ФАКУЛЬТЕТ АВТОМАТИКИ, ТЕЛЕМЕХАНІКИ ТА ЗВ'ЯЗКУ

Кафедра „Обчислювальна техніка та системи управління"

ВИКОРИСТАННЯ ІНТЕГРОВАНОГО СЕРЕДОВИЩА BORLAND C++ ДЛЯ РОЗВ'ЯЗАННЯ ІНЖЕНЕРНО-ТЕХНІЧНИХ ЗАДАЧ

КОНСПЕКТ ЛЕКЦІЙ
з дисципліни

«КОМП'ЮТЕРНА ТЕХНІКА ТА ПРОГРАМУВАННЯ»

Частина 2

Харків 2006

Конспект лекцій розглянуто та рекомендовано до друку на засіданні кафедри “Обчислювальна техніка та системи управління” 28 січня 2003 р., протокол № 5.

Рекомендовано для студентів академії всіх форм і строків навчання та відповідають робочій програмі з курсу «Комп’ютерна техніка та програмування».
.

Укладач

доц. С.Є.Бантюков

Рецензент

проф. Г.І.Загарій

ВИКОРИСТАННЯ ІНТЕГРОВАНОГО
СЕРЕДОВИЩА BORLAND C++ ДЛЯ РОЗВ'ЯЗАННЯ
ІНЖЕНЕРНО-ТЕХНІЧНИХ ЗАДАЧ

КОНСПЕКТ ЛЕКЦІЙ
з дисципліни

«КОМП'ЮТЕРНА ТЕХНІКА ТА ПРОГРАМУВАННЯ»

Частина 2

Відповідальний за випуск Бантюков С.Є.

Редактор Дмитроченко Л.І.

Підписано до друку 20.03.05 р.

Формат паперу 60x84 1/16 . Папір писальний.

Умовн.-друк.арк. 2,25. Обл.-вид.арк. 2,5.

Замовлення № Тираж 200. Ціна

Видавництво УкрДАЗТу, свідоцтво ДК № 112 від 06.07.2000 р.
Друкарня УкрДАЗТу,
61050, Харків - 50, пл. Фейєрбаха, 7

УКРАЇНСЬКА ДЕРЖАВНА АКАДЕМІЯ
ЗАЛІЗНИЧНОГО ТРАНСПОРТУ

Кафедра обчислювальної техніки і систем управління

ВИКОРИСТАННЯ ІНТЕГРОВАНОГО СЕРЕДОВИЩА BORLAND C++ ДЛЯ РІШЕННЯ ІНЖЕНЕРНО-ТЕХНІЧНИХ ЗАДАЧ

КОНСПЕКТ ЛЕКЦІЙ
з дисципліни
«Комп’ютерна техніка та програмування»

Частина 2

Харків 2006

Методичні вказівки розглянуті та рекомендовані до друку на засіданні кафедри «Обчислювальна техніка і системи управління» 28 січня 2003 р., протокол № 5.

Призначені для студентів академії всіх форм і строків навчання та відповідають робочій програмі з курсу «Комп’ютерна техніка та програмування».

Склав:
доц. С.Є. Бантюков

Рецензент
проф. Г.І. Загарій

ЗМІСТ

1 ПОКАЖЧИКИ

Для написання ефективних програм мовою C++ дуже важливо розуміти роботу і вміти використовувати покажчики. Це необхідно по трьом причинам: по-перше, покажчики дозволяють змінювати аргументи функцій, що знаходяться у викликах; по-друге, покажчики можна використовувати для підтримки динамічного розміщення пам'яті; по-третє, покажчиками можна заміняти масиви з метою підвищення ефективності роботи програми.

Однак, не дивлячись на те, що покажчики є одним з наймогутніших засобів мови C++, вони, у той же час, небезпечний інструмент. Наприклад, використання неініціалізованих покажчиків може викликати аварійне завершення роботи системи. Крім того, некоректний запис покажчиків є джерелом помилок, які важко знайти.

1.1 Адреси і покажчики. Покажчик – це змінна, що вміщує адресу комірки пам'яті іншої змінної. Коли одна змінна містить адресу іншої змінної, говорять, що перша змінна вказує на другу. Графічно це зображено на рисунку 1.

1.2 Покажчик–змінна. Якщо змінна повинна містити покажчик, то її необхідно оголосити відповідним чином. Загальний формат оголошення покажчик-змінної має вид:

*<тип> *<ім'я_змінної>;*

де *тип* – один з допустимих у мові C++ базових типів;
ім'я_змінної – ім'я покажчик-змінної.

Базовий тип покажчика визначає, на який тип змінної він може вказувати. Наприклад, у наступних операторах оголошено покажчики на символьну і цілі змінні:

```
char *p;
int *temp, *start;
```

Рисунок 1 – Змінна, що вказує на іншу змінну

1.3 Оператори з покажчиками. Є два оператори, що вживаються з покажчиками: * і &. Оператор & є унарним і *повертає адресу комірки пам'яті свого операнда*. (Унарний оператор може застосовуватися тільки до одного операнда). Наприклад,

count_addr = &count;

поміщає в змінну *count_addr* адресу комірки пам'яті змінної *count*. Ця адреса вказує на місце розташування змінної в пам'яті комп'ютера. Варто мати на увазі, що адреса змінної не має нічого спільного зі значенням змінної. Наприклад, припустимо, що змінна *count* розміщається за адресою *2000*. Після виконання вищевказаного оператора змінна *count_addr* буде мати значення *2000*.

Другий оператор, *, є доповненням оператора &. Це також унарний оператор, що повертає *значення змінної, розташованої за даною адресою*. Наприклад, якщо *count_addr* містить адресу комірки пам'яті змінної *count*, тоді

```
val = *count_addr;
```

помістить значення змінної *count* у змінну *val*.

У мові С++ знак множення і знак *«за адресою»* однакові, тому при написанні програми треба не забувати робити між ними розходження.

Оператори & і * мають більш високий пріоритет, чим всі інші арифметичні оператори, за винятком унарного мінуса, з яким за пріоритетом вони рівні.

Приклад програми, у якій використовуються два приведених оператори присвоювання для виведення на екран числа *100*:

```
#include <stdio.h>
main()
{
int *count_addr, count, val;
count = 100;

count_addr = &count;      // присвоєння count_addr адреси
count
val = *count_addr;    // присвоєння val значення, що
                      знаходиться за //адресою, що записана
                      в count_addr
printf("%d",val);       // виведення на екран числа 100
}
```

Як видно, змінній *val* можна присвоїти значення змінної *count* непрямим чином, використовуючи покажчик. У зв'язку з цим може виникнути питання, як С++ визначає число байтів, які необхідно копіювати в *val* з адреси, зазначеної в змінній *count_addr*? Відповідь така: компілятор припускає, що покажчик

указує на змінну, тип якої визначається базовим типом покажчика.

У нашому випадку, оскільки *count_addr* покажчик цілого типу, C++ копіює в змінну *val* два байти інформації з адреси, на який указує *count_addr*. Якби *count_addr* був покажчиком подвійної точності, компілятор скопіював би вісім байтів. (Як уже згадувалося вище, під змінну кожного типу відводиться відповідна кількість пам'яті).

1.4 Вирази з покажчиками. У загальному випадку вирази з покажчиками підпорядковуються тим же правилам, що і будь-які інші вирази в мові C++. Однак існують деякі особливості запису цих виразів.

Оператор присвоювання покажчика. Як і у випадку з будь-якою змінною, покажчик можна записувати в правій частині оператора присвоювання для присвоювання значення покажчика іншій змінній, як показано в прикладі:

```
#include <stdio.h>
main()
{
int x;
int *p1, *p2;

p1 = &x;
p2 = p1;
printf("%p",p2);        // печатка адреси x, а не значення
}
```

У даній програмі на екран виводиться адреса змінної *x* шляхом використання спеціального формату функції *printf()*. Специфікатор формату *%p* визначає, що буде виводитися значення покажчика.

Арифметичні операції з покажчиками. У мові C++ над покажчиками можна виконувати тільки дві арифметичні операції: + і –. Щоб пояснити, що відбувається в діях над покажчиками, припустимо, що *p1*– покажчик на ціле значення з поточною величиною *2000*. Після виконання виразу

p1++;

вміст *p1* буде *2002*, а не *2001*. Тобто при кожному збільшенні покажчик повинен указувати на наступне ціле. Те ж справедливе і для від'ємного збільшення. Наприклад, при виконанні

p1– –;

p1 буде мати значення *1998*.

Таким чином, покажчик після кожного збільшення буде вказувати на наступний елемент базового типу. Це стосується як додатнього, так і від'ємного збільшення. В усіх випадках покажчик буде збільшуватися чи зменшуватися відповідно до кількості байтів, займаних типом даного, на яке він указує. Наприклад, якщо покажчик указує на однобайтові символи, його значення буде змінюватися на одиницю.

У мові С++ дії над покажчиками не обмежуються тільки їх додатнім чи від'ємним збільшенням. До покажчиків можна додавати і віднімати з них цілі величини. Так, у виразі

p1 = p1 + 9;

p1 буде вказувати на дев'ятий елемент базового типу *p1* стосовно поточного.

Покажчики не можуть брати участі ні в яких інших операціях, крім їхнього додавання і вирахування з цілими числами. Не можна множити і ділити покажчики, не можна додавати і віднімати два покажчики. Крім того, до покажчика не можна застосовувати операції додавання і віднімання з типами *float* і *double*.

Операції порівняння покажчиків. До покажчиків можна застосовувати операції порівняння. Наприклад, якщо дано покажчики *p* і *q*, то справедливий вираз:

if(p<q) printf("p указує на комірку пам'яті з більш низькою адресою, ніж q \n");

Звичайне порівняння покажчиків використовується, коли два чи більше покажчики вказують на один об'єкт.

1.5 Покажчики і масиви. Існує близький взаємозв'язок між покажчиками і масивами. Розглянемо наступний фрагмент програми:

```
char str[80], *p1;
char *p1;

p1 = str.
```

У даному фрагменті покажчику *p1* присвоюється адреса першого елемента масиву *str*. У мові С++ ім'я масиву без індексів трактується як адреса початкового елемента. Тобто ім'я масиву є покажчиком на масив. Таким чином, звертання до п'ятого елемента масиву можна записати як

```
str[4]
```

або як

```
*(p1+4).
```

Обидва оператори повернуть п'ятий елемент масиву. Пам’ятайте, що елементи масиву починають нумеруватися з нуля, тому в обох вищевказаних операторах використовується цифра 4.

Мова С++ дає два способи звертання до елемента масиву: за допомогою індексованого імені та за допомогою арифметики з покажчиками. Остання здійснюється швидше, ніж звертання до елемента по індексу. У зв'язку з цим, використання покажчиків для доступу до елементів масиву в мові С++ застосовується досить часто.

Як приклад використання покажчика замість індексованого імені масиву розглянемо наступні дві програми.

```
// Версія з використанням індексованого імені
#include <stdio.h>
main()
{
char str[80];
int i;
```

```
printf("Ввести рядок із заголовних літер");
gets(str);

printf("Тут є наявним рядок з малих літер:");
for(i=0; str[i]; i++) printf("%c",tolower(str[i]));
}

// Версія з використанням покажчика

#include <stdio.h>
main()
{
char str[80], *p;

printf("Ввести рядок із заголовних літер");
gets(str);

printf("Тут є наявним рядок з малих літер:");

p=str;       // одержати адресу str
while(*p) printf("%c",tolower(*p++));
}
```

Версія з індексованим ім'ям виконується повільніше, тому що на індексування масиву йде більше часу, ніж на виконання оператора *.

Однак, не завжди використання покажчиків більш ефективне, чим індексування. Якщо, наприклад, необхідно звертатися до елементів масиву строго один по одному в порядку зростання чи спадання індексів, то використання покажчиків прискорить виконання програми. З іншого боку, якщо необхідно звертатися до елементів масиву в довільному порядку, тоді використання індексованих імен більш переважне, оскільки складні вирази з покажчиками будуть довше зчитуватися, до того ж програма з індексованими іменами виходить більш зрозумілою. Крім того, коли використовуються індексовані масиви, компілятор сам виконує частину роботи.

1.6 Індексований покажчик. У мові С++ допускається індексування покажчиків. Наприклад, усі записи в наступній програмі цілком допустимі. У процесі її виконання на екран будуть виводитися числа від 1 до 5.

```
// Індексований покажчик
#include <stdio.h>
main()
{
int i[5]={1,2,3,4,5};
int *p, t;

p=i;

for(t=0; t<5; t++) printf("%d",p[t]);
}
```

1.7 Покажчики і рядки. Оскільки ім'я масиву без індексів є покажчиком на перший елемент цього масиву, то при обробці рядків з неіндексованими іменами рядкових масивів їм будуть передаватися не самі рядки, а покажчики на них. Для ілюстрації цього, розглянемо функцію *strcmp()*, що порівнює рядки:

```
strcmp(s1,s2)
{
char *s1, *s2;
while(*s1)
        if(*s1-*s2)       // якщо не дорівнює, тоді повернути
                return *s1-*s2;   // різниця
        else  {
                s1++;
                s2++;
                }
return '\0'; // дорівнює
}
```

Усі рядки в мові С++ закінчуються нулем, що має значення «хибно». Таким чином, умова в операторі

*while(*s1)*

буде істиною доти, поки комп'ютер не досягне кінця рядка. Тут *strcmp()* буде повертати нуль, якщо *s1* дорівнює *s2*. Якщо *s1* менше *s2*, функція буде повертати від'ємне значення. І, навпаки, значення, що повертається, буде додатнім, якщо *s1* більше *s2*.

При використанні строкової константи в будь-якому виразі комп'ютер сприймає цю константу як покажчик на перший символ у рядку. Наприклад, у наступній програмі на екран виведеться фраза *«ця програма працює»*:

```
#include <stdio.h>
main()
{
char *s;

s="ця програма працює";
printf(s);
}
```

1.8 Звернення до адреси елемента масиву. До цього розглядалося присвоєння покажчику адреси тільки першого елемента масиву. Однак це можна робити і з адресою будь-якого окремого елемента масиву шляхом додавання & до індексованого імені. Наприклад, у наступному виразі покажчику *p* присвоюється адреса третього елемента масиву *x*:

p = &x[2];

Особливо зручно користуватися цим правилом при виділенні підрядка. Наприклад, наведена програма виводить на екран залишок введеного рядка після першого пробілу:

```
// Вивести на екран залишок рядка після першого пропуску
#include <stdio.h>
main()
{
char s[80];
char *p;
int i;

printf("Введіть рядок: ");
gets(s);

// знайти перший пропуск чи кінець рядка
for(i=0; s[i] && s[i]!=' '; i++)
p = &s[i];
printf(p);
}
```

Ця програма працює, тому, що *p* буде вказувати або на пропуск, якщо він є, або на нуль, якщо в рядку немає пропусків. Якщо *p* указує на пропуск, то програма виведе на екран цей пропуск і потім залишок рядка. Наприклад, якщо вводиться фраза *«ця програма»*, функція *printf()* надрукує спочатку пропуск і потім *«програма»*. Якщо *p* укаже на нуль, то нічого не виводиться на екран.

1.9 Масив покажчиків. У мові С++ допускається організовувати масиви покажчиків. Так, щоб оголосити масив цілих покажчиків розміром 10, необхідно записати

```
int *x[10];
```

Щоб присвоїти адресу цілої змінної з ім'ям *var* третьому елементу масиву покажчиків, необхідно записати:

```
x[2] = &var;
```

Для одержання значення змінної *var* необхідно записати:

```
*x[2]
```

1.10 Покажчики, що вказують на інші покажчики. Масив покажчиків – те ж саме, що і покажчики на покажчики. Покажчик на покажчик є однією з форм багаторівневої (непрямої) адресації. Як видно з рисунка 2, у випадку нормального покажчика, значенням його є адреса змінної, що утримує необхідне значення. У випадку покажчика на покажчик, перший покажчик містить адресу другого покажчика, що, у свою чергу, указує на змінну, що утримує необхідне значення.

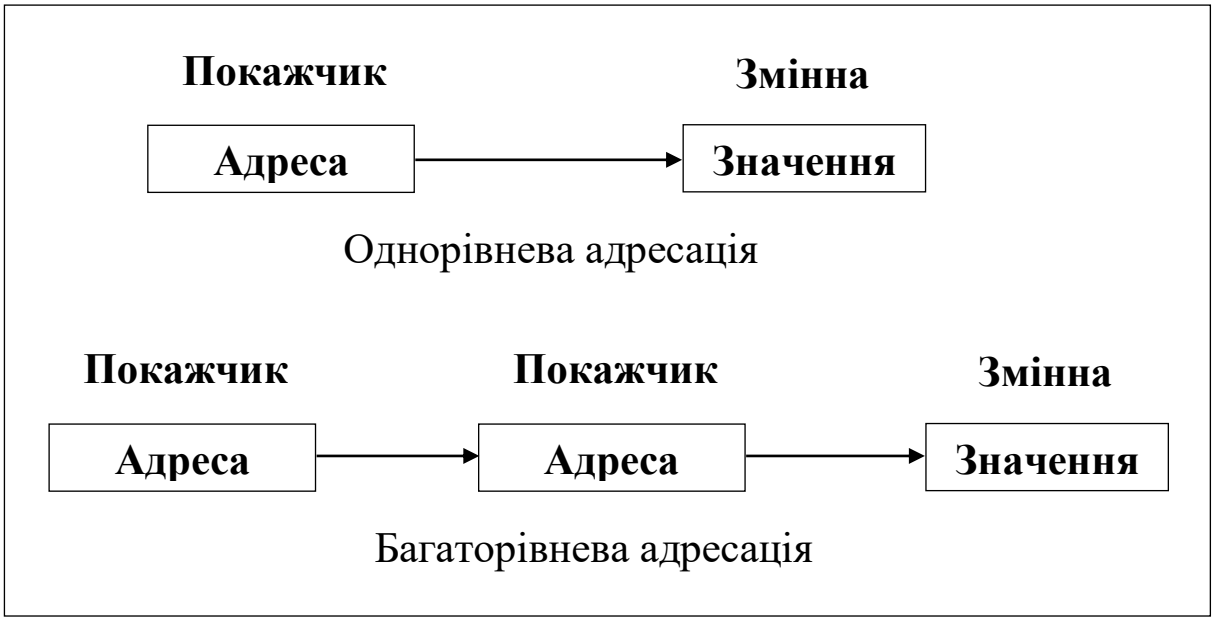

Рисунок 2 – Однорівнева і багаторівнева адресації

Рівень непрямої адресації можна підвищувати до будь-якого ступеня, однак на практиці мало зустрічається випадків, коли це необхідно чи виправдано. Адресацію занадто високого рівня важко відстежити, і вона є джерелом багатьох помилок.

Для оголошення змінної, що є покажчиком на покажчик, необхідно до її імені додати додаткову зірочку.

Наприклад, так оголошується змінна *newbalance*, що є покажчиком на покажчик типу *float*:

*float **newbalance;*

Тут важливо зрозуміти, що *newbalance* не є покажчиком на число з плаваючою точкою, а є покажчиком на покажчик типу *float*.

Для доступу до кінцевого значення, на яке вказує покажчик на покажчик, необхідно два рази використовувати оператор *, як показано в наступному прикладі:

```
#include <stdio.h>
main()
{
int x, *p, **q;

x = 10;
p = &x;
q = &p;

printf("%d",**q);          // виведення значення x
}
```

Тут *p* з'являється покажчиком на цілу змінну, а *q* - покажчиком на покажчик *p*. Функція *printf()* виводить на екран число *10*.

1.11 Ініціалізація покажчиків. Після оголошення покажчик–змінної, але перед присвоюванням їй якого-небудь значення, вона буде мати деяку невідому величину. Якщо спробувати використовувати покажчик, не присвоївши йому визначеного значення, то це може привести не тільки до аварійного завершення програми, але і до необхідності перезавантаження операційної системи.

Існує домовленість, що покажчик, який не вказує ні на яку змінну, повинен мати нульове значення. Однак використання нульового покажчика також не є безпечним. Так, якщо його помістити в ліву частину оператора присвоювання, може відбутися збій програми.

Оскільки нульовий покажчик не вказує ні на яку змінну, його можна використовувати в деяких випадках для спрощення програми. Наприклад, нульовий покажчик може відзначати

кінець масиву покажчиків. При обробці такого масиву програмою, кінець масиву буде легко виявлений. Це можна проілюструвати в циклі *for*, що наводиться нижче:

```
for(t=0; p[t]; ++t)
        if(! strcmp(p[t],name)) break;
```

Цей цикл буде працювати доти, поки не буде виявлена рівність величин чи зустрінуто нульовий покажчик. Остання умова напевно виконається, коли буде досягнуто кінець масиву.

У написаних професійно програмах мовою С++ прийнято ініціалізувати рядки. Наприклад:

```
char *p="hello, world\n";
```

Як видно, тут покажчик *p* не є масивом. Допустимість такої ініціалізації зв'язана з особливістю роботи компілятора: компілятор С++ створює, так звану, рядкову таблицю, що містить рядкові константи програми. Таким чином, у наведеному вище операторі оголошення, покажчику *p* присвоюється адреса рядка *"hello, world"* з рядкової таблиці. В усій програмі *p* можна використовувати як будь-який інший рядок.

Наприклад, наведена програма цілком правомірна:

```
#include <stdio.h>
char *p="hello, world";
main()
{
int t;

// друкувати рядок у прямому і зворотному порядках
printf(p);
for(t=strlen(p)–1; t>–1; t– –) printf("%c",p[t]);
}
```

У цьому прикладі використовується бібліотечна функція визначення довжини рядка *strlen()*.

1.12 Помилки, що найбільш часто зустрічаються при використанні покажчиків. При програмуванні ніщо не завдасть більше неприємностей, ніж невірно визначений покажчик. Взагалі, покажчики є досить могутнім засобом програмування і необхідні в багатьох програмах, але якщо один з них раптом одержить не те значення, то цю помилку буде дуже важко відшукати.

Труднощі пошуку помилки покажчика полягають не в самому покажчику, а в тому, що при використанні його для читання і запису даних, останні попадають у невідому область пам'яті.

Якщо прочитати дані з цієї області пам'яті, то в гіршому випадку вони виявляться безглуздими. Однак, якщо дані записуються в невизначену область пам'яті, вони можуть записатися на програму чи інші дані. Ця помилка може проявитися тільки при виконанні програми. Причому не буде ніякого доказу, який підказує, що помилка зв'язана з покажчиком.

Розглянемо кілька характерних помилок. Перша з них, що найбільш часто зустрічається, - це непроініціалізований покажчик. Розглянемо програму:

```
// Ця програма помилкова
main()
{
int x, *p;
x = 10;
*p = x;
}
```

В цій програмі значення *10* заноситься в невизначену область пам'яті, тому що *p* не одержало ніякого значення. Ця помилка часто залишається непоміченою, якщо програма невелика і існує велика імовірність, що *p* буде містити безпечну адресу, тобто поза областю програми і даних. Однак зі збільшенням програми зростає й імовірність того, що *p* буде вказувати на важливі комірки пам'яті. У цьому випадку програма припинить свою роботу. Щоб уникнути такої помилки, необхідно завжди перевіряти покажчики на зміст значимої адреси.

Друга загальна помилка виникає від незнання правил використання покажчиків. Розглянемо програму:

```
// Ця програма помилкова
#include <stdio.h>
main()
{
int x, *p;
x = 10;
p = x;
printf("%d",*p);
}
```

Тут функція *printf()* не роздрукує значення *x*, що дорівнює *10*, а виведе невизначене значення. Це зв'язане з тим, що оператор

```
p = x;
```

записаний невірно, тому що значення *10* присвоюється покажчику *p*, що може містити тільки адресу, а не значення. Правильний запис повинен виглядати так:

```
p = &x;
```

Але, не дивлячись на те, що неправильне використання покажчиків приводить до важкознайдених помилок, зовсім відмовлятися від їх застосування не потрібно.

2 ФАЙЛИ

2.1 Загальні відомості. У файлах розміщаються дані, призначені для тривалого збереження. Кожному файлу присвоюється унікальне ім'я, що використовується при зверненні до нього. Файли широко застосовуються при розв’язанні різних задач. Наприклад, у деяких задачах щораз необхідно вводити з клавіатури велику кількість інформації, а це

дуже утомливо. Набагато зручніше ввести цю інформацію один раз і зберегти її у файлі на диску. Тоді для одержання необхідних даних досить скористатися інформацією на диску, а не вводити її щораз заново.

Перш ніж одержати можливість читання чи запису інформації у файл, він повинен бути відкритий. Це можна зробити за допомогою бібліотечної функції *fopen()*. Вона бере зовнішнє подання – фізичне ім'я файлу (наприклад, *a:\test.cpp*, де *a:* – диск, на якому розміщений файл; *test* – ім'я файлу; *.cpp* - розширення файлу) і ставить йому у відповідність внутрішнє логічне ім'я, що використовується далі в програмах. Логічне ім'я – це покажчик на необхідний файл (на область пам'яті, де міститься інформація про файл). Покажчик на файл необхідно оголошувати. Зробити це можна таким чином:

*FILE *fopen( ), *lst;*

де *FILE* – ім'я типу, описане в стандартній бібліотеці *stdio.h*;
lst – покажчик на файл (логічне ім'я);
fopen() – функція, що видає покажчик на файл.

Звертання до функції *fopen()* у програмі виконується так:

lst=fopen(<фізичне ім'я файлу>,<режим>);

Фізичне ім'я файлу може, наприклад, бути *"prn"* – для пристрою печатки, *"a\:test.cpp"* – для файлу *test.cpp* на диску *a:* і т.п. Режими використання файлу представлені в таблиці 1. Якщо файл відкривається для запису чи доповнення, але ще не існує, то він створюється. Відкриття існуючого файлу для запису приводить до знищення його старого вмісту. Спроба прочитати неіснуючий файл – це помилка (*fopen()* видає покажчик зі значенням *NULL*).

Таблиця 1 – Режими використання файлу

Режим	**Зміст**
"r"	Відкрити текстовий файл для читання
"w"	Створити текстовий файл для запису
"a"	Додавати в текстовий файл
"r+"	Відкрити текстовий файл для читання/запису
"w+"	Створити текстовий файл для читання/запису
"rb"	Відкрити двійковий файл для читання
"wb"	Створити двійковий файл для запису
"ab"	Додати в двійковий файл

Для роботи з файлами використовуються бібліотечні функції *fprintf(), fscanf(), fgets(), fputs(), fread(), fwrite().* Після закінчення роботи з файлом він повинен бути закритий. Це робиться за допомогою бібліотечної функції *fclose()*, наприклад, *fclose(lst.)*

2.2 Функції *fprintf()* і *fscanf()*. Функції *fprintf()* і *fscanf()* поводяться точно так, як функції *printf()* і *scanf()*, за винятком того, що вони працюють з дисковими файлами. Загальні формати оголошення функцій *fprintf()* і *fscanf()* мають такий вид:

fprintf(fp,"<управляючий_рядок>",<список_аргументів>);
fscanf(fp,"<управляючий_рядок>",<список_аргументів>);

де *fp* – файловий покажчик, що повертається викликом функції *fopen()*.

Як уже було сказано раніше, за винятком напрямку їхнього виведення в дискові файли, що обумовлюється покажчиком *fp*, ці функції працюють так само, як функції *printf()* і *scanf()* відповідно.

2.3 Функції *fgets()* і *fputs()*. Загальні формати оголошення функцій *fgets()* і *fputs()* мають такий вид:

fputs(<ім'я_масиву>,<файловий_покажчик>);
fgets(<ім'я_масиву>,<довжина>,<файловий_покажчик>);

Функція *fputs()* працює так само, як функція *puts()*, за винятком того, що функція *fputs()* пише ланцюжок у специфікований файл. Функція *fgets()* читає ланцюжок із специфікованого файлу доти, доки вона не прочитає або символ нового рядка, або *довжину–1* символів.

2.4 Функції *fread()* і *fwrite()*. Функції *fread()* і *fwrite()* дозволяють читати і писати блоки даних. Загальні формати оголошення цих функцій мають наступний вид:

fread(<буфер>,<кількість_байтів>,<рахунок>,
<файловий_покажчик>);
fwrite(<буфер>,<кількість_байтів>,<рахунок>,
<файловий_покажчик>);

У випадку функції *fread()* буфер є покажчиком на область пам'яті, що буде приймати дані, що зчитуються з файлу. Для функції *fwrite()* буфер є покажчиком на інформацію, що буде записуватися у файл. Для обох цих функцій кількість байтів специфікує кількість байтів, що повинні бути зчитані чи записані. Аргумент *"рахунок"* визначає, скільки елементів буде зчитано чи записано. Файловий покажчик – це покажчик на попередньо відкритий файл.

Оскільки файл відкривається для двійкових даних, функції *fread()* і *fwrite()* можуть читати і писати інформацію будь-якого типу.

2.5 Функція *fclose()*. Функція *fclose()*використовується для закриття файлу, що був відкритий за допомогою функції *fopen()*. Перед завершенням програми необхідно закрити усі файли. Функція *fclose()* записує у файл дані, що ще залишаються в дисковому буфері, і виконує формальне на рівні операційної системи закриття файлу. Невдача при закритті файлу спричиняє всілякі неприємності, включаючи втрату даних, руйнування файлів і можливі помилки в програмі. Як відомо, операційна система обмежує кількість відкритих файлів, яку можна мати в будь-який час одночасно, так що можливо доведеться закрити один файл, перш ніж можна буде відкрити інший.

Загальний формат функції *fclose()* має такий вид:

*fclose(FILE *fp);*

де *fp* – покажчик файлу, що повертається викликом функції *fopen()*.

Повернення нульового значення означає успішну операцію закриття; будь-яке інше значення повернення показує помилку.

2.6 Програми, що використовують функції *fopen(), fprintf(), fscanf(), fgets(), fputs(), fread(), fwrite(), fclose()*. Перша програма показує організацію виведення інформації на друкувальний пристрій:

```
// Робота з файлами(виведення інформації принтер)
#include <stdio.h>
main()
{
int i=150;
FILE *fopen(), *lst;   /* FILE – ім'я файлового типу (подібно int чи
                          float); fopen() -бібліотечна функція
                          відкриття файлу (вона повертає покажчик
                          на внутрішнє ім'я відкритого файлу); lst -
                          покажчик на файл (об'єкт типу FILE) */
lst = fopen("prn","w");  /* lst одержує адресу відкритого файлу з
                          ім'ям prn, призначеного для запису в нього
                          інформації ("w"); prn – стандартне ім'я
                          пристрою печатки, тому вся інформація з
                          файлу з адресою lst буде виводитися на
                          друкуючий пристрій */
fprintf(lst,"число i = %d \n",i);   /* перший параметр функції
fprintf()
                                   – це покажчик на відповідний файл; на
                                   друк буде виведено рядок: число i =
                                   150 */
```

```
fclose(lst);                    /* функція fclose() закриває файл із
                                покажчиком lst; тепер посилання lst
                                можна використовувати
                                для іншого файлу */
}
```

Друга програма показує організацію виведення інформації на дискету:

```
// Робота з файлами (запис інформації на дискету)
#include <stdio.h>
main()
{
int i=150;
FILE *fopen(), *lds;
lds = fopen("a:primer.tst","w");  /* перший параметр функції
                                fopen() є специфікацією файлу,
                                що складається з імені поточного
                                накопичувача (a:), імені файлу
                                (primer) і типу файлу (tst)*/
fprintf(lds,"%d\n",i);          /* значення змінної і (i=150) буде
                                записане у файл primer.str на
                                диску a: */
fclose(lds);
}
```

Третя програма показує організацію читання інформації з дискети:

```
// Робота з файлами (читання інформації з файлу на дискеті)
#include <stdio.h>
main()
{
int i;
char s[50];
FILE *fopen(), *rsd;
rsd = fopen("a:primer.tst","r"); /* другий параметр функції fopen() –
                                "r", що говорить про необхідності
                                читання інформації */
```

```
fscanf(rsd,"%d",&i);        /* з файлу primer.tst (диск a:) буде
                               прочитане значення і */
printf("число і = %d (початкове значення і)\n",i);
while(fscanf(rsd,"%d",&i)!=EOF)
printf("число і = %d\n",i);  /* послідовне виведення цілих чисел з
                               файлу;  виведення припиняється, коли
                               буде  досягнуто кінець файлу (EOF) */
fclose(rsd);
}
```

Четверта програма ілюструє доповнення файлу на дискеті (доповнення у файл нових даних):

```
// Робота  з файлами (доповнення файлу на дискеті)
#include <stdio.h>
main()
{
int з;
FILE *fopen(), *lds;
lds = fopen("a:primer.tst","a"); /* другий параметр функції fopen() –
                               "a", що  говорить про можливості
                               доповнення файлу */
puts("Введіть ціле число: ");
scanf("%d",&c);
fprintf(lds,"%d\n",c);       /* файл primer.tst буде доповнено новим
                               числом */
fclose(lds);
}
```

З початком роботи будь-якої програми автоматично відкриваються три файли і для них визначаються відповідні покажчики. Перший з них - це файл для стандартного введення інформації з клавіатури з покажчиком *stdin*, другий - для стандартного виведення інформації на екран дисплея з покажчиком *stdout*, третій – для стандартного виведення помилок на екран дисплея з покажчиком *stderr*. Об'єкти *stdin*, *stdout*, *stderr* - константи, а не змінні, і їм не можна що-небудь присвоювати. В ЕОМ можна перепризначувати вхідні і вихідні

потоки. Іншими словами, можна зробити так, щоб вся інформація виводилася не на дисплей, а на друкувальний пристрій. Подібно тому, як команда операційної системи

c:\dir > prn <*Enter*>

приводить до печатки директорії диску *c:* на принтері, аналогічна команда операційної системи

c:\tc > prn <*Enter*>

перенаправляє на друкувальний пристрій всю вихідну інформацію системи C++. Однак, при будь-якому перевизначенні інформація, записана у файл із покажчиком *stderr*, з'явиться на екрані дисплея. Тому цей файл звичайно використовують для збереження різних діагностичних повідомлень.

П'ята програма показує, як записати у файл рядок символів, і демонструє виведення інформації в стандартний файл для помилок:

```
// Робота з файлами (введення і запис рядків). Виведення інформації
// в стандартний файл для помилок
#include <stdio.h>
main()
{
char str[50];
FILE *fopen(), *fl;
fl = fopen("a:fal.f","w");
puts("Введіть рядок (до 49 символів без пропуску)");
scanf("%s",str);
fputs(str,fl);          // функція fputs() записує у файл fal.f рядок str
fprintf(stderr,"Демонстрація виведення в стандартний файл для помилок");
// текст у функції fprintf() буде виведено на екран
fclose(fl);
}
```

Введений рядок не більше 49 символів буде записано у файл із специфікацією *a:fal.f* за допомогою функції *fputs()*, перший параметр якої *str* – ідентифікатор записуваного рядка, а другий – *fl* – покажчик на відповідний файл.

Шоста програма показує, як прочитати рядок з файлу:

```
// Робота з файлами (читання і виведення рядків)
#include <stdio.h>
main()
{
char str[25];
FILE *fopen(), *fr;
fr = fopen("a:fal.f","r");
fgets(str,20,fr);          /* функція fgets() зчитує з файлу fal.f рядок
                           str,  максимальна довжина якої 20–1=19;
                           рядок, що одержали, закінчується символом
                           '\0' */
printf("str = %s",str);
fclose(fr);
}
```

Читання рядка забезпечує функція *fgets()*, перший параметр якої *str* – це ідентифікатор рядка, що читається, другий - її максимальна довжина і третій – *fr* – покажчик на відповідний файл.

Сьома програма пише в дисковий файл числа з плаваючою точкою:

```
// Робота з файлами (запис числа з плаваючою точкою у
дисковий
// файл)
#include <stdio.h>
main()
{
FILE *fp;
float f=12.23;
```

```
if((fp=fopen("test","wb"))==NULL)
        {
        printf("файл не відкривається\n");
        return;
        }
fwrite(&f,sizeof(float),1,fp);
fclose(fp);
}
```

У програмі використовується функція *sizeof()*, що повертає величину об’єму пам'яті в байтах, що виділяється під змінну чи тип, що вказується операндом у дужках.

3 ВИКОРИСТАННЯ ГРАФІКИ

3.1 Загальні відомості. У мові програмування С++ є графічна бібліотека, що містить велику кількість функцій, що дозволяють здійснювати різноманітні графічні операції. Настройка цих функцій на роботу з конкретним відеоадаптером досягається за рахунок підключення необхідного графічного драйвера. **Адаптер** - це електронна схема, що управляє зовнішнім приладом ПЕОМ. **Драйвер** - це спеціальна програма для управління тим чи іншим приладом комп'ютера.

Графічні драйвери знаходяться в окремих файлах з розширеннями BGI (Borland Graphics Interface).

При необхідності використання графічних функцій у вхідному модулі програми треба помістити директиву *#include <graphics.h>*, що підставляє на своє місце вміст файлу заголовка з прототипом функцій.

3.2 Ініціалізація графічної системи. Виконання кожної програми починається в текстовому режимі. Для переведення системи у графічний режим необхідне виконання функції *initgraph ( )*:

initgraph(&<графічний_драйвер>, &<графічний_режим>,
"<шлях_до_bgi_файлів>");

Графічні драйвери використовуються такі :

CGA	EGA64
EGA	IBM8514
EGAMONO	ATT400
HERCMONO	PC3270
VGA	DETECT (Автовизначення
MCGA	драйвера)

Деякі графічні режими :

EGAHI	640 x 350	16 кольорів
EGAMONOHI	640 x 350	2 кольори
HERCMONOHI	720 x 348	2 кольори
VGALO	640 x 200	16 кольорів
VGAMED	640 x 350	16 кольорів
VGAHI	640 x 480	16 кольорів
PC3270HI	720 x 350	2 кольори
IBM8514LO	640 x 480	256 кольорів
IBM8514HI	1024 x 768	256 кольорів

Якщо BGI-драйвери знаходяться в поточній директорії, то замість параметра *"<шлях_до_bgi_файлів>"* можна задати пустий рядок :

initgraph (&gdriver, &gmode, "");

У протилежному випадку шлях до драйвера записується в такому форматі:

".. \\bgi\\drivers "

Існує функція *graphresult()*, яка аналізує причини невірного виконання графічної операції :

graphresult ( ).

Якщо виконання графічної функції завершилося успішно, функція повертає значення 0. У протилежному випадку, значення, що повертається, ідентифікує причину невдачі.

Існує можливість надрукувати повідомлення, що відповідає значенню коду помилки. Для цього використовується функція

grapherrormsg (<код_помилки>);

Наприклад,

```
error = graphresult( );
printf ( "Помилка графіки : %s ", grapherrormsg ( error ) );
```

Для тимчасового переходу у текстовий режим в графічній бібліотеці існує функція

restorecrtmode ( );

що відновлює той текстовий режим, що був перед зверненням до функції *initgraph()*. Повернутися у графічний режим можна за допомогою функції

setgraphmode (<графічний_режим>);

де *графічний режим* - цілочисловий номер режиму, що допущений для даного драйвера. Попередній графічний режим можна відновити, якщо його номер був своєчасно визначений за допомогою функції

getgraphmode( );

і збережений в програмі.

Одержати максимальне значення номера графічного режиму, що допущений для поточного драйвера, можна за допомогою функції

getmaxmode ( ).

Після закінчення роботи з графічною системою її необхідно відключити. Це виконує функція

closegraph ( ).

3.3 Робота з вікнами і координатами. Після встановлення графічного режиму весь екран по умовчанню стає графічним вікном. Всередині основного графічного вікна (екрана) можна виділити прямокутне вікно зі своєю системою координат. Виконується це за допомогою функції

setviewport (<x1>, <y1>, <x2>, <y2>, <відсікання>);

де *x1, y1* - координати лівого верхнього кута вікна;
x2, y2 - координати правого нижнього кута вікна.

Якщо параметр *<відсікання>* дорівнює 1, то ті елементи зображення, що не вміщаються у вікні, будуть відсічені, а якщо 0, то межі вікна проігноруються.

Очищення графічного вікна виконує функція

clearviewport ( );

Очищення екрана виконується за допомогою функції

cleardevice ( );

Всі встановлені раніше графічними процедурами параметри скидаються і набувають значення по умовчанню.

Для відновлення параметрів, прийнятих по умовчанню, є функція

graphdefaults ( );

Відновлюються параметри, встановлені функцією *initgraph ( ).*

Координати лівого верхнього кута графічного екрана (0, 0), а координати правого нижнього кута визначаються функціями

getmaxx ( ); і *getmaxy ( );* .

Ці функції повертають максимальну координату по горизонталі і по вертикалі відповідно.

Координати графічного курсора завжди відраховуються щодо лівого верхнього кута вікна. Графічний курсор є невидимим, але його поточні горизонтальні і вертикальні координати можуть бути визначені за допомогою функцій

getx ( ); і *gety ( );* .

Ці координати подаються даними типу *int* і можуть бути як додатними, так і від'ємними.

Переустановлення покажчика позиції виконує функція

moveto (<x>, <y>);

де *x, y* - нові координати в системі координат вікна.

3.4 Установлення кольорів, стилів ліній і зафарбовування, шрифтів. Існує можливість встановлювати окремо основний колір, що малює, і колір фону. Поточне значення основного кольору (колір символів і ліній, що виводяться) встановлюється функцією

setcolor (<колір>);

Кольором фону можна управляти за допомогою функції

setbkcolor (<колір>);

де *колір* - один з перелічених кольорів чи код.

Код	Колір	Значення
0	BLACK	чорний
1	BLUE	синій
2	GREEN	зелений
3	CYAN	бірюзовий
4	RED	червоний
5	MAGENTA	малиновий
6	BROWN	коричневий
7	LIGHTGRAY	світло-сірий
8	DARKGRAY	темно-сірий
9	LIGHTBLUE	світло-синій
10	LIGHTGREEN	світло-зелений
11	LIGHTCYAN	світло-бірюзовий
12	LIGHTRED	світло-червоний
13	LIGHTMAGENTA	світло-малиновий
14	YELLOW	жовтий
15	WHITE	Білий

Максимальне значення кольору повертає функція

getmaxcolor ( );

Для встановлення характеру і товщини ліній геометричних об'єктів використовується функція

setlinestyle (<стиль>, <зразок>, <товщина>);

Коди для параметра *стиль* (тільки для кусково-лінійних графічних примітивів)

Код	Стиль
0	SOLID_LINE (суцільна)
1	DOTTED_LINE (з точок)
2	CENTER_LINE (з точок і тире)
3	DASHED_LINE (пунктирна)
4	USERBIT_LINE (що визначається користувачем)

Параметр *зразок* задається тільки тоді, коли *стиль* дорівнює 4 (в інших випадках він ігнорується, тому його можна робити рівним 0).

Допущені значення для параметра *товщина*:

1 NORM_WIDTH (лінія в один піксель)
3 THICK_WIDTH (лінія в три пікселі)

Існує можливість зафарбувати виділену на екрані замкнуту область певним засобом. Для установлення стилю і кольору зафарбування використовується функція

setfillstyle (<стиль>, <колір>);

Допущені значення параметра *стиль*:

0	EMPTY_FILL	Заповнення пустотою
1	SOLID_FILL	Заповнення суцільним кольором
2	LINE_FILL	Заповнення горизонтальною лінією ---
3	LTSLASH_FILL	Заповнення ///
4	SLASH_FILL	Заповнення зтовщенними лініями ///
5	BKSLASH_FILL	Заповнення зтовщенними лініями \\\
6	LTBKSLASH_FILL	Заповнення \\\
7	HATCH_FILL	Горизонтальні грати
8	XHATCH_FILL	Скісні грати
9	INTERLEAVE_FILL	Коса лінія з прошарками
10	WIDE_DOT_FILL	Рідко розташовані точок
11	CLOSE_DOT_FILL	Щільно розташовані точок

При виведенні текстового повідомлення існує можливість вибору одного з декількох шрифтів, розміру символів, що виводяться, і напрямку тексту. Ці параметри задаються за допомогою функції

settextstyle (<шрифт>, <напрямок>, <розмір_символів>);

Допущені значення параметра *шрифт* :

0	DEFAULT_FONT (стандартний)
1	TRIPLEX_FONT (типу тріплекс)
2	SMALL_FONT (зменшений)
3	SANS_SERIF_FONT (прямий)
4	GOTHIC_FONT (готичний)

Допущені значення параметра *напрямок* :

0	HORIZ_DIR (зліва_направо)
1	VERT_DIR (знизу_вгору)

Існує можливість вирівняти текст за допомогою функції

settextjustify (<горизонтально>, <вертикально>);

Допущені значення параметрів *горизонтально* і *вертикально:*

Параметр	**Назва**	**Значення**	**Вирівнювання**
Горизонтально	LEFT_TEXT	(0)	ліворуч <
	CENTER_TEXT	(1)	> в центрі <
	RIGHT_TEXT	(2)	> праворуч
Вертикально	BOTTOM_TEXT	(0)	знизу вгору
	CENTER_TEXT	(1)	в центрі
	TOP_TEXT	(2)	зверху вниз

Виведення тексту у вікно можливе за допомогою стандартних функцій *printf()* і *puts()*. Але вони обмежені виглядом і розміром символів шрифту, а також можливістю розміщення символів тільки в тих позиціях екрана, що допускаються в текстовому режимі. Спеціальні ж графічні функції виводу тексту дозволяють працювати з ним, як з повноправним елементом графіки.

Функцій виводу графічного тексту - дві:

outtext (<рядок>);

виводить рядок символів, починаючи з поточної графічної позиції;

outtextxy (<x>, <y>, < рядок >);

виводить рядок символів, починаючи з позиції *(x, y).*

3.5 Графічні примітиви. Графічні примітиви призначені для рисування різноманітних геометричних об'єктів.

Засіб взаємодії прямих ліній, що виводяться, задає функція

setwritemode (<режим>);

Допущені значення параметра *режим:*

0	COPY_PUT
1	XOR_PUT

За допомогою цієї функції встановлюється засіб, яким код кольору, що рисує, встановлений функцією *setcolor()*, буде взаємодіяти з атрибутами пікселів, що вже знаходяться на місці об'єкту, що рисується. Якщо установлено режим 0, то лінія, що малюється, затирає те, що було на екрані. Якщо задано режим 1, то лінія, що рисується, комбінується з тим, що було на екрані. Для цього використовується операція XOR. Цікава властивість цієї операції в тому, що вивід двічі на одне і те ж місце лінії призводить до її затирання і відновлення вхідного зображення на екрані. Ця функція застосовується для роботи з кусково-лінійними зображеннями.

Для виведення пікселя в специфікованій точці екрана використовується функція

putpixel (<x>, <y>, <колір>);

де *x, y* - координати пікселя в системі координат вікна.

Рисує лінію між двома специфікованими точками функція

line (<x1>, <y1>, <x2>, <y2>);

де *x1, y1* - координати початку відрізка прямої;
x2, y2 - координати кінця відрізка прямої.

При цьому положення покажчика поточної позиції не змінюється.

За допомогою функції

lineto (<x>, <y>);

рисується лінія з поточної точки у точку з новими координатами. При цьому покажчик поточної позиції зміщується зі старої точки у нову.

Лінії рисуються поточним стилем, встановленим функцією *setlinestyle( )*, і поточним кольором, встановленим функцією *setcolor( ).*

Рисує прямокутник функція

rectangle (<x1>, <y1>, <x2>, <y2>);

де *x1, y1* - координати лівого верхнього кута;
x2, y2 - координати правого нижнього кута.

Зафарбування прямокутника встановленим стилем виконує функція

bar (<x1>, <y1>, <x2>, <y2>);

де *x1, y1* - координати лівого верхнього кута;
x2, y2 - координати правого нижнього кута.

Паралелепіпед із зафарбованою передньою гранню можна нарисувати за допомогою функції

bar3d (<x1>, <y1>, <x2>, <y2>, <глибина>, <дах>);

Паралелепіпед обрамляється зовнішнім контуром. Якщо параметр *глибина* буде дорівнювати 0, то буде отримано

обрамований прямокутник. Якщо параметр *дах* дорівнює 1 (TOP_ON), то буде нарисована верхня грань; якщо він дорівнює 0 (TOP_OFF), то верхня грань не рисується. Наприклад, паралелепіпеди, що розташовані один на одному:

```
setbkcolor ( WHITE );
setcolor ( GREEN );
bar3d ( 230, 50, 250, 150, 15, 1 );
bar3d ( 220, 150, 260, 180, 15, 1 );
bar3d ( 300, 150, 340, 180, 15, 0 );
bar3d ( 300, 50, 340, 150, 15, 1 );
```

Накреслити ламану лінію дозволяє функція

drawpoly (<кількість_вершин>, <покажчик_на_масив_цілих>);

Кожна пара чисел інтерпретується як пара координат чергової вершини ламаної. Наприклад,

```
int  x [ 10 ] = { 20, 200, 500, 20, 300, 150, 300, 200 };
x [ 8 ] = x [ 0 ];          // замикання контуру
x [ 9 ] = x [ 1 ];
drawpoly ( 5, x );
```

Побудувати і зафарбувати багатокутник дозволяє функція

fillpoly (<кількість_вершин>, <покажчик_на_масив_цілих>);

Ця функція завжди замикає контур, з'єднуючи першу точку списку вершин з останньою.

Для побудови кривих використовуються наступні функції.

Для рисування кола з центром у точці *(x,y)* із заданим радіусом у пікселях:

circle (<x>, <y>, <радіус>);

Для рисування дуги кола:

arc (<x>, <y>, <початковий_кут>, <кінцевий_кут>, <радіус>);

Кут виражається в градусах і відраховується проти годинникової стрілки.

Для рисування дуги еліпса:

ellipse (<x>, <y>, <початковий_кут>, <кінцевий_кут>, <x_радіус>, <y_радіус>);

де *x, y* - центральна точка;
x_радіус, y_радіус - довжини напівосей еліпса в пікселях.

Зафарбований еліпс з контуром можна одержати, використовуючи функцію

fillellipse (<x>, <y>, <x_радіус>, <y_радіус>);

Зафарбований круговий сектор з контуром рисує функція

pieslice(<x>, <y>, <початковий_кут>, <кінцевий_кут>, <радіус>);

Зафарбований еліптичний сектор з контуром рисує функція

sector (<x>, <y>, <початковий_кут>, <кінцевий_кут>, <x_радіус>, <y_радіус>);

Заповнення кольором обмеженої області виконується функцією

floodfill (<x>, <y>, <колір_межі>);

де *x, y* - координати точки, з якої починається заповнення замкнутої області в усіх напрямках до *кольору_межі*.

Встановлення кольору заповнення слід здійснювати раніше за допомогою функції *setfillstyle ( )*, наприклад,

```
setcolor ( 4 );
rectangle ( 10, 10, maxx-20, maxy-20 );
setfillstyle ( SOLID_FILL, 2 );
floodfill ( 12, 12, 4 );
```

Приклад програми, що реалізує деякі наведені функції бібліотеки *graphics.h :*

```
#include <graphics.h>
#include <stdlib.h>   // бібліотека стандартних функцій
#include <string.h>   // бібліотека обробки рядків
#include <stdio.h>      // бібліотека стандартного введення-виведення
#include <conio.h>
main ( )
{
// Назва стилів
char *fname [ ] = { "EMPTY_FILL ", "SOLID_FILL ", "LINE_FILL",
            "LTSLASH_FILL ", "SLASH_FILL ", "BKSLASH_FILL",
            "LTBKSLASH_FILL", "HATCH_FILL","XHATCH_FILL",
            "INTERLEAVE_FILL", "WIDE_DOT_FILL",
            "CLOSE_DOT_FILL", "USER_FILL"
                  };
 // Автовизначення графічного драйвера, оголошення змінних
int gdriver = DETECT, gmode, errorcode;
int style, midx, midy;
char stylestr [ 40 ];

// Ініціалізація графічної системи
initgraph ( &gdriver, &gmode, " " );

// Аналіз результатів ініціалізації
errorcode = graphresult ( );
if ( errorcode != grOk )            // випадок помилки
    {
    printf ( "Графічна помилка : %s \n ", grapherrormsg ( errorcode ) );
    printf ( "Натисніть будь-яку клавішу для зупинки... " );
    getch ( );
```

```
    exit ( 1 );                    // Завершення з кодом помилки
    }
midx = getmaxx ( ) / 2;
midy = getmaxy ( ) / 2;
for ( style = EMPTY_FILL; style < USER_FILL; style++ )
    {
     // Вибір стилю заповнення і кольору
     setfillstyle ( style, style );

    // Перетворення стилю у рядок символів
     strcpy ( stylestr, fname [ style ] );

    // Виконання смуги
     bar3d ( 0, 0, midx-10, midy, 0, 0 );

    // Виведення повідомлення про стиль
    outtextxy ( midx, midy, stylestr );

   // Очікування натиску клавіші
    getch ( );
    cleardevice ( );
     }
// Відключення режиму графіки
outtextxy ( 10, 450, "Натисніть будь-яку клавішу ... " );
getch ( );
closegraph ( );
return 0;
}
```