string.h>                    //Програма 6.6
#include<iostream.h>
#include<conio.h>
#include<windows.h>
char*Ukr(const char* text);//Прототип функції
int main()
{
 char Ryad1[50];char Ryad2[50];
 cout<<Ukr("Уведіть 1-е слово:"); cin>>Ryad1;
 cout<<Ukr("Уведіть 2-е слово:"); cin>>Ryad2;
 cout<<Ryad1<<"  "<<Ryad2<<endl;
 //Обчислення довжини рядків
 int x=strlen(Ryad1);
 int y=strlen(Ryad2);
 cout<<" Ryad1="<<x<<" Ryad2="<<y<<endl;
 //Об'єднання рядків
 strcat(Ryad1, Ryad2);
 cout<<"Ryad1="<<Ryad1<<" Ryad2="<<Ryad2<<endl;
 //Перетворення малих літер рядка на великі
 strupr(Ryad1); strupr(Ryad2);
 cout<<"Ryad1="<<Ryad1<<" Ryad2="<<Ryad2<<endl;
 //Перетворення великих літер рядка на малі
 strlwr(Ryad1);strlwr(Ryad2);
 cout<<"Ryad1="<<Ryad1<<" Ryad2="<<Ryad2<<endl;
 //Копіювання рядка 2 у рядок 1
 strcpy(Ryad1, Ryad2);
 cout<<"Ryad1="<<Ryad1<<" Ryad2="<< Ryad2<<endl;
 //Заповнення рядка 2 зірочками
 strset (Ryad2, '*');
 cout<<"Ryad2="<<Ryad2<<endl;
 //Зміна порядку символів у рядку на протилежний
 cout<<strrev(Ryad1)<<endl;
 getch();return 0;
}
// функція для використання українських літер
char bufUkr [256];
char*Ukr(const char* text)
{
CharToOem(text, bufUkr); return bufUkr;
}
```

Дослідивши цю програму, ви навчитеся працювати з рядковими величинами в С++. А підключивши бібліотеку <*windows.h*> ви

зможете використати український алфавіт, «пропустивши» рядок через функцію **CharToOem()**.

Приклад 2. *Написати програму для порівняння двох рядків.*

Всі символи для обробки у комп'ютері подаються у вигляді числових кодів, які разом складають кодову таблицю. Коли комп'ютер порівнює два рядки, він насправді порівнює коди відповідних символів у рядках, починючи з перших. За кодами перших двох неоднакових символів робиться висновок про результат порівняння. Наприклад, «ABCD» < «ABCE», «ABC» < «ABCD»

Перед написанням програми розглянемо ще одну функцію C++. Досить часто виникає потреба уводити в масив цілий рядок тексту. З цією метою використовують функцію **cin.getline**. Зверніть увагу, що назва функції складається з двох частин, відокремлених крапкою. Це означає, що функція **getline** є **методом** об'єкту **cin**.

Функція **cin.getline** вимагає три аргументи: масив символів, у якому буде зберігатися рядок тексту, довжина й символ-обмежувач. Наприклад, у такому фрагменті програми

```
char ryad1[32];
cin.getline(ryad1, 32, '\n');
```

оголошується масив **ryad1** з 32 символів, а потім із клавіатури в цей масив зчитується рядок тексту. Функція припиняє зчитування символів у випадках:

- якщо зустрічається символ-обмежувач '**\n**';
- якщо уводиться вказівник кінця файлу;
- якщо кількість уведених символів виявляється на один менше, ніж зазначене в другому аргументі (останній символ у масиві резервується для завершального нульового символу).

Якщо зустрічається символ-обмежувач, він зчитується й відкидається. Третій аргумент **cin.getline** має '**\n**', як значення за замовчуванням, так що попередній виклик функції міг бути написаний у такому вигляді: **cin.getline(ryad1, 32);**

```
#include<iostream.h>                  //Програма 6.7
#include<string.h>
#include<conio.h>
int main()
{
 int len;                           // довжина рядка
 char r[81];                 // місце зберігання рядка
 char *r1,*r2;
 cout<<"Vvedi 1 ryadok: ";
 cin.getline(r, 80);     // уведення першого рядка
 len = strlen(r);      // визначення довжини рядка
 r1 = new char[len + 1];   // динамічне виділення
                           //пам'яті під рядок r1
 strcpy(r1, r);            //копіювання уведеного
                               //рядка в рядок r1
 cout<<"Vvedi 2 ryadok: ";
 cin.getline(r, 80);     //уведення другого рядка
 len = strlen(r);
 r2 = new char[len + 1];    //динамічне виділення
                           //пам'яті під рядок r2
 strcpy(r2, r);
 if(strcmp(r1, r2) > 0)          //який з уведених
                                 //рядків більший?
    cout<<"Ryad r1:\t"<<r1<<"\n\t>\n"<<
    "Ryad r2:\t"<<r2<<endl;
 else if(strcmp(r1, r2) == 0)
          cout<<"Ryad r1:\t"<<r1<<"\n\t=\n"<<
          "Ryad r2:\t"<<r2<<endl;
      else
          cout<<"Ryad r1:\t"<<r1<<"\n\t<\n"<<
          "Ryad r2:\t"<<r2<<endl;
 // видалення рядків з пам'яті
 delete []r1; delete []r2;
getch();return 0;
}
```

Питання для самоконтролю:

1. *Як визначити рядок за допомогою вказівника?*
2. *Назвіть основні функції для роботи з рядками?*
3. *Як увести в масив повний рядок тексту?*

Вправа 6-2.

1) Випробуйте програму 6.5. Уважно проаналізуйте її роботу.
2) Напишіть програму, яка в уведеному із клавіатури рядку перетворить малі літери англійського алфавіту на великі.
3) Напишіть програму, яка об'єднає в одне 2 уведені слова, і виведе на екран довжину створеного рядка.

Збережіть програми, створивши у власній папці нову папку *wpr6-2*.

6.3. Практична робота №11 «Опрацювання рядкових величин»

1) Написати програму, яка запитає у користувача його ім'я, а потім привітається з ним, вказавши, крім цього, зі скількох літер складається уведене ім'я. А якщо через пропуск уведено і прізвище – його програма проігнорує.
2) Написати програму, яка порівнює уведені з клавіатури 3 слова і виводить на екран результати порівняння.
3) Написати функцію, яка виводитиме на екран уведений у головній програмі рядок посимвольно зі звуковим супроводом (див. п.1.1. «спеціальні символи»).

Збережіть програми, створивши у власній папці нову папку *pr11*.

6.4. Тематичне оцінювання з теми «Рядкові величини»

1) Написати программу для підрахування кількості входжень у заданий текст символу '**z**'.
2) Написати програму, яка перевірятиме, чи задане слово читається однаково з початку в кінець і з кінця до початку.
3) Написати програму, яка при введенні цифри від 0 до 9 видає її текстовий еквівалент: 0 – (нуль), 1 – (один)…

Збережіть програми, створивши у власній папці нову папку *ta4*.

Питання для самоконтролю:

1. *Чи може довжина нединамічного рядка бути змінною? Як описується нединамічний рядок?*
2. *Чи може довжина динамічного рядка бути змінною? Як оголошується динамічний рядок?*
3. *Чи може довжина динамічного рядка бути константою?*
4. *Як оголосити рядок з 7 символів (наприклад, рядок, ініціалізований словом «команда»)?*
5. *Чи можна присвоїти один рядок іншому? Як це зробити?*
6. *Чи можна увести рядок (наприклад, «Сьогодні гарна погода») за допомогою операції >>?*
7. *Чи контролюється вихід за межі рядка й відсутність нуль-символу?*

7. Файлові операції

7.1. Виведення та читання файлів

Виведення у файловий потік

cout являє собою об'єкт типу **ostream** (вихідний потік). Заголовковий файл *iostream.h* визначає вихідний потік cout. Аналогічно, заголовковий файл *fstream.h* визначає клас **вихідного файлового потоку** з ім'ям **ofstream**. Для запису на диск файлу потрібно оголосити об'єкт типу **ofstream**, указавши ім'я необхідного вихідного файлу як символьний рядок:

```
ofstream fout ("FILENAME.EXT");
```

Якщо при оголошенні об'єкта типу **ofstream** вказати ім'я файлу, на диску буде створений новий файл із зазначеним ім'ям, або перезаписаний файл із таким же ім'ям, якщо він уже існує. У програмі 7.1 створюється об'єкт типу **ofstream** і потім використовується операція **<<** для виведення трьох рядків тексту у файл *output.txt*:

```
#include<iostream.h>                    //Програма 7.1
#include<fstream.h>
int main(){
ofstream fout ("output.txt");
fout<<"Prog na C++ "<<" klas 'Server'"<<endl;
fout<<"Bilgorod-Dnistrovsky"<<endl;
fout<<"2007 year."<< endl;
return 0;
}
```

Запустіть на виконання цю програму. Якщо ви працюєте в середовищі WINDOWS, після цього можете використати «Блокнот» для перегляду вмісту щойно створеного файлу *output.txt* (з папки, де знаходиться ваш проект):

```
Prog na C++  klas 'Server'
Bilgorod-Dnistrovsky
2007 year.
```

Читання із вхідного файлового потоку

Уведення з файлу можна реалізувати, використовуючи об'єкти типу **ifstream** (клас вхідних файлових потоків). Як і при

виведенні, слід просто створити об'єкт, передаючи йому, як параметр, необхідне ім'я файлу:

```
ifstream fin("FILENAME.EXT");
```

Після цього з файлу можна читати дані.

У наступній програмі відкривається файл *output.txt*, створений за допомогою програми 7.1 і з нього читаються дані:

```
#include<iostream.h>                    //Програма 7.2
#include<fstream.h>
#include<conio.h>
int main()
{
ifstream fin("output.txt");
char one[64], two[64], three[64];
fin >> one;
fin >> two;
fin >> three;
cout << one << endl;
cout << two << endl;
cout << three << endl;
getch();
return 0;
}
```

Якщо ви відкомпілюєте й запустите цю програму, то вона замість усього тексту відобразить тільки перші три слова:

```
Prog
na
C++
```

Це відбувається тому, що, подібно до **cin**, вхідні файлові потоки використовують порожні символи (пропуск), щоб визначити, де закінчується одне значення й починається інше.

Читання цілого рядка файлового уведення

Вище ви ознайомились з функцією **cin.getline** для читання цілого рядка із клавіатури. Об'єкти типу **ifstream** також можуть використовувати **getline** для читання рядка файлового уведення. Наступна програма використовує функцію **getline** для читання всіх трьох рядків файлу:

```
#include<iostream.h>                    //Програма 7.3
#include<fstream.h>
#include<conio.h>
```



```
int main()
{
ifstream fin ("output.txt");
char one[64], two[64], three[64];
fin.getline(one, sizeof(one));
fin.getline (two, sizeof(two));
fin.getline (three, sizeof(three)) ;
cout << one << endl;
cout << two << endl;
cout << three << endl;
getch();
return 0;
}
```

Вміст файлу був прочитаний успішно, тому що програміст знав, що файл складається з трьох рядків. Однак у більшості випадків ми не знаємо, скільки рядків містить файл.

Визначення кінця файлу

Для успішного читання довільних файлів потрібно знати, коли зустрінеться кінець файлу. Щоб виявити кінець файлу, у програмі використовують функцію **eof** потокового об'єкту. Ця функція повертає значення 1, якщо зустрівся кінець файлу, в іншому випадку повертає 0. Використаємо цикл **while** для безперервного читання вмісту файлу:

```
while (!fin.eof()) {// Оператори }
```

У цьому випадку програма буде продовжувати виконувати цикл, поки функція **eof** повертає **false** (тобто, 0). Зверніть увагу на використання логічної операції НЕ (**!**).

У наступній програмі завдяки використанню функції **eof** вмісту файлу *output.txt* читається і виводиться повністю (поки не буде досягнутий кінець файлу):

```
#include<iostream.h>                    //Програма 7.4
#include<fstream.h>
#include<conio.h>
int main()
{
ifstream fin("output.txt");
char line[64];
```

```
while (!fin.eof())
      {
      fin.getline(line,sizeof(line));
      cout << line << endl;
      }
getch();return 0;
}
```

А у наступній програмі вміст файлу читається по одному слову за один раз, поки не зустрінеться кінець файлу:

```
#include <iostream.h>                    //Програма 7.5
#include <fstream.h>
#include<conio.h>
int main()
{
ifstream fin("output.txt");
char text[64] ;
     while (!fin.eof())
     {
       fin >> text;
       cout << text << endl;
     }
getch();
return 0;
}
```

У програмах 7.4 та 7.5 інформація виводиться на екран відразу після читання з файлу. Це може призвести до помилки у роботі програми: навіть після спроби читання **за межами файлу** буде здійснене виведення на екран. Таких ситуацій потрібно уникати, тому розглянемо, як це можна зробити.

Обробка помилок при виконанні файлових операцій

Програмісти-початківці часто припускаються помилок при запису у файл чи зчитуванні з файлу. Наприклад, робиться спроба відкрити неіснуючий файл або на диску не вистачає місця для запису файлу, робляться спроби читання даних за межами файлу тощо. Щоб допомогти програмам стежити за помилками, використовується функція **`fail`** файлового об'єкта. Якщо у процесі файлової операції помилок не було, функція поверне **`false`** (0). Однак, якщо трапилася помилка, функція **`fail`** поверне **`true`**.

Таким чином, відразу після файлової операції слід переконатися, що вона пройшла успішно, а після цього приймати рішення про подальші дії. У програмі 7.6 функція **`fail`** використовується для уникнення різних помилкових ситуацій:

```
#include <iostream.h>                    //Програма 7.6
#include <fstream.h>
#include<conio.h>
int main()
{
char line[256];
ifstream fin("output.txt");
  if (fin.fail())
  cout<<"Error files";
  else
  {
    while((!fin.eof())&&(!fin.fail()))
    {
     fin.getline(line, sizeof(line)) ;
     if (!fin.fail())
       cout<<line<<endl;
    }
  }
getch();return 0;
}
```

Спробуйте видалити файл *output.txt* з папки, де знаходиться ваша програма, і в результаті отримаєте повідомлення **`"Error files"`**. Для читання файлу, збереженого у іншій папці (наприклад, на диску *D:* у папці *priklad*), треба при оголошенні об'єкта вказати шлях до файлу у такий спосіб:

```
ifstream fin("d:\\priklad\\output.txt");
```

І, нарешті, коли для продовження роботи файл більше не потрібен, його слід закрити, використавши функцію (метод):

```
fin.close();
```

При цьому всі дані зберігаються на диску, а дані про цей файл оновлюються у файловій системі.

Питання для самоконтролю:

1. *Як записати файл на диск?*
2. *Як прочитати файл з диску?*
3. *Як прочитати повний рядок тексту з файлу?*

4. *Як здійснюється перевірка помилок при виконанні файлових операцій?*
5. *Як зберегти файл у потрібну папку на диску?*
6. *За допомогою якої функції закривають непотрібні файли?*

Вправа 7-1.

1) Випробуйте програми 7.1. та 7.2. Чому друга програма не виводить вміст усього файлу?
2) Випробуйте програми 7.3.–7.5. Яка з них найкраще підходить для виведення тексу невідомої довжини на екран?
3) Запустіть програму 7.5, попередньо видаливши з папки файл *output.txt*. Спробуйте пояснити результат.
4) Випробуйте програму 7.6. Спробуйте видалити створений програмою 7.1. файл з папки, де знаходиться ваша програма. Як після цього змінилась робота програми 7.6.? Помістіть файл *output.txt* у папку: *d:\priklad*, зробіть відповідні зміни у програмі 7.6. та випробуйте її.

Збережіть програми, створивши у власній папці нову папку *wpr7-1*.

7.2. Приклад використання файлових операцій

Спробуємо переробити задачу (п. 5.10, приклад 1) для уведення вхідних даних з файлу (погодьтеся, що уведення при налагодженні програми двовимірних масивів чисел з клавіатури – доволі кропітка робота). Тоді, використавши текстовий редактор «Блокнот», ми зможемо швидко підготувати декілька різних наборів даних для тестування програми.

```
#include <iostream.h>            //Програма 7.7
#include <fstream.h>
#include <iomanip.h>
#include <conio.h>
int main()
{
ifstream fin("c:\\pr\\input.txt");
if (!fin)
{
  cout<<"Fajl input.txt ne znajdeno"<<endl;
  getch();return 1;
}
ofstream fout("c:\\pr\\output.txt");
```



```
if (!fout)
{
 cout<<"Nemoglivo vidkryty dlya zapisu"<<endl;
 getch();return 1;
}
int ryad, stovp;
fin>>ryad;          //зчитування кількості рядків
fin>>stovp;       //зчитування кількості стовпців
// виділення пам'яті під масив
int i, j;
int **a = new int *[ryad];
for( i = 0; i < ryad; i++)
   a[i] = new int [stovp];
// уведення масиву
for (i = 0; i < ryad; i++)
  for (j = 0; j < stovp; j++)
    fin >> a[i][j];
// виведення масиву
for (i = 0; i < ryad; i++)
{
  for (j = 0; j < stovp; j++)
     fout<<setw(4)<<a[i][j]<<"  ";
  fout<<endl;
}
//аналіз масиву
int n = -1;
bool dodat;
for (j = 0; j < stovp; j++)
{
//перегляд по стовпцях
dodat = true;
//аналіз елементів стовпця
for (i = 0; i < ryad; i++)
  if (a[i][j] < 0)
  {
    dodat = false;break;
  }
  if (dodat)
  {
    n = j;break;
  }
}
```

```
if (n ==-1  ) fout<<"Stovpciv nema"<<endl;
else fout<<"Nomer stovpcya: "<<n<<endl;
cout<<"Shlyah do fajlu rezultatu: c:\\pr";
getch();
return 0;
}
```

Уведення розмірності масиву і його елементів виконується з файлу *input.txt*, розташованого в папці *c:\pr*, а результати виводяться у файл *output.txt*. У програмі визначений об'єкт **fin** класу вхідних файлових потоків й об'єкт **fout** класу вихідних файлових потоків.

Після визначення об'єктів перевіряється успішність їхнього створення. Це особливо важливо робити для вхідних файлів, щоб уникнути помилки в імені або місці розташування файлу.

Якщо програма завершується успішно, то на екран виводиться повідомлення «**Shlyah do fajlu rezultatu: c:\\pr**». Завдяки цьому користувач вашої програми зрозуміє, де шукати результати роботи програми.

Вхідний файл *input.txt* можна створити в будь-якому текстовому редакторі. Він, природно, повинен існувати на момент першого запуску програми. На розташування й формат вихідних даних у файлі ніяких обмежень не накладається.

Для перевірки роботи програми підготуємо файл *input.txt*, наприклад:

```
3 4
-4   -5    3     9
 5    0    3     8
 8    1    7     3
```

Тут перші два числа вказують на те, що масив складається з 3-х рядків та 4-х стовпців, а далі розміщений сам масив чисел. Помістимо цей файл у папку *c:\pr* і запустимо програму на виконання. У результаті у тій же папці ми знайдемо файл *output.txt* з таким вмістом:

```
   -4    -5     3     9
    5     0     3     8
    8     1     7     3
Nomer stovpcya: 2
```


Питання для самоконтролю:

1. *Як вивести дані у файл на диску?*
2. *Як здійснюється читання з файлу?*
3. *Як прочитати файл невідомої довжини?*
4. *Як контролювати помилки при виконанні файлових операцій?*
5. *Чи можна для тестування програм вхідні дані підготувати за допомогою текстового редактора?*
6. *Як прочитати такі дані?*

Вправа 7-2.

1) Випробуйте програму 7.7. Підготуйте у текстовому редакторі дані для тестування програми (коли є шуканий стовпець, та коли його немає).
2) Підготуйте у файлі *input.txt* масив даних розміром 10x10 та випробуйте програму. Чи можна використовувати цю програму для читання даних у двовимірні масиви будь-якого розміру? Чи можна це зробити, не використавши динамічне виділення пам’яті?
3) Переробіть програму для знаходження кількості стовпців, що не містять жодного нульового елемента.

Збережіть програми, створивши у власній папці нову папку *wpr7-2*.

8. Зберігання зв'язаної інформації в структурах

8.1. Структури. Структури й функції

З розділу 5 ви довідалися, що масив – це сукупність елементів одного типу і з'ясували, що використання масивів дуже зручне при написанні багатьох програм. Досить часто виникає потреба групувати зв'язану інформацію **різних типів**. Припустимо, що ваша програма працює з інформацією про школярів. Вона повинна відслідковувати дані про прізвище, вік, адресу, телефон, середній бал й т.д. Для зберігання цієї інформації програмі будуть потрібні змінні типу **char**, **int**, **float**, а також символьні рядки.

*Для зберігання зв'язаної інформації різних типів у програмі використовують **структури**.*

Оголошення структури

Структура визначає шаблон, за допомогою якого ваша програма може пізніше оголосити одну або декілька змінних. Інакше кажучи, у програмі слід спочатку визначити структуру, а потім можна оголошувати змінні типу цієї структури. Для визначення структури використовують ключове слово **struct**:

```
struct name {
    int name_1; // Оголошення елементів структури
    float name_2;
} var;                          // Оголошення змінної
```

У цьому прикладі визначена структура з ім'ям **name**, яка об'єднує два елементи: **name_1** типу **int** та **name_2** типу **float**. Відразу ж оголошена змінна **var** типу **name**.

Для ініціалізації структури значення її елементів можна перерахувати у фігурних дужках у порядку їхнього опису:

```
struct {
    char fio[30];
    int date, code;
    double zarplata;
} worker={"Petrov", 28, 118, 500.55};
```


Зверніть увагу, що в цьому випадку тип структури не має власного імені, тому не можна буде оголосити структуру такого ж типу в іншому місці програми.

Як і дані інших типів, структури можна об'єднувати в одно- та баготовимірні масиви. При ініціалізації масивів структур варто брати у фігурні дужки кожен елемент масиву (з огляду на те, що багатовимірний масив – це масив масивів):

```
struct coordinata{
int x, y;
} coord[2][3]={
        {{1,1},{3,2},{5,4}}, //масив coord[0]
        {{8,0},{1,7},{1,0}}, //масив coord[1]
  };
```

Наступний приклад демонструє визначення структури, що міститиме інформацію про учня:

```
struct uchen {
  char name[64] ;
  int riknar;
  float serball;
  char phone[10];
};
```

Після визначення структури, у програмі також можна оголосити змінні типу цієї структури, як показано нижче:

```
uchen shkolyar, star_shkolyar, new_shkolyar;
```

У цьому випадку оператор створює три змінні типу структури **uchen**.

Використання елементів структури

Структура дозволяє групувати інформацію різних типів, що складає елементи (інша назва – поля) структури, в одній змінній. Щоб присвоїти значення елементу або звернутися до значення елемента, використовуватимемо операцію «вибір елемента» – крапка (**.**). Присвоїмо значення різним елементам змінної з ім'ям **shkolyar** типу **uchen**:

```
strcpy(shkolyar.name, "Ivanov Sergiy");
shkolyar.riknar = 1998;
shkolyar.serball=7.5;
strcpy(shkolyar.phone, "25-56-89");
```

Наступна програма ілюструє використання структури типу **uchen**:

```
#include<string.h>            //Програма 8.1
#include<iostream.h>
#include<conio.h>
int main()
{
struct uchen {
      char name[64] ;
      int riknar;
      float serball;
      char phone[10];
}shkolyar;
// Копіювати ім'я в поле-рядок
strcpy(shkolyar.name, "Ivanov Sergiy");
// Присвоювання значень числовим полям
shkolyar.riknar = 1998;
shkolyar.serball=7.5;
// Копіювати номеру телефону в поле-рядок
strcpy(shkolyar.phone, "25-56-89");
cout<<"Uchen: "<< shkolyar.name << endl;
cout<<"Rik nar : "<< shkolyar.riknar << endl;
cout<<"Ser. bal.: "<< shkolyar.serball << endl;
cout<<"Tel: "<< shkolyar.phone << endl;
getch();
return 0;
}
```

Як бачите, надати значення елементам типів **int** та **float** дуже просто: використовується оператор присвоювання. Однак зверніть увагу, що для копіювання символьних рядків у елементи **name** й **phone** довелося скористатись функцією **strcpy**.

*Отже, звертання до поля структури (наприклад, **shkolyar.name**) використовують так само, як і імена простих змінних такого ж типу.*

Структури й функції

Якщо функція не змінює структуру, ви можете передати структуру у функцію за ім'ям. Наприклад, у програмі 8.2 функція **show_uchen** використовується для виведення елементів структури типу **uchen**:


```
#include<string.h>            //Програма 8.2
#include<iostream.h>
#include<conio.h>
struct uchen {
    char name[64] ;
    int riknar;
    float serball;
    char phone[10];
};
//функція виведення на екран
void show_uchen(uchen shkolyar)
{
cout<<"Uchen: "<< shkolyar.name << endl;
cout<<"Rik nar.: "<< shkolyar.riknar << endl;
cout<<"Ser. bal: "<< shkolyar.serball << endl;
cout<<"Tel: "<< shkolyar.phone << endl;
}
int main()
{
uchen shkolyar;   //оголошення змінної-структури
//заповнення даними
strcpy(shkolyar.name, "Ivanov Sergiy");
shkolyar.riknar = 1998;
shkolyar.serball=7.5;
strcpy(shkolyar.phone, "25-56-89");
//Звернення до функції виведення на екран
show_uchen(shkolyar);
getch();return 0;
}
```

Зверніть увагу, що програма тепер визначає структуру **uchen** поза головною функцією **main** і до функції **show_uchen**. Оскільки функція оголошує змінну **shkolyar** типу **uchen**, визначення структури **uchen** має розташовуватися до функції.

Функції, що змінюють елементи структури

Як ви знаєте, якщо функція змінює параметр, його належить передавати у функцію за допомогою адреси. Це повністю стосується й структур: якщо функція змінює елемент структури, слід передавати цю структуру у функцію за допомогою адреси. Для

передачі змінної типу структури за допомогою адреси перед ім'ям змінної ставлять знак операції взяття адреси (**&**):

```
fun (&shkolyar);
```

Всередині функції, що змінює один або кілька елементів, належить працювати з вказівником.

Наприклад, у наступній програмі структура типу **uchen** передається у функцію з ім'ям **get_uchen_serball**, що запитує в користувача середній бал учня й потім присвоює уведене число елементу структури **serball**. Щоб змінити елемент, функція працює з вказівником на структуру:

```
#include<string.h>              //Програма 8.3
#include<iostream.h>
#include<conio.h>
//Оголошення структури
struct uchen {
    char name[64] ;
    int riknar;
    float serball;
    char phone[10];
};
//функція виведення на екран
void show_uchen(uchen shkolyar)
{
cout<<"Uchen:"<<shkolyar.name<<endl;
cout<<"Rik nar:"<<shkolyar.riknar <<endl;
cout<<"Seredniy ball:"<<shkolyar.serball<<endl;
cout<<"Tel:"<< shkolyar.phone<<endl;
}
//функція уведення середнього балу
void get_uchen_serball(uchen *shkolyar)
{
cout<<"Uvedit seredniy ball:";
cin>>shkolyar->serball;
}
int main()
{
uchen  shkolyar;  //оголошення змінної-структури
```



```
//заповнення даними
strcpy(shkolyar.name, "Ivanov Sergiy");
shkolyar.riknar = 1998;
shkolyar.serball=7.5;
strcpy(shkolyar.phone, "25-56-89");
//звернення до функції виведення на екран
show_uchen(shkolyar);
//звернення до функції заміни даних
get_uchen_serball(&shkolyar);
//звернення до функції виведення на екран
show_uchen(shkolyar);
getch();return 0;
}
```

Як бачите, у головній функції **main** програма передає змінну **shkolyar** типу структури у функцію **get_uchen_serball** за допомогою адреси. Всередині функції **get_uchen_serball** значення, уведене користувачем, присвоюється елементу **serball** за допомогою оператора «вибір елемента» (**->**):

```
cin >> shkolyar->serball;
```

Питання для самоконтролю:

1. *Чи можливо у одному масиві зберігати елементи різних типів? А у структурі?*
2. *Як оголосити структуру?*
3. *Як оголосити змінні типу структури?*
4. *Як присвоїти значення елементу або звернутися до значення елемента структури?*
5. *Як передати структуру у функцію?*
6. *Як забезпечити можливість змінювати значення полів структури-аргументу при роботі функції?*

Вправа 8-1.

1) Випробуйте програму 8.1.
2) Випробуйте програму 8.2. Спробуйте написати функцію для уведення середнього балу у поле структури **serball**. Чи можна це зробити, не використовуючи вказівники?
3) Випробуйте програму 8.3.

Збережіть програми, створивши у власній папці нову папку *wpr8-1*.

8.2. Практична робота №12 «Файлові операції. Структури»

1) Увести структуру для реєстрації автомобіля. Вона повинна мати такі поля (типи доберіть самостійно):
   дата реєстрації (рік);
   марка машини;
   рік випуску;
   колір;
   номер.
2) Зареєструвати 5 машин (у масиві), вивести на екран дані про марку і рік випуску машин.
3) Запишіть у файл вірш Лесі Українки:
   З тихим плескотом на берег
   Рине хвилечка перлиста;
   Править хтось малим човенцем,
   Стиха весла піднімає,
   І здається, що з весельця
   Щире золото спадає.
   Організуйте читання з файлу і виведення на екран цього вірша (використайте функцію для виведення українських літер).

Збережіть програми, створивши у власній папці нову папку *pr12*.

Питання для самоконтролю:

1. *У чому полягає відмінність структури від масиву?*
2. *Що входить у поняття поля?*
3. *Як описати й ініціалізувати структуру?*
4. *Як відбувається звертання до елемента структури?*

9. Створення графічних зображень

9.1. Основи об'єктно-орієнтованого програмування

Досі ми працювали тільки з консольними програмами, щоб вивчити основи мови програмування C++. Але режим Console Wizard не дозволить нам вивчити графічні можливості мови. Тому вивчення графіки буде поєднане зі знайомством з основами **об'єктно-орієнтованого програмування** під Windows. Розглянемо детальніше середовище візуальної розробки Borland C++ Builder .

Середовище візуальної розробки програми

Після запуску системи ми побачимо на екрані п'ять вікон:

1. **Головне вікно**. Головне керуюче вікно системи C++ Builder розміщується у верхній частині екрана. Воно містить засоби керування створенням програми. В ньому є:

a) ***Рядок заголовка.*** Він не відрізняється від подібних рядків інших вікон. У рядку заголовка записане ім'я відкритого в середовищі проекту (за замовчуванням – **`Project1`**).

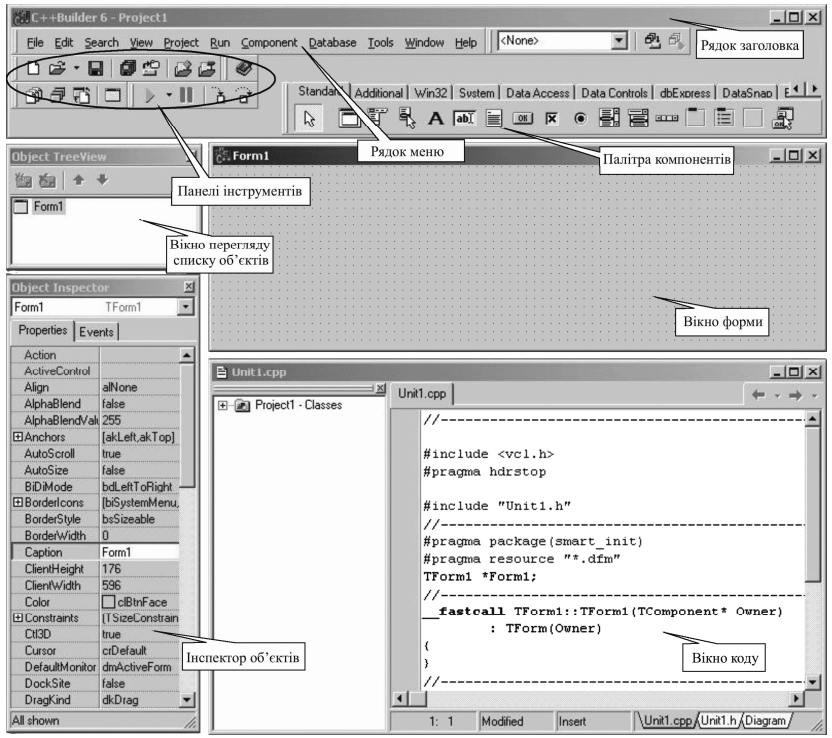
b) ***Рядок меню.*** Під рядком заголовка вікна системи розташовується рядок меню, що містить команди системи C++ Builder. Крім рядка меню програма має й інші елементи керування: командні кнопки, кнопки інструментів та інші. Кнопками зручно користуватися для виконання операцій, але кнопка є не для кожної операції, а тільки для тих, які зустрічаються особливо часто. Зате за допомогою рядка меню можна знайти будь-яку з допустимих команд.

c) ***Панелі інструментів***. Під рядком меню розташовуються панелі інструментів із кнопками. У системі C++ Builder їх декілька. Кнопки панелей інструментів забезпечують доступ до команд, що часто зустрічаються. Щоб довідатися, як називається та або інша кнопка, треба навести на неї вказівник миші й почекати, поки поруч із ним з'явиться спливаюча підказка.

d) ***Палітра компонентів.*** На ній зібрані кнопки, що відповідають компонентам вікна програми – «цеглинкам», з яких збирають програми у C++Builder. Зверніть увагу на те, що палітра компонентів містить багато вкладок. Кожна з них надає свій набір компонентів. Перейдемо до розгляду інших вікон.

2. **Вікно перегляду списку об'єктів**. Містить список компонентів, використовуваних у програмі, й дозволяє легко здійснювати перехід від одного до іншого.

3. **Інспектор об'єктів**. З його допомогою налагоджують об'єкти, використовувані в програмі. У цьому вікні ми побачимо властивості кожного з об'єктів і зможемо їх змінити. Це ж вікно використовується для вибору й налагодження подій, на які будуть



реагувати об'єкти нашої програми. З його допомогою створюють або вибирають потрібні функції обробки подій (іноді їх називають **процедурами обробки подій**).

4. **Вікно форми**. Форма – це заготовка вікна майбутньої програми. Більшість програм містить хоча б одне вікно й, отже, одну форму. Заготовка першого вікна називається **`Form1`**. Якщо в програмі буде два вікна, то заготовка другого буде називатися **`Form2`** і так далі. Стандартну назву, звичайно, можна змінити. На малюнку вікно форми порожнє, але, створюючи програму, ми додамо в нього потрібні елементи керування.

5. **Вікно коду**. Останнє з відкритих вікон містить код (текст) програми. Значну частину тексту програми система C++ Builder

формує автоматично. Навіть «порожня» програма для Windows складається з декількох тисяч операторів мови C++. Проте, коли нам доведеться додавати оператори у свою програму, ми будемо робити це саме у вікні коду. Деякі оператори система C++ Builder додасть сама, інші ми уведемо вручну.

Швидка розробка програм та ООП

В основі систем швидкої розробки програм (RAD-систем, **R**apid **A**pplication **D**evelopment – швидка розробка програм) лежить технологія візуального проектування й подійного програмування, суть якої полягає в тому, що середовище розробки бере на себе більшу частину роботи з генерації коду програми, залишаючи програмісту роботу з конструювання діалогових вікон і написання процедур обробки подій.

Швидка розробка програм передбачає підтримку властивостей, методів і подій компонентів у рамках об'єктно-орієнтованого програмування. **Компонент** – це заготовка майбутнього елементу керування. За допомогою компонента можна створити багато схожих **об'єктів**. Спочатку ці об'єкти майже однакові й розрізняються тільки за іменами. Але потім їх треба налаштувати, після чого вони відрізнятимуться й іншими властивостями. Коли програма буде готова, об'єкти стануть **елементами керування**.

C++ Builder якраз і реалізує **візуальну** методику побудови програм за допомогою вибору з «Палітри компонентів» потрібних керуючих елементів та подальшого їх налаштування.

*З кожним компонентом позв'язані **властивості** та **методи**, від яких залежить його вигляд і поведінка. Кожен з компонентів може викликати серію **подій**, які визначають його реакцію на різні впливи.*

Властивості об'єкта – це характеристики, що визначають його вигляд, розміщення й поведінку. Наприклад, властивості **`Width`** й **`Height`** задають розміри (ширину й висоту) форми, властивості **`Top`** та **`Left`** – розміщення форми на екрані, властивість **`Caption`** – текст заголовка. У верхній частині вікна ***Object Inspector*** зазначений об'єкт (ім'я об'єкта), значення властивостей якого відображені нижче.

Методи призначені для виконання певних операцій над об'єктом. Вони по суті являють собою функції, які належать даному

класу об’єктів. Далі ми детальніше розглянемо методи для малювання прямокутників, кіл, еліпсів тощо.

Події зв'язують впливи користувача на компоненти – такі як активізація, натискання кнопок або уведення даних – з вашими функціями, що описують реакції на ці впливи.

Саме об’єктно-орієнтоване програмування з використанням властивостей, методів та подій лежить в основі роботи середовища швидкої, інтуїтивно зрозумілої розробки програм для операційної системи Windows.

Щоб розпочати розробку програми в об’єктно-орієнтованому середовищі, потрібно просто запустити Borland C++ Builder, чи скористатись у ньому командою меню:

File → New → Application

Окрім вікна для уведення коду на екрані з’явиться вікно стартової форми – **Form1**.

Розробка інтерфейсу

Робота над новим проектом, починається зі створення стартової форми – головного вікна програми.

Стартова форма створюється шляхом зміни значень властивостей форми **Form1** і додавання до форми необхідних компонентів.

Основні властивості форми:

Name – ім'я форми. У програмі використовується для керування формою й доступу до компонентів форми.

Caption – текст заголовка.

Width – ширина форми.

Height – висота форми.

Top – відстань від верхньої межі форми до верхньої межі екрана.

Left – відстань від лівої межі форми до лівої межі екрана.

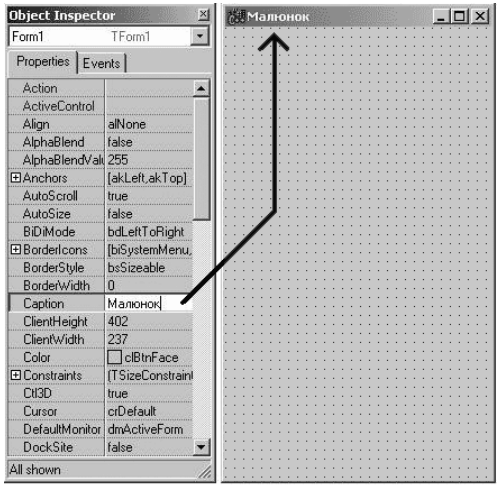
Відстані, розміри форми та інших компонентів задають у пікселях, тобто крапках екрана (англ. PICTure ELement).

Color – колір фону. Колір можна задати, вказавши назву кольору або прив'язку до поточної колірної схеми операційної системи. У другому випадку колір визначається поточною колірною схемою, обраним компонентом прив'язки й міняється при зміні колірної схеми операційної системи.

Font – шрифт. Шрифт, використовуваний за замовчуванням компонентами, що перебувають на поверхні форми. Зміна властивості Font-форми призводить до автоматичної зміни властивості компонента, що розташовується на поверхні форми.

BorderStyle – вид межі. Межа може бути звичайною (**bsSizeable**), тонкою (**bsSingle**), або бути відсутньою (**bsNone**). Якщо у вікна звичайна межа, то під час роботи програми користувач може змінити розмір вікна. Змінити розмір вікна з тонкою межею не можна. Якщо межа відсутня, то на екран під час роботи програми буде виведене вікно без заголовка. Положення й розмір такого вікна під час роботи програми змінити не можна.

BorderIcons – кнопки керування вікном. Ця властивість визначає, які кнопки керування вікном будуть доступні користувачеві під час роботи програми. Значення властивості задається шляхом присвоєння значень уточнюючим властивостям: **biSystemMenu** – визначає доступність кнопки «Згорнути» і кнопки системного меню, **biMinimize** – кнопки «Згорнути», **biMaximize** – кнопки «Розгорнути», **biHelp** – кнопки виклику довідки.



Введення значення властивості Caption

Для зміни значень властивостей об'єктів, у тому числі й форми, використовується вкладка ***Properties*** (властивості) діалогового вікна ***Object Inspector***. У лівій колонці цієї вкладки перераховані властивості обраного об'єкта, у правій – вказані значення властивостей.

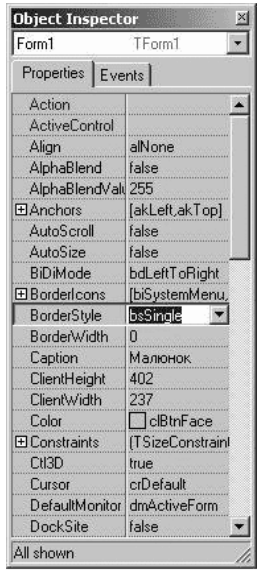
При створенні форми насамперед варто змінити значення властивості **Caption** (заголовок). У нашому прикладі треба замінити текст «**Form1**» на «**Малюнок**». Щоб це зробити, потрібно у вікні ***Object Inspector*** клацнути лівою кнопкою миші в рядку

Caption (у результаті буде виділене значення властивості й з'явиться курсор) і увести текст: **Малюнок** (див. мал.).

Так само можна встановити значення властивостей **Height** й **Width**, які визначають висоту й ширину форми.

Форма – це звичайне вікно. Тому розмір форми можна змінити так само, як розмір будь-якого вікна Windows, тобто шляхом перетягування межі. По закінченні переміщення межі значення властивостей **Height** й **Width** автоматично зміняться.

Положення вікна на екрані після запуску програми відповідає положенню форми під час розробки, що визначається значенням властивостей **Top** (відступ від верхнього краю екрана) і **Left** (відступ від лівого краю екрана). Значення цих властивостей також можна задати шляхом переміщення форми за допомогою миші. Додаткові засоби для керування положенням та розміром вікна програми надає властивість форми **Position**.

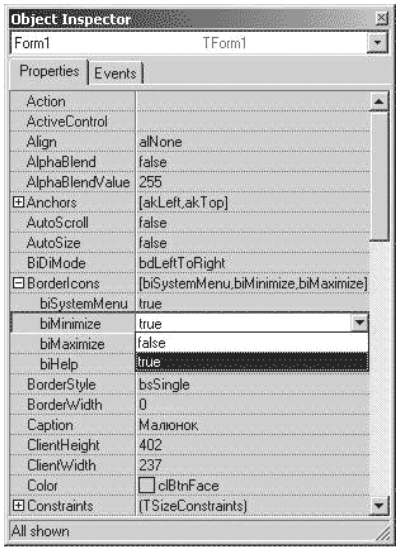


Вибір значення властивості зі списку

При виборі деяких властивостей, наприклад, **BorderStyle**, праворуч від поточного значення властивості з'являється значок списку, що розкривається (▾). Очевидно, що значення таких властивостей можна задати шляхом вибору зі списку (див. мал.).

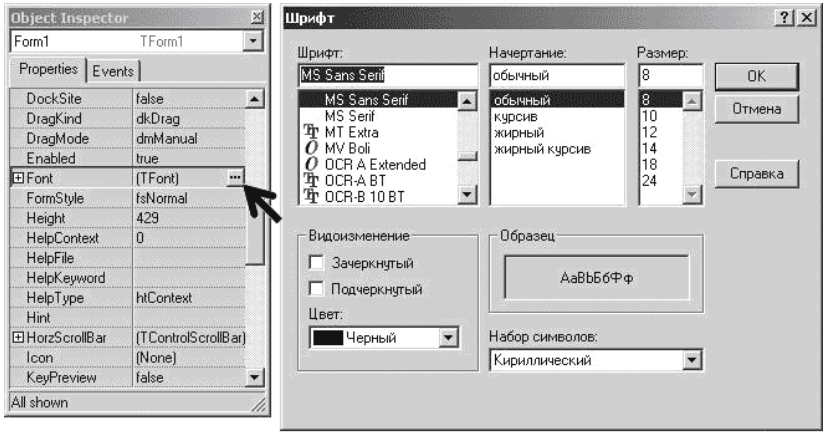
Деякі властивості є складеними, тобто їхнє значення визначається сукупністю значень інших (уточнюючих) властивостей. Наприклад, властивість **BorderIcons** визначає, які кнопки керування вікном будуть доступні під час роботи програми. Її значення складається зі значень властивостей **biSystemMenu**, **biMinimize**, **biMaximize** й **biHelp**, кожне з яких, у свою чергу, визначає наявність відповідної командної кнопки в заголовку вікна під час роботи програми. У вікні ***Object Inspector*** перед іменем складеної властивості розташований значок "+", у результаті клацання мишею на якому

розкривається список уточнюючих властивостей (див. мал.), значення яких можна задати звичайним способом (увести в поле або вибрати зі списку припустимих значень).



Зміна значення уточнюючої властивості

У результаті вибору деяких властивостей (клацання кнопкою миші на властивості) поруч зі значенням властивості з'являється командна кнопка із трьома крапками. Це значить, що задати значення властивості можна в додатковому діалоговому вікні. Наприклад, значення складеної властивості **Font** можна задати у вікні ***Object Inspector*** шляхом уведення значень уточнюючих властивостей, а можна скористатися стандартним діалоговим вікном ***Шрифт***, що з'явиться в результаті клацання на кнопці із трьома крапками (див. мал.).



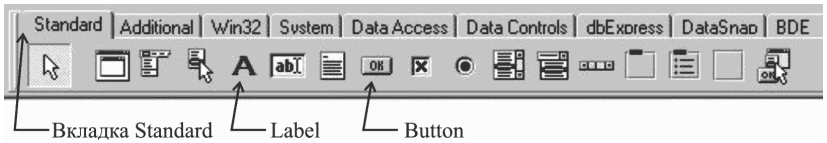
Задання значення складеної властивості Font у діалоговому вікні

Приклад 1. *А тепер спробуємо створити просту програму в середовищі візуального проектування. Ця програма буде виводити повідомлення на екран і матиме кнопку керування – для закриття вікна:*

Як ви вже знаєте, в C++ кожна незавершена програма – це проект. Він включає певну кількість файлів. Серед них, зокрема, файл форми, файл коду і файл проекту, які містять текст програми. Надалі ми будемо вносити зміни лише у файл коду, а інші файли будуть створені автоматично.

Після запуску Borland C++ Builder ми побачимо на екрані порожню форму. Змінимо її властивість **`Caption`** й уведемо нову назву форми: «**`Малювання`**» (див. мал.2.). Потім, використовуючи мишу, встановимо розміри вікна **`Height`** й **`Width`**. Для точного задання розміру можна скористатися відповідними властивостями в інспекторі об'єктів й увести значення в пікселях.

Тепер ознайомимося з компонентами. Їх, як деталі дитячого конструктора, можна додавати на форму. Знаходимо потрібні нам компоненти **`Label`** (напис) і **`Button`** (кнопка) на вкладці **`Standard`** (стандартні компоненти):



Клацнемо мишею на компоненті **`Label`**, а потім – на формі, де ми його хочемо розмістити: у місці клацання з'явиться компонент з назвою **`Label1`**. Таким самим способом додамо на

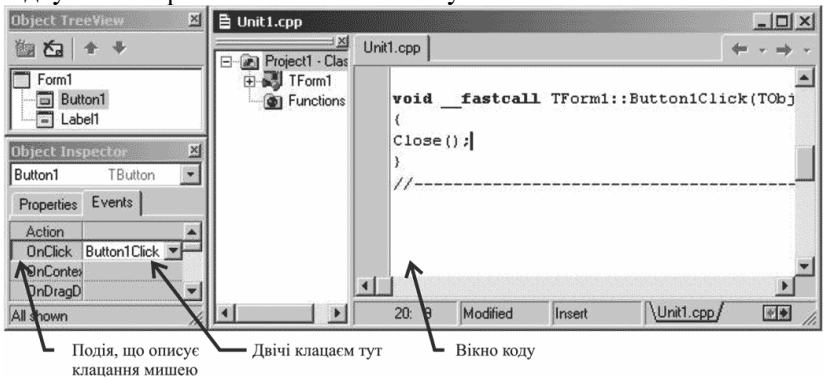
форму кнопку (компонент **`Button`**). У результаті ми одержимо форму із двома компонентами: **`Label1`** та **`Button1`**.

Подвійним клацанням на потрібному компоненті ми спричинимо його появу в центрі форми.

Настроїмо властивості **`Caption`** для напису й кнопки, як вимагає наше завдання. Для компонента **`Label1`** також настроїмо параметри шрифту, як зазначено на мал. 4. Залишається гарно розташувати створені нами об'єкти на формі за допомогою миші. Запустивши програму на виконання ми побачимо вікно із кнопкою й написом. Але спроба натиснути на кнопку не призведе до бажаного результату – закриття вікна. Закриємо вікно за допомогою кнопки з хрестиком у верхньому правому куті.

Написання коду

Для повноцінної роботи програми необхідно вказати, що має відбуватися при натисканні на кнопку.



Для цього виділимо кнопку на формі й у вікні інспектора об'єктів перейдемо на вкладку **`Events`** (події). Тепер лівий стовпець інспектора об'єктів містить всі події, на які може реагувати даний об'єкт, тобто наша кнопка. Проти кожної події може бути зазначена процедура – обробник даної події. Поки що у правому стовпці порожньо. Наша кнопка повинна реагувати на клацання мишею. Відповідна подія називається **`OnClick`** («при клацанні мишею»). Двічі клацнемо мишею навпроти цієї події. При цьому C++ Builder автоматично активізує вікно коду й додасть у нього обробник обраної події, оформлений у вигляді функції.

Функція починається із ключового слова **`void`**, після якого йдуть ім'я функції та її параметри. Текст функції вміщують у фігурні дужки. Нам залишається тільки записати потрібні оператори. Закриття вікна відбувається за допомогою оператора **`Close();`**, тому впишемо його у поки що порожній обробник події.

Тепер можна запустити програму (натиснемо F9) і потім закрити її, клацнувши по створеній нами кнопці.

Питання для самоконтролю:

1. *Назвіть вікна системи програмування **C++ Builder**.*
2. *Що таке «об'єкти»?*
3. *Що являють собою властивості об'єкта?*
4. *Для чого використовують методи?*
5. *Що таке «подія»?*
6. *Що таке форма? Назвіть основні властивості форми.*
7. *Для чого призначене діалогове вікно **Object Inspector**?*
8. *Що означає командна кнопка із трьома крапками біля значення властивості у вікні **Object Inspector**?*
9. *Що означає знак «+» перед назвою властивості у вікні **Object Inspector**?*
10. *Назвіть відомі вам компоненти. Як компоненти встановити на форму?*
11. *Для чого використовують вікно коду?*

Вправа 9-1.

1) Створіть програму «Вікно для малювання»
2) Використавши інспектор об’єктів, змініть властивості форми: **`Caption`**, **`Width`**, **`Height`**, **`Top`**, **`Left`**, **`Color`**.
3) Змініть шрифт напису за допомогою властивості **`Font`**, спробуйте міняти властивості **`BorderStyle`** та **`BorderIcons`**. Як змінюється робота програми?

Збережіть програми, створивши у власній папці нову папку *wpr9-1*.

9.2. Графіка у середовищі Borland C++ Builder

Полотно та примітиви

Borland C++ Builder дозволяє нам виводити графіку на поверхню форми чи компонента **`Image`**, за допомогою властивості **`Canvas`** (англ. *Canvas* – полотно для малювання).

Малюнок являє собою сукупність графічних примітивів.

Примітивами *називають найпростіші геометричні фігури, такі як: лінія, крапка, прямокутник, дуга й ін.*

Креслення графічних примітивів на поверхні форми здійснюється застосуванням відповідних методів до властивості **`Canvas`** цієї форми (чи компонента **`Image`**).

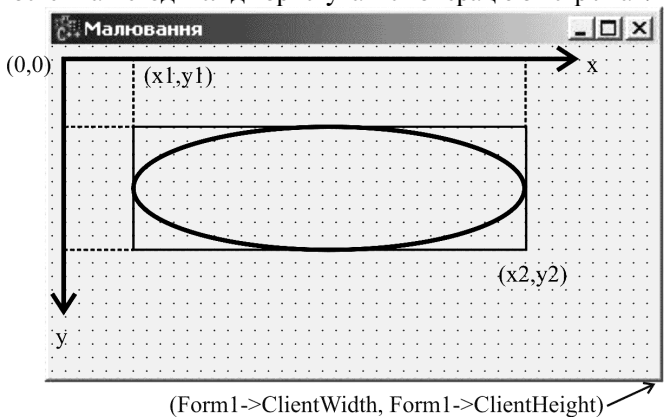
Наприклад, виклик методу **`Ellipse`**:

```
Form1->Canvas->Ellipse(x1,y1,x2,y2);
```

призведе до малювання на поверхні форми (мал.1) еліпса, в якому **`x1`**, **`y1`**, **`x2`**, **`y2`** – координати двох протилежних вершин прямокутника, у який вписано еліпс.

Зверніть увагу на спосіб доступу до властивостей об'єктів із програми. У даному випадку для цього використовують операцію, що позначається «стрілкою», яка складається із двох символів: «мінус» й «більше»: **`->`**. Пробіл між ними неприпустимий.

Для доступу до методів **об'єктів `cin`**, **`cout`**, **`fin`**, **`fout`** затосовувалась операція «крапка»: **`cin.getline()`**, **`fin.close()`** тощо. **`Form1`** являє собою **вказівник на об’єкт**, тому для звертання до його властивостей та методів слід користуватися операцією «стрілка».



Методи виведення графічних примітивів розглядають властивість **`Canvas`**, як «полотно», на якому вони можуть малювати. Полотно складається з окремих крапок – **пікселів**. Положення пікселя на поверхні полотна характеризується горизонтальною (x) і вертикальною (y) координатами. Координатна система має вигляд, наведений на малюнку. Лівий верхній піксель поверхні форми має координати (0, 0), правий нижній – (**`ClientWidth`**, **`ClientHeight`**).

Зображення, отримане на поверхні форми, може бути зіпсоване у результаті перекриття вікна програми іншим вікном. Щоб

виправити це, операційна система Windows інформує програму про необхідність перемальовування вікна, посилаючи їй відповідне повідомлення. У результаті цього виникає подія **OnPaint**.

Подія **OnPaint** виникає й у момент запуску програми, коли вікно з'являється на екрані вперше. Таким чином, інструкції, що забезпечують виведення графіки на поверхню форми, треба помістити у процедуру обробки події **OnPaint**.

Для цього у вікні ***Object Inspector*** відкриємо закладку ***Events*** (події) та двічі клацнемо мишею навпроти події **OnPaint**. При цьому зявиться процедура-обробник даної події: **Tform1::FormPaint**.

У вікно *Unit1.cpp*, яке з'явиться автоматично, будемо уводити програмний код. Але спочатку ознайомимося з основними об'єктами та методами для малювання.

Олівець і пензель

Вигляд графічного елемента визначають властивості **Pen** (олівець) і **Brush** (пензель) поверхні **Canvas**, на якій малює метод.

Олівець і пензель, будучи властивостями об'єкта **Canvas**, у свою чергу являють собою об'єкти **Pen** та **Brush**. Властивості об'єкта **Pen** задають колір, товщину й тип лінії або межі геометричної фігури. Властивості об'єкта **Brush** задають колір та спосіб зафарбовування області всередині кола, прямокутника, сектора або замкнутого контуру.

Властивості об'єкта Pen (олівець)

Color – задає колір лінії.

Width – задає товщину лінії.

Style – задає вид лінії (**psSolid** – суцільна; **psDash** – пунктирна, довгі штрихи; **psDot** – пунктирна, короткі штрихи; **psDashDot** – пунктирна, чергування довгого й короткого штрихів; **psDashDotDot** – пунктирна, чергування одного довгого й двох коротких штрихів; **psClear** – лінія не відображається (використовується, якщо не треба зображувати межу області – наприклад, прямокутника).

Властивості об'єкта Brush (пензель)

Color – задає колір зафарбовування замкнутої області.

Style – задає стиль заповнення області: **bsSolid** — суцільне заливання; **bsClear** – заповнення відсутнє; штрихування:

bsHorizontal – горизонтальні лінії; **bsVertical** – вертикальні лінії; **bsFDiagonal** – діагональне з нахилом ліній вперед; **bsBDiagonal** – діагональне з нахилом ліній назад; **bsCrossv** – у клітинку; **bsDiagCross** – діагональна клітинка.

Лінія

Метод **LineTo(x, y)** малює лінію з тієї крапки, у якій у цей момент перебуває олівець (ця крапка називається "поточною"), до крапки з координатами (x,y).

Наприклад:

```
Canvas->LineTo(300,250);
```

малює лінію в крапку з координатами (300, 250), після чого поточною стає крапка з координатами (300, 250).

Метод **MoveTo(x,y)** робить поточною крапку з координатами (x,y).

Наприклад, результатом виконання такої послідовності команд:

```
Canvas->MoveTo(5,25);
Canvas->LineTo(30,25);
```

буде горизонтальна лінія із крапки (5, 25) у крапку (30, 25).

Завдяки автоматичній зміні поточної крапки, можна легко намалювати ламану лінію. Наприклад, ця програма малює букву И:

```
Canvas->MoveTo(10, 10);
Canvas->LineTo(10, 200);
Canvas->LineTo(100, 10);
Canvas->LineTo(100, 200);
```

Прямокутник

Метод **Rectangle (xl, yl, x2, y2)** малює зафарбований прямокутник, в якому: xl, yl – координати лівого верхнього, x2, y2 – правого нижнього кутів прямокутника. Наприклад:

```
Canvas->Rectangle(5,5,200,200);
```

малює квадрат, лівий верхній кут якого знаходиться у крапці (5, 5), а правий нижній – у крапці (200, 200).

Кольори межі й внутрішньої області прямокутника можуть бути різними. Колір, вид і ширина лінії контуру прямокутника визначаються значенням властивості **Pen**, а колір та стиль заливання області всередині прямокутника – значенням властивості **Brush** тієї поверхні, на якій малюється прямокутник. Намалюємо державний прапор України:

```
// колір пензля - синій
    Canvas->Brush->Color = clBlue;
    Canvas->Rectangle (10,10,180,80);
// колір пензля - жовтий
    Canvas->Brush->Color = clYellow;
    Canvas->Rectangle (10,80,180,160);
```

Як бачите, ми використовуємо операцію присвоювання «**=**», щоб задати колір (**Color**) об'єкту **Brush**.

Замість координат кутів, методу **Rectangle** можна передати один параметр – структуру типу **TRect,** яка буде визначати координати кутів:

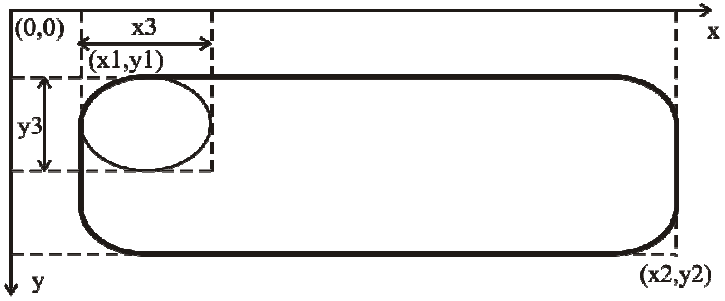
```
    TRect rct;                  // прямокутна область
    rct.Top = 5;
    rct.Left = 5;
    rct.Bottom = 200;
    rct.Right = 200;
    Canvas->Rectangle(rct);
```

Метод **FillRect** малює зафарбований прямокутник, використовуючи тільки пензель (**Brush**).

Метод **FrameRect** малює контур прямокутника, використовуючи тільки олівець (**Pen**).

Кожен з цих методів має один параметр – структуру типу **TRect**, значення якої можна задати за допомогою функції **Rect**:

```
    TRect rct;
    rct = Rect(5,5,200,200);
    Canvas->Brush->Color = clBlue;
    Canvas->FillRect(rct);
```



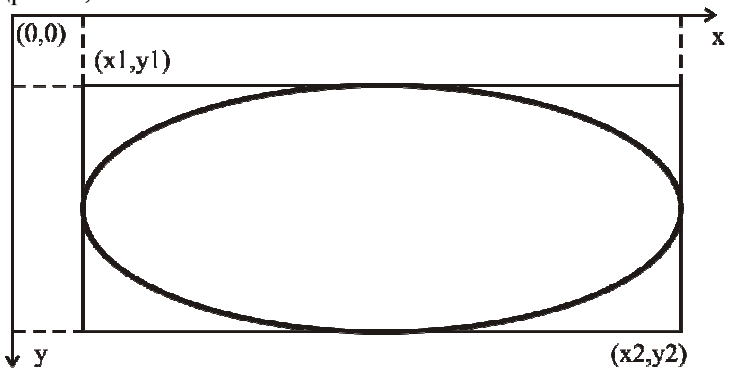
Метод **RoundRec(xl,yl,x2,y2,x3,y3)** намалює прямокутник з округленими кутами (див. мал.).

Параметри **xl**, **y1**, **x2**, **y2** визначають положення кутів прямокутника, а параметри **x3** й **y3** – розмір еліпса, одна чверть якого використовується для креслення округленого кута (див. мал.).

Коло й еліпс

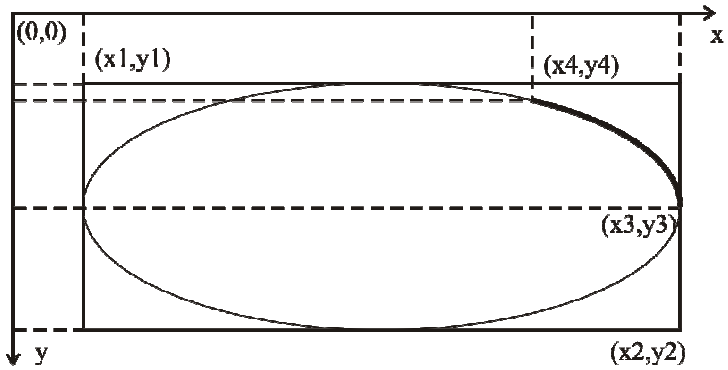
Метод **Ellipse(x1,y1,x2,y2)** малює еліпс або коло.

Параметри **xl**, **yl**, **x2**, **y2** визначають координати вершин прямокутника, в який вписано еліпс або, якщо прямокутник є квадратом, – коло.



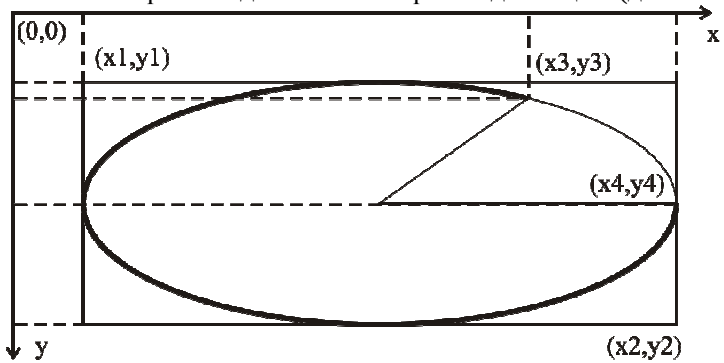
Дуга

Метод **Arc(x1, y1, x2, y2, x3, y3, x4, y4)** малює дугу.



Параметри **xl**, **y1**, **x2**, **y2** визначають еліпс (коло), частиною якого є дуга. Параметри **x3** й **y3** задають початкову, а **x4** й **y4** –

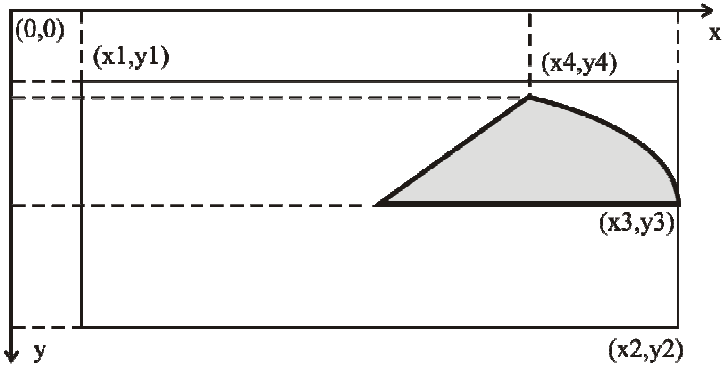
кінцеву крапку дуги. Початкова (кінцева) крапка дуги – це крапка перетину межі еліпса й прямої, проведеної із центра еліпса у крапку з координатами **x3** й **y3** (**x4**, **y4**). Метод **Arc** креслить дугу проти годинникової стрілки від початкової крапки до кінцевї (див. мал.).



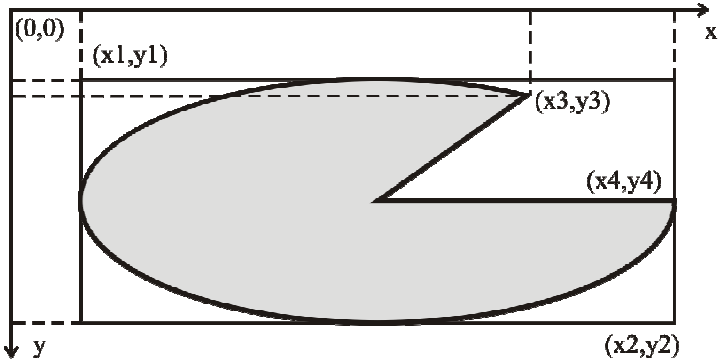
Колір, товщина й стиль лінії, якою малюється дуга, визначаються властивістю **Pen**.

Сектор

Метод **Pie (x1, y1, x2, y2, x3, y3, x4, y4)** малює еліптичний чи круговий сектор.



Параметри **x1**, **yl**, **x2**, **y2** визначають еліпс (круг), частиною якого є сектор; **x3**, **y3**, **x4** й **y4** – прямі – межі сектора. Початкова крапка меж збігається із центром еліпса. Сектор вимальовується проти годинникової стрілки від прямої, заданої крапкою з координатами (**x3**, **y3**) та центром фігури, до прямої, заданої крапкою з координатами (**x4**, **y4**) та центром фігури (див. мал.).



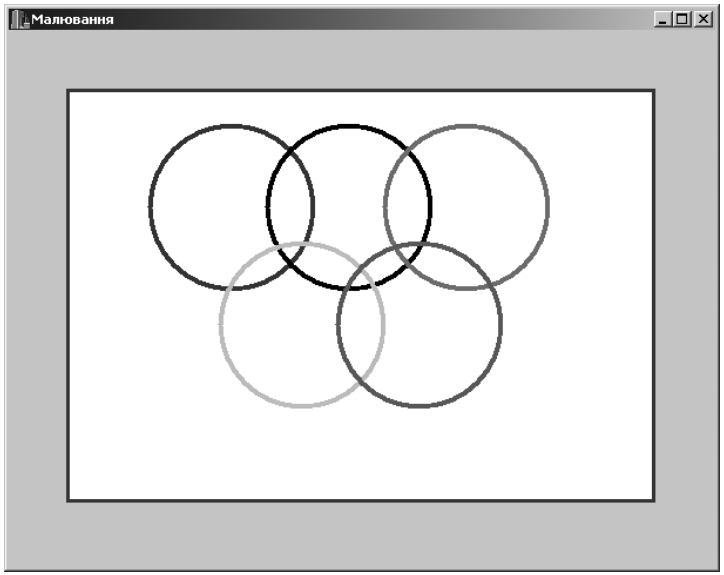
Приклади малювання

Приклад 1. *Нижче наведена процедура обробки події* ***onPaint****, що малює на поверхні форми олімпійський прапор.*

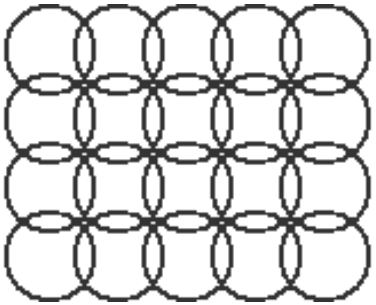
Уведемо розміри об'єкту **Form1** у вікні інспектору обєктів: **ClientWidth** – 600, **ClientHeight** – 450. Властивості **Color** надамо значення **clCream**, а властивості **Caption** дамо ім'я: «**Малювання**». Не забудемо на вкладці ***Events*** навпроти події **OnPaint** двічі клацнути мишею. При цьому з'явиться вікно, куди уведемо такий код:

```
void _fastcall TForm1::FormPaint(TObject*Sender)
{                                          //Програма 9.1
//полотнище прапора
      Canvas->Pen->Width = 3;
      Canvas->Pen->Color = clPurple;
      Canvas->Brush->Color = clWhite;
      Canvas->Rectangle (50,50,550,400);
//ширина контурів кілець
      Canvas->Pen->Width = 4;
//щоб коло, намальоване
//методом Ellipse, не було зафарбоване
      Canvas->Brush->Style = bsClear;
//малюємо кільця
      Canvas->Pen->Color = clBlue;
      Canvas->Ellipse(120, 80,260, 220);
```

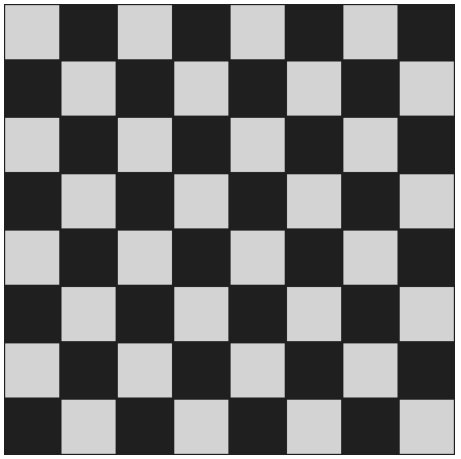
```
    Canvas->Pen->Color = clBlack;
    Canvas->Ellipse(220,80,360,220);
    Canvas->Pen->Color = clRed;
    Canvas->Ellipse(320, 80,460,220);
    Canvas->Pen->Color = clYellow;
    Canvas->Ellipse(180,180,320,320);
    Canvas->Pen->Color = clGreen;
    Canvas->Ellipse(280,180,420, 320);
}
```



Приклад 2. *Написати програму, що виводить на екран візерунок:*




```
void _fastcall TForm1::FormPaint(TObject*Sender)
{                                          //Програма 9.2
//ширина контурів кілець
      Canvas->Pen->Width = 2;
//колір олівця - синій
      Canvas->Pen->Color = clBlue;
//коло не зафарбоване
      Canvas->Brush->Style = bsClear;
//малювання
int x=100,y=100;          //координати центра кола
int r=20;                              //радіус кола
int d=30;               //відстань між центрами кіл
  for( int i=0;i<4;i++)              //чотири рядки
  {
   x=100;
     for( int j=0;j<5;j++)   //по п'ять у кожному
     {
       Canvas->Ellipse(x-r, y-r,x+r, y+r);
       x+=d;
     }
   y+=d;
  }
}
```



***Приклад 3**. Написати програму, що виводить на екран зображення шахівниці:*

```
void _fastcall TForm1::FormPaint(TObject*Sender)
{                                        //Програма 9.3
Canvas->Pen->Width = 1;
Canvas->Pen->Color = clBlack;
int x0=100;          //координати верхнього лівого
int y0=100;                          //кута шахівниці
int x,y;             //координати верхнього лівого
                          //кута поточної клітинки
int w=50;                           //розмір клітинки
y=y0;
Canvas->Brush->Style = bsSolid;
  for( int i=0;i<8;i++)                  //8 рядків
  {
     x=x0;
    for( int j=0;j<8;j++)    //8 клітинок у рядку
    {
     //якщо сума номера рядка і номера стовпця,
     //на перетині яких знаходиться клітинка,
     //парна, то клітинка - чорна, інакше -
     //жовта
    if((i+j)%2)//перевірка парності
       Canvas->Brush->Color = clBlack;
    else
       Canvas->Brush->Color = clYellow;
    Canvas->Rectangle(x, y,x+w, y+w);
    x+=w;
    }
   y+=w;
  }
}
```

Питання для самоконтролю:

1. *Що таке «примітив»?*
2. *Який вигляд має координатна система форми?*
3. *Для чого використовують «олівець» і «пензель»?*
4. *Назвіть декілька відомих вам методів малювання примітивів?*
5. *За якими параметрами будується прямокутник зі скругленими кутами?*

6. *Які параметри потрібно передати при виклику методу **Ellipse**?*
7. *Який зміст мають параметри у такій команді: **Canvas->Pie (100, 80, 300, 220, 220, 80, 100, 300);**?*

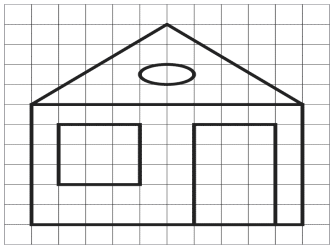
Вправа 9-2.

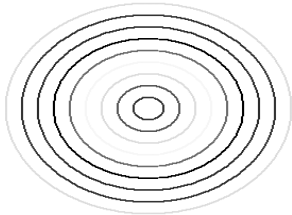
1) Випробуйте програму 9.1. для малювання олімпійських кілець.
2) Випробуйте програму 9.2. Уведіть радіус кола: r=50, а відстань між центрами кіл: d=50.
3) Випробуйте програму 9.3. «шахівниця». Змініть розмір клітинки: w=25.

Збережіть програми, створивши у власній папці нову папку *wpr9-2*.

9.3. Практична робота № 13 «Малювання примітивів»

1) Написати програму, при виконанні якої на екрані малюється хатинка.





2) Написати програму для малювання мішені з різнокольоровими колами.

Вибір кольору можна реалізувати, наприклад, так:

```
…
int c;
…
switch (c)
   {
     case 1: Canvas->Pen->Color = clBlue;
     break;
     case 2: Canvas->Pen->Color = clGreen;
     break;
```

```
      case 4: Canvas->Pen->Color = clYellow;
      break;
      case 5: Canvas->Pen->Color = clRed;
      break;
      case 6: Canvas->Pen->Color = clBlack;
      break;
      case 7: Canvas->Pen->Color = clMaroon;
      break;
      case 8: Canvas->Pen->Color = clPurple;
      break;
      default: Canvas->Pen->Color = clAqua;
   }
…
```

Якщо встановити курсор у вікні уведення коду і натиснути комбінацію клавіш <Ctrl>+<J>, то з'явиться список шаблонів коду. Вибравши, наприклад, *switche*, ми побачимо у вікні коду заготовку:

```
switch ()
   {
     case : ;
     break;
     case : ;
     break;
     default: ;
   }
```

Використовуйте шаблони для уведення коду програми.

3) Написати програму для малювання на екрані різнокольорової ламаної лінії з 1000 ланок з випадковими координатами вершин.

Збережіть програми, створивши у власній папці нову папку *pr13*.

9.4. Виведення тексту. Малювання крапками

Текст у візуальній системі програмування C++Builder для рядкових даних можна використати змінні типу **AnsiString**.

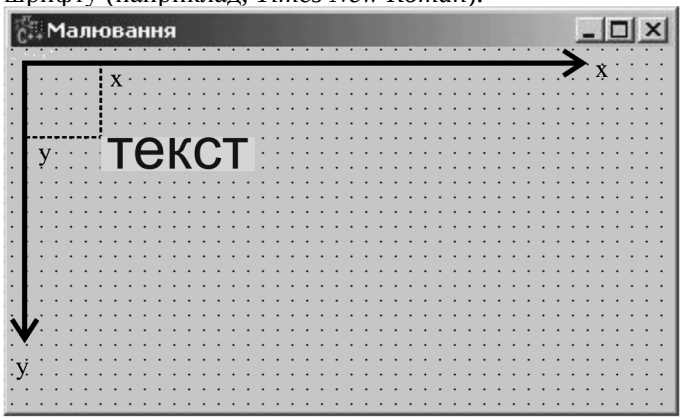
Метод **TextOutA(x, y, Tekst)** виводить текст на поверхню графічного об'єкта.

Параметр **Tekst** задає текст для виведенння. Параметри **x** і **y** визначають координати крапки графічної поверхні, від якої виконується виведення тексту (див. мал.).

Шрифт, що використовується для виведення тексту, визначається властивістю **Font** відповідного об'єкта **Canvas**. Властивість **Font** являє собою об'єкт типу **TFont**. Нижче перераховані властивості об'єкта **TFont**, що визначають характеристики шрифту, використовуваного методом **Textout** для виведення тексту.

Властивості об'єкта TFont

Name – використовуваний шрифт. Як значення слід вказати назву шрифту (наприклад, *Times New Roman*).



Size – розмір шрифту в пунктах (points). Пункт являє собою одиницю виміру розміру шрифту, яка становить 1/72 дюйма (приблизно 0,35 мм).

Style – стиль накреслення символів. Стиль може бути: нормальним, напівжирним, курсивним, підкресленим, перекресленим і задається за допомогою таких констант:

- **fsBold** - **напівжирний**;
- **fsItalic** - *курсив*;
- **fsUnderline** - підкреслений;
- **fsStrikeOut** - ~~перекреслений~~.

Властивість **Style** дозволяє комбінувати необхідні стилі. Наприклад, програмний код, що встановлює стиль «напівжирний курсив», виглядатиме так:

```
Canvas->Font->Style=TFontStyles()<<fsBold
<<fsItalic;
```

Color – колір символів. Як значення можна використовувати константу типу **TColor**.

При виведенні тексту використовують також методи **TextWidth** й **TextHeight**, значеннями яких є, відповідно, ширина й висота області виведення тексту, які залежать від характеристик використовуваного шрифту. Цим методам, як параметр, передається рядок, що передбачається вивести на поверхню методом **TextOutA**. Знаючи ширину і висоту текстового блоку, можна більш вдало розмістити його на формі, обчисливши координати лівого верхнього кута.

Приклад 1. *Виведення тексту на поверхню форми.*

Процедура обробки події **OnPaint** зафарбовує верхню половину вікна білим, нижню – бірюзовим кольорами, потім у центрі вікна, вздовж межі зафарбованих областей, виводить текст.


```
void _fastcall TForml::FormPaint(TObject*Sender)
{                                        //Програма 9.4
AnsiString tekst="Малювання у вікні";
TRect aRect;
// крапка, від якої буде виведений текст
int x,y;
//верхню половину вікна зафарбовуємо білим
aRect=Rect(0,0,ClientWidth,ClientHeight/2);
Canvas->Brush->Color=clWhite;
Canvas->FillRect(aRect);
//нижню половину вікна зафарбовуємо бірюзовим
aRect=Rect(0,ClientHeight/2,ClientWidth,
ClientHeight);
```



```
Canvas->Brush->Color=clAqua;
Canvas->FillRect(aRect);
//Назва шрифту та його розмір і стиль
Canvas->Font->Name="Times New Roman";
Canvas->Font->Size=24;
//текст розмістимо в центрі вікна
x = (ClientWidth-Canvas->TextWidth(tekst))/2;
y = ClientHeight/2-Canvas->TextHeight(tekst)/2;
//область виведення тексту не зафарбовувати.
Canvas->Brush->Style = bsClear;
Canvas->Font->Color = clBlack;
// вивести текст
Canvas->TextOutA(x, y, tekst);
}
```

Крапка

Як ви вже знаєте, поверхні, на яку програма може здійснювати виведення графіки, відповідає об'єкт **Canvas**. Властивість **Pixels**, що являє собою двовимірний масив типу **TColor**, містить інформацію про кольори кожної крапки графічної поверхні. Використовуючи властивість **Pixels**, можна задати кольори будь-якої крапки графічної поверхні, тобто "намалювати" крапку.

Наприклад:

```
Canvas->Pixels[100][200] = clBlue;
```

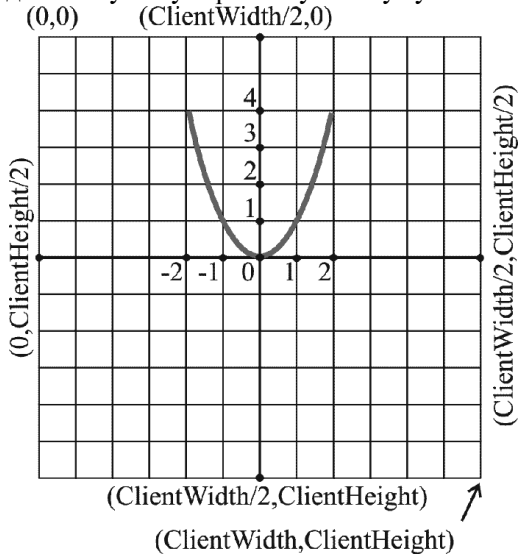
забарвлює крапку поверхні форми з координатами (100; 200) в синій колір.

Розмірність масиву **Pixels** визначається реальним розміром графічної поверхні. Розмір графічної поверхні форми (робочої області, що також називають «клієнтською») визначають властивості **ClientWidth** й **ClientHeight** (**Width** і **Height** – для графічної поверхні компонента **Image**). Лівій верхній крапці робочої області форми відповідає елемент **Pixels[0][0]**, а правої нижньої – **Pixels[ClientWidth-1][ClientHeight-1]**.

Приклад 2. *Намалювати графік функції $y=x^2$ (параболу).*

Для побудови графіка використовується вся доступна область форми. Причому, якщо під час роботи програми користувач змінить розмір вікна, то графік буде виведений все одно посередині з урахуванням реальних розмірів вікна.

Умовно розіб'ємо нашу форму на 12 рядків та 12 стовпців (див. мал.). Графік будуватимемо, як це звично у математиці, для значень *x* від -2 до 2 з кроком 0,01 (крок підберемо експериментально, від нього залежить, бачитимемо ми окремі крапки на екрані чи вони зіллються в одну лінію). Залишається звести обчислені координати до екранної системи координат з нулем у верхньому лівому куті.



Координати крапки на екрані можна знайти за такими формулами:

```
Y= -(x*x)*ClientHeight/12+ClientHeight/2;
X= -x*ClientWidth/12+ClientWidth/2;
```

Тепер можна скласти програму для побудови графіка:

```
void _fastcall TForm1::FormPaint(TObject*Sender)
{                                        //Програма 9.5
float x,y,Y,X;
//побудова осей
Canvas->MoveTo(0,ClientHeight/2);
Canvas->LineTo(ClientWidth,ClientHeight/2);
Canvas->MoveTo(ClientWidth/2,0);
Canvas->LineTo(ClientWidth/2,ClientHeight);
//побудова графіка
for( x=-2;x<=2;x+=0.01)
{
```



```
      Y= -x*x*ClientHeight/12+ClientHeight/2;
      X= -x*ClientWidth/12+ClientWidth/2;
      Canvas->Pixels[X][Y]=clRed;
}
}
```

Питання для самоконтролю:

1. *Як вивести текст на поверхню графічного об'єкта?*
2. *Назвіть основні властивості об'єкта* ***TFont.***
3. *Поясніть, завдяки чому текст при виконанні програми 9.4 виводиться посередині вікна.*
4. *Як здійснюється малювання крапками?*
5. *Поясніть, як перетвоюють координати формули, використані у програмі.*

Вправа 9-4.

1) Випробуйте програму 9.4. Уведіть свій текст. Змініть шрифт на *Arial* (підкреслений). Встановіть нові розміри шрифту.
2) Змініть програму 9.4 так, щоб текст був виведений у нижньому правому куті вікна.
3) Випробуйте програму 9.5.
4) Змініть програму так, щоб отримати зображення параболи $y = x^3$.

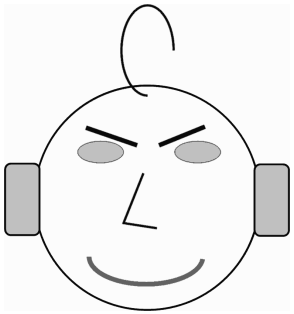
Збережіть програми, створивши у власній папці нову папку *wpr9-4*.

9.5. Практична робота № 14 «Малювання графіків функцій»

1) Написати програму, яка малює на екрані оцифровану координатну сітку.
2) Використавши програму з завдання 1, написати програму для малювання графіка функції *y=sin(x)*. (Пригадайте, що M_PI=3,14159).
3) Намалювати графік функції *y= a·sin(n·x)*, де a=0,5; a=2; n=0,5; n=3.

Збережіть програми, створивши у власній папці нову папку *pr14*.

9.6. Тематична атестація «Побудова графічних зображень»



1) Намалювати кумедне обличчя жовтого кольору (див. мал.).

> **Порада**. Для більш швидкого визначення координат примітивів перенесіть малюнок на аркуш в клітинку і зобразіть на ньому координатні осі у потрібному масштабі.

2) Скласти програму, яка виводитиме на екран монітора графіки траєкторій тіла, кинутого під кутом до горизонту, для кутів від 10° до 70° через кожні 20° з початковими швидкостями 20 м/с та 30 м/с, скориставшись для більшої наочності різними кольорами. Опором повітря знехтувати.

> **Зауваження**. Для побудови графіка необхідно скористатися формулою:
>
> $$y = \frac{v_{0y}}{v_{0x}} x - \frac{g}{2v_{0x}^2} x^2 \text{, де } v_x = v_{0x} = v_0 \cos\alpha, \quad v_y = v_{0y} = v_0 \sin\alpha$$

3) У верхній частині вікна вивести текст: «Траєкторії тіла».

> Збережіть програми, створивши у власній папці нову папку *ta7*.

Питання для самоконтролю:

1. *Назвіть 5 вікон середовища візуальної розробки **Borland C++ Builder**. Яке призначення кожного з них?*
2. *Що таке об'єкт? З чого він складається?*
3. *Назвіть основні властивості форми? У якому вікні їх можна змінити?*
4. *Що таке «графічний примітив»?*
5. *Який вигляд має координатна система при роботі з графікою?*
6. *Назвіть основні властивості об'єктів **Pen** та **Brush**?*
7. *Як намалювати прямокутник, еліпс та лінію? Які методи при цьому використовують?*
8. *Який метод виводить текст на поверхню графічного об'єкта?*
9. *Назвіть основні властивості об'єкта **TFont**?*
10. *Як намалювати крапку на поверхні графічного об'єкта?*

ДЛЯ ЗАМІТОК

ВИДАВНИЦТВО «АСПЕКТ» ПРОПОНУЄ:

Серію посібників «Для початківця»:

- книга 4 «Основи комп'ютерної грамотності (Windows'XP, Word'XP, Paint), Internet», Шестопалов Є.А., 2006, – 176 стор.
- книга 5 «Word'97&2000 для початківця», Шестопалов Є.А., 2005, – 112 стор.
- книга 6 «Excel'2000&XP для початківця», Шестопалов Є.А., 2005, – 112 стор.
- книга 7 «Windows'XP для початківця», Шестопалов Є.А., Моісеєва О.В., 2006, – 160 стор.
- книга 8 «Internet для початківця», Шестопалов Є.А., 2005, – 112 стор.
- книга 9 «Access'2000 для початківця», Чаповська Р.Б., 2005, – 96 стор.
- книга 10 «Power Point для початківця», Сальнікова І.І., 2005, – 112 стор.

Серію посібників для 11-річних навчальних закладів:

- «Інформатика. Короткий курс. 10 клас», Шестопалов Є.А., 2006, – 176 стор.
- «Інформатика. Короткий курс. 11 клас», Сальнікова І.І., Шестопалов Є.А., 2006, – 208 стор.
- «Інформатика. Базовий курс. 10 клас», Шестопалов Є.А., 2006, – 160 стор.
- «Інформатика. Основи алгоритмізації та програмування. 10 клас», Караванова Т.П., 2006, – 192 стор.
- «Інформатика. Збірник вправ та задач з алгоритмізації та програмування. 10-11 клас.», Караванова Т.П., 2007, – 152 стор.
- «Інформатика. Базовий курс. 11 клас», Шестопалов Є.А., Сальнікова І.І., 2007, – 336 стор.
- «Інформатика. Практичні та тематичні роботи і проекти. 10-11 клас», Сальнікова І.І., Шестопалов Є.А., 2007, – 160 стор.

Серію посібників для 12-річних середніх навчальних закладів:

- «Інформатика. Вступ до програмування мовою ЛОГО. 5 клас», Пахомова Г.В., 2007, – 136 стор.
- «Інформатика. Базовий курс. 7 клас», Шестопалов Є.А., 2006, – 176 стор.
- «Інформатика. Баз. курс. 8 кл.», Шестопалов Є.А., Сальнікова І.І., 2006, – 208 с.
- «Інформатика. Баз. курс. 9 кл.», Шестопалов Є.А., Пилипчук О.П., 2006, – 176 с.
- «Інформатика. Visual Basic. 9 клас.», Бондаренко О.О., 2007, – 200 стор.
- «Інформатика. Turbo Pascal. Спецкурс», Бондаренко О.О., 2007, – 272 стор.
- «Інформатика. Мова програмування С++. Спецкурс», Лєхан С.А., 2007, – 160 с.

Серию учебных пособий на русском языке:

- «Информатика. Базовый курс. В 3-х частях, часть 1», Шестопалов Е.А., 2006, – 144 стр.
- «Информатика. Базовый курс. В 3-х частях, часть 2», Шестопалов Е.А., 2006, – 160 стр.
- «Информатика. Базовый курс. В 3-х частях, часть 3», Сальникова И.И., Шестопалов Е.А., 2006, – 168 стр.

Компакт-диск:

- Алго. ЛогоМиры. Тренажери. Контрольно-діагностична система Test-W2 легко і просто ***працює з формулами, графікою і таблицями***. Банк тестів. Календарні плани для 7-11 класів. Друкована інструкція з експлуатації тощо.

Для замовлення книг звертайтесь за адресою:

Шестопалов Євген Анатолійович, вул. Тургенєва, буд. 31,
м. Шепетівка, Хмельницької обл., 30400
дом. тел. 8-03840-473-07, моб. тел. 8-066-283-66-18
E-mail: aspekt@sh.km.ua,

Ознайомитися з посібниками і зробити замовлення можна також з мого сайту www.aspekt-edu.kiev.ua

ДО КНИГ можна замовити ***ОДИН*** компакт-диск.

Лєхан С.А.

Інформатика

Мова програмування С++
Спецкурс

Навчальний посібник

Підписано до друку 05.07.07
Формат 60x84/16. Папір офсетний.
Гарнітура Times. Друк офсетний.
Ум. друк. аркуш 10.0
Зам. Наклад 1000.

Видавець Шестопалов Є.А.,
Свідоцтво про внесення до Державного реєстру
суб'єкта видавничої справи ДК № 2170 від 26.04.2005 р.
вул. Тургенєва, буд. 31, м. Шепетівка, Хмельницька обл., 30400,
Тел: (03840)-4-73-07, E-mail: aspekt@sh.km.ua

Шепетівська міжрайонна друкарня.
30400, м. Шепетівка, Старокостянтинівське шосе, 11
Свідоцтво ХЦ № 008 від 9.10.2000 р.
тел. (03840) 5-15-30