Лекція 4
Оператори циклу
План

1. Цикл for.
2. Варіанти циклу for.
3. Нескінченний цикл.
4. Порожній цикл for.
5. Цикл while.
6. Цикл do-while.
7. Оголошення змінних в умовних операторах і циклах
8. Оператори переходу:
   - return
   - goto
   - break
   - continue

У мові C/C++, як і у всіх інших сучасних мовах програмування, оператори циклу призначені для виконання повторюваних інструкцій, поки діє певна умова. Ця умова може бути як задана заздалегідь (у циклі **for),** так і мінятися під час виконання циклу (в операторах **while** й **do-while**).

Цикл for

У тому або іншому виді цикл **for** є у всіх процедурних мовах програмування. Однак у мові C/C++ він забезпечує особливо високу гнучкість і ефективність.

Загальний вид оператора **for** такий.

for *(ініціалізація; умова; збільшення)*

Цикл **for** має багато варіантів. Однак найбільш загальна форма цього оператора працює в такий спосіб. Спочатку виконується *ініціалізація* (initialization) — оператор присвоювання, що задає початкове значення лічильника циклу. Потім перевіряється *умова* (condition), що представляє собою умовний вираз. Цикл виконується доти, поки значення цього виразу залишається істинним. *Збільшення* (increment) змінює значення лічильника циклу при черговому його виконанні. Ці розділи оператора відокремлюються один від одного крапкою з комою. Як тільки умова циклу стане хибною, програма припинить його виконання й перейде до наступного оператора.

У наступному прикладі цикл **for** виводить на екран числа від 1 до 100.

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

int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	int  x;
	for (x = 1; x <= 100; x++)   cout << x;

	cout << endl;
	system("pause");
	return 0;
}
```

Спочатку змінній *x* присвоюється число 1, а потім вона порівнюється із числом 100. Оскільки її значення менше 100, виконується команда cout<<. Потім змінна x збільшується на одиницю, і умова циклу перевіряється знову. Як тільки її значення перевищить число 100, виконання циклу припиниться. У цьому випадку змінна *x* є лічильником циклу, що змінюється й перевіряється на кожній ітерації.

Розглянемо приклад циклу for, тіло якого складається з декількох операторів.

```
for (x = 100; x != 65; x -= 5)
      {
            z = x * x; cout << "Квадрат числа”<<x<<”дорівнює”<<z
      }
```

Піднесення числа *x* у квадрат і виклик команди cout<< виконуються доти, поки значення змінної *x* не стане рівним 65. Зверніть увагу на те, що в цьому циклі лічильник зменшується: спочатку йому привласнюється число 100, а потім на кожній ітерації з нього віднімається число 5.

У циклі **for** перевірка умови виконується перед кожною ітерацією. Іншими словами, якщо умова циклу із самого початку є помилковою, його тіло не буде виконано жодного разу. Розглянемо приклад.

```
int y;
      int x = 10; for (y = 10; y != x; ++y)  cout<<y;

      cout << y; /*   Це  єдиний стейтмент що виконується*/
```

Цей цикл ніколи не буде виконаний, оскільки значення змінних x та y при вході в цикл рівні. Отже, умова циклу є помилковим, і ні тіло циклу, ні збільшення лічильника виконуватися не будуть. Таким чином, значення змінної в залишиться рівним 10, і саме воно буде виведено на екран.

Варіанти циклу for

Найпоширенішим є варіант, у якому використається оператор послідовного виконання ("кома"), що дозволяє застосовувати кілька лічильників циклу одночасно. Наприклад, змінні x і y є лічильниками наведеного нижче циклу. Їхня ініціалізація виконується в тому самому розділі циклу.

```
int x, y;
      for (x = 0, y = 0; x + y < 10; ++x)
      {
      y = getchar(); y = y - '0';   /*  Відняти  зі  змінної в ASCII-код нуля   */
      }
```

Як бачимо, два оператори ініціалізації розділені комою. При кожній ітерації значення змінної x збільшується на одиницю, а змінна y вводиться із клавіатури. Незважаючи на це, змінна y повинна мати якесь початкове значення, інакше перед першою ітерацією циклу умова може виявитися помилковою.

Функція converge(), наведена нижче, демонструє одночасне застосування декількох лічильників циклу. Вона копіює один рядок в інший, переміщаючись від кінців до середини.

```
#include<iostream>
#include"windows.h"
using namespace std;
void converge(char* targ, const char *src);
int main()
{
      SetConsoleCP(1251);
```

```
	SetConsoleOutputCP(1251);


		char target[80] = "XXXXXXXXXXXXXXXXXXXXXXXXXXXX"; // "Перевірка функції
converge()."
		converge(target, "Перевірка функції converge().");
		cout << "Результат:" << target << endl;


	cout << endl;
	system("pause");
	return 0;
}

/* Ця функція копіює один рядок в інший, переміщаючись від кінців до середини. */
void converge(char* targ, const char *src)
{
	int i, j;
	cout << targ << endl;
	for (i = 0, j = strlen(src); i <= j; i++, j--) {
		targ[i] = src[i];
		targ[j] = src[j];
		cout << targ << endl;
	}
}
```

Програма виводить на екран наступні рядки.

```
D:\c++ projects\Project5\x64\Debug\Project5.exe
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
ПXXXXXXXXXXXXXXXXXXXXXXXXXXX
ПеXXXXXXXXXXXXXXXXXXXXXXXXXX.
ПерXXXXXXXXXXXXXXXXXXXXXXXX).
ПереXXXXXXXXXXXXXXXXXXXXXX().
ПеревXXXXXXXXXXXXXXXXXXXe().
ПеревіXXXXXXXXXXXXXXXXXge().
ПеревірXXXXXXXXXXXXXXXrge().
ПеревіркXXXXXXXXXXXXXerge().
ПеревіркаXXXXXXXXXXXverge().
Перевірка XXXXXXXXXnverge().
Перевірка фXXXXXXXonverge().
Перевірка фуXXXXXXconverge().
Перевірка фунXXXX converge().
Перевірка функXXї converge().
Перевірка функції converge().
Результат:Перевірка функції converge().

Press any key to continue . . .
```

У функції **converge** () для індексації рядка з обох кінців використаються два лічильники циклу **for** — змінні і й j. При виконанні циклу лічильник і збільшується, а лічильник j зменшується. Виконання циклу припиняється, коли значення лічильника і стає більше, ніж значення лічильника j. У цей момент всі символи рядка вже скопійовані.

Умовне вираження не обов'язково пов'язане з перевіркою лічильника циклу. Як умова циклу може використатися будь-який припустимий оператор

порівняння або логічний оператор. Це дозволяє задавати кілька умов циклу одночасно.

Розглянемо функцію, що перевіряє пароль користувача. Для уведення пароля користувач може зробити три спроби. Виконання циклу припиняється, якщо користувач вичерпав всі спроби або ввів правильний пароль.

```
void sign_on()
{
	char str[20];
	int x;
	for (x = 0; x < 3 && strcmp(str, "пароль"); ++x) {
		cout << "Уведіть пароль: ";
		gets_s(str);
	}
	if (x == 3)  return;
	/*  Інакше користувач  одержує доступ до системи . . . */
    }
```

У цій функції використовується стандартна функція **strcmp()**, що порівнює два рядки й повертає нуль, якщо вони збігаються.

Нагадаємо, що кожний із трьох розділів циклу **for** може складатися з будь-яких припустимих виразів. Ці вирази можуть бути ніяк не пов'язані із призначенням розділів. З огляду на вищесказане, розглянемо наступний приклад.

```
#include<iostream>
#include"windows.h"
using namespace std;
int sqrnum(int num);
int readnum(void);
int prompt(void);

int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	int t;
	for (prompt(); t = readnum(); prompt())
		sqrnum(t);

	cout << endl;
	system("pause");
	return 0;
}

int prompt(void)
{
	cout<<"Уведіть число: ";
	return 0;
}
int readnum(void)
{
	int t;

	cin>>t;
	return t;
}
int sqrnum(int num)
{
	cout<<num * num<<endl;
	return num * num;
}
```

Зверніть увагу на цикл **for** у функції **main()**. Кожний з його розділів містить виклик функції, у якій користувачеві пропонується ввести із клавіатури якесь число. Якщо уведено число 0, виконання циклу припиняється, оскільки умовне

вираження стає помилковим. У протилежному випадку число зводиться у квадрат. Таким чином, даний цикл **for** використовує розділи ініціалізації й збільшення вкрай незвичайно, при цьому й із синтаксичної, і з семантичної точки зору цикл є абсолютно правильним.

Інша цікава особливість циклу **for** полягає в тім, що його розділи можна пропускати. Кожний з його розділів є необов'язковим. Наприклад, цикл, наведений нижче, виконується доти, поки користувач не введе число **123:**

```
for (x = 0; x != 123; )   cin>>x;
```

Зверніть увагу на те, що розділ збільшення лічильника в даному циклі **for** відсутній. Це значить, що при кожній ітерації значення змінної *x* рівняється із числом **123,** і ніякі дії з нею більше не виконуються. Однак, якщо користувач уведе із клавіатури число **123,** умова циклу стане помилковим, і програма припинить його виконання.

Лічильник можна ініціалізовувати за межами циклу **for.** Цим способом користуються, коли початкове значення лічильника є результатом складних обчислень, як у наступному прикладі.

```
gets(s);  /* Вважати рядок як змінну s */
if (*s) x = strlen(s); /* Обчислити довжину рядка */
else x = 10;

for (; x < 10; ) {
      cout<<x;
      ++x;
    }
```

Тут розділ ініціалізації залишений порожнім, а змінна *x* ініціалізується до входу в цикл.

Нескінченний цикл

Хоча в якості нескінченного можна використати будь-який цикл, традиційно для цієї мети застосовується оператор **for.** Оскільки всі розділи оператора **for** є необов'язковими, його легко зробити нескінченним, не задавши ніякого умовного виразу.

for(; ;) printf("Цей цикл виконується нескінченно.\n");

```
for (; ; )   cout<<"Цей  цикл  виконується  нескінченно."<<endl;
```

Якщо умовний вираз не зазначений, він вважається істинним. Зрозуміло, у цьому випадку можна як і раніше виконувати ініціалізацію й збільшення лічильника, однак програмісти мовою C++ як нескінченний цикл найчастіше використають конструкцію **for** (; ;).

Насправді конструкція **for(;;)** не гарантує нескінченне виконання циклу, оскільки його тіло може містити оператор **break,** що приводить до негайного виходу. (Ми детально вивчимо цей оператор небагато пізніше.) У цьому випадку програма передасть керування наступному операторові, що перебуває за межами тіла циклу **for,** як показано нижче.

```
ch = '\0';
for (; ; ) {
      ch = getchar(f);   /*  Ввести символ  */
      if (ch == 'A')   break;   /*  Вихід із циклу  */
}
```

```
cout<<"Ви  ввели  букву A";
```

Цей цикл виконується доти, поки користувач не введе із клавіатури букву А.

Порожній цикл for

Оператор може бути порожнім. Це значить, що тіло циклу **for** (як і будь-якого іншого циклу) може не містити жодного оператора. Цей факт можна використати для підвищення ефективності деяких алгоритмів і затримки виконання програми.

Видалення пробілів із вхідного потоку – одне з найпоширеніших завдань. Наприклад, система керування базою даних може допускати запит "показати всі рахунки, залишок на яких менше 400". База даних розпізнає кожне слово окремо, не з огляду на пробіли. Отже, вона розпізнає слово "показати", але не зрозуміє слова " показати". Таким чином, пробіли в рядку запиту необхідно ігнорувати. Це завдання вирішує цикл **for**, що пропускає всі пробіли, що коштують перед словами в рядку str.

```
for (; *str == '   '; str++)  ;
```

Як бачимо, цей цикл не має тіла – воно йому не потрібно.

Цикли часто використаються для затримки виконання програми. Нижче показано, як цього можна досягти, використовуючи оператор **for**.

```
for (t = 0; t < SOME_VALUE; t++);
```

Цикл while

Другий по значимості цикл у мові C/C++ - оператор **while**. Він має такий вигляд.

while *(умова) оператор;*

Тут *оператор* може бути порожнім, окремим оператором або блоком операторів. *Умова* може задаватися будь-яким виразом. Умова циклу вважається істиною, якщо значення цього виразу не дорівнює нулю. Як тільки умова циклу стає хибною, програма передає керування операторові, що знаходиться відразу після оператора while. Розглянемо приклад функції, що обробляє уведення із клавіатури, що терпляче чекає, поки користувач не введе букву А.

```
char wait_for_char()
{
      char ch;

      ch = ' \0';      /*   початкове  значення  змінної  ch  */
      while (ch != 'A')   ch = getchar(t);
      return  ch;
}
```

Спочатку змінній **ch** привласнюється нуль. Оскільки вона є локальною, її значення поза функцією **wait_for__char()** не визначено. Потім цикл **while** перевіряє, чи не дорівнює значення змінної **ch** символу А. Оскільки початкове значення змінної **ch** дорівнює нулю, умова циклу є щирим, і його виконання триває. Умова циклу перевіряється щораз, коли користувач уводить символ із клавіатури. Як тільки він уведе букву А, умова циклу стане помилковим, і програма припинить його виконання.

Як і цикл **for**, цикл **while** перевіряє умову перед початком виконання тіла. Отже, якщо умова циклу із самого початку є хибною, його тіло не буде виконано

жодного разу. Завдяки цій властивості немає необхідності окремо перевіряти умову циклу перед входом у нього. Цю особливість добре ілюструє функція **pad()**, що додає пробіли в кінець рядка, поки її довжина не стане рівною заданій. Якщо довжина рядка вже дорівнює необхідній величині, пробіли не додаються.

```
#include<iostream>
#include"windows.h"
using namespace std;
void pad(char* s, int length);

int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	char str[80];
	strcpy(str, "Перевірка");
	pad(str, 40);
	cout<<strlen(str);


	cout << endl;
	system("pause");
	return 0;
}

/* Додає пробіли в кінець рядка. */
void pad(char* s, int length)
{
	int l;
	l = strlen(s); /* Обчислюємо довжину рядка */
	while (l < length) {
		s[l] = ' '; /* Вставляємо пробіл */
		l++;
	}
	s[l] = '\0';   /*   Рядок   повинна   завершуватися   нульовим   байтом   */
}
```

Функція **pad()** має два аргументи - покажчик **s** на довжину рядка і цілочисельну змінну **length**, що задає необхідну довжину рядка. Якщо вихідна довжина рядка дорівнює числу **length** або перевищує його, то тіло циклу **while** не виконується. Якщо ж довжина рядка, на яку посилається покажчик **s**, менше необхідної, функція **pad()** додає потрібну кількість пробілів. Довжина рядка обчислюється за допомогою стандартної функції **strlen()**.

Якщо вихід із циклу **while** залежить від декількох умов, звичайно як умова циклу використають окрему змінну, значення якого змінюється декількома операторами в різних місцях циклу. Розглянемо приклад.

```
void func1()
{
	int working;

	working = 1; /* тобто, істина */

	while (working) {
		working = processl();
		if (working)
			working = process2();
		if (working)
			working = process3();
	}
    }
```

Кожна функція, викликана в цьому фрагменті програми, може повернути помилкове значення й привести до виходу із циклу.

Тіло циклу **while** може бути порожнім. Наприклад, наведений нижче порожній цикл виконується доти, поки користувач не введе букву A.

```
while((ch=getchar())  !=  'A')  ;
```

Оператор присвоювання усередині умовного вираження циклу **while** виглядає незвично, але все стане на свої місця, якщо згадати, що його значенням є значення операнда, розташованого в правій частині.

Цикл do-while

На відміну від операторів **for** й **while**, які перевіряють умови на початку циклу, оператор **do-while** робить це наприкінці. Іншими словами, цикл **do-while** виконується принаймні один раз. Загальний вид оператора **do-while** такий.

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

Якщо тіло циклу складається лише з одного оператора, фігурні дужки не обов'язкові, хоча вони роблять цей цикл зрозуміліше (програмістові, але не компіляторові). Цикл **do-while** повторюється доти, поки *умова* не стане помилковим.

Розглянемо приклад, у якому цикл **do-while** зчитує числа із клавіатури, поки вони не перевищать 100.

```
do {
      cin>>num;
    } while (num > 100);
```

Найчастіше оператор **do-while** застосовується для вибору пунктів меню. Коли користувач вводить припустиме значення, воно повертається відповідною функцією. Неприпустимі значення ігноруються. Розглянемо поліпшену версію програми для перевірки правопису.

```
void menu(void)
{
       char ch;
       cout<<"1.   Перевірка правопису"<<endl;
       cout << "2.   Виправлення помилок" << endl;
       cout << "3.   Вивід помилок " << endl;
       cout << "      Виберіть пункт меню: ";

       do {
              ch = getchar(f); /* Зчитуємо символ із клавіатури */
              switch (ch) {
              case '1':
                     check_spelling();
                     break;
              case '2':
                     correct_errors();
                     break;
              case '3':
                     display_errors();
                     break;
```

```
        }
    } while (ch != 'l' && ch != '2' && ch != '3');
}
```

Тут оператор **do-while** дуже зручний, оскільки функція повинна вивести меню принаймні один раз. Якщо користувач увів припустиме значення, програма продовжить виконувати цикл.

Оголошення змінних в умовних операторах і циклах

У мові C++ (але не в стандарті C89) можна повідомляти змінні усередині умовних виражень, що входять в оператори **if** або **switch**, усередині умови циклу **while**, а також у розділі ініціалізації циклу **for**. Область видимості змінної, оголошеної в одному із цих трьох місць, обмежена блоком, що ставиться до даного оператора. Наприклад, змінна, оголошена усередині циклу **for**, є локальною стосовно нього.

Розглянемо приклад, у якому змінна оголошена в розділі ініціалізації циклу **for**.

/* Змінна i є локальною стосовно циклу for, а змінна j існує як усередині, так і поза циклом. */

```
int  j;
for (int i = 0; i < 10; i++)
    j = i * i;
/* i  =  10;  //  ***  Помилка *** -  змінна  i  тут  невидима! */
```

У цьому прикладі змінна і оголошена усередині циклу **for** і використається як лічильник. Поза циклом **for** ця змінна не існує!

Оскільки лічильник потрібний лише усередині циклу, його оголошення в розділі ініціалізації оператора **for** стало загальноприйнятою практикою.

Якщо ваш компілятор повністю підтримує стандарт мови C++, то змінні можна повідомляти не тільки усередині циклів, але й усередині звичайних умовних виражень. Розглянемо приклад.

```
if (int x = 20) {
    x = x - y;
    if (x > 10)   y = 0;
}
```

Тут оголошується змінна x, якій привласнюється число 20. Оскільки це значення вважається істинним, виконується блок, пов'язаний з оператором **if**. Область видимості змінної, оголошеної усередині умовного вираження, обмежується цим блоком. Таким чином, поза оператором **if** змінної *x* не існує. Чесно говорячи, не всі програмісти вважають прийнятним такий спосіб оголошення змінних, тому ми не будемо його застосовувати.

Оператори переходу

У мові C/C++ передбачені чотири оператори безумовного переходу: **return, goto, break, continue.** Оператори **return** й **goto** можна застосовувати в будь-якому місці програми, у той час як оператори **break** й **continue** пов'язані з

операторами циклів. Крім того, оператор **break** можна застосовувати усередині оператора **switch**.

Оператор return

Цей оператор використається для повернення керування з функції. Він ставиться до операторів безумовного переходу (jump operators), оскільки виконує повернення в точку виклику функції. З ним може бути зв'язане певне значення, хоча це й не обов'язково. Якщо оператор **return** пов'язаний з певним значенням, воно стає результатом функції. У стандарті С89 всі функції, що повертають значення, формально не зобов'язані були містити оператор **return.** Якщо програміст забував указати його, функція повертала якесь випадкове значення, так зване "сміття". Однак у мові С++ (і в стандарті С99) всі функції, що повертають значення, *зобов'язані* містити оператор **return.** Іншими словами, якщо функція в мові С++ повертає якесь значення, будь-який оператор **return,** що з'являється в її тілі, повинен бути пов'язаний з якимсь значенням.

Оператор **return** має такий вигляд.

return *вираз;*

Вираз вказується лише тоді, коли у відповідності зі своїм оголошенням функція повертає якесь значення. У цьому випадку результатом функції є значення даного *виразу*.

Усередині функції можна використати скільки завгодно операторів **return.** Однак функція припинить свої обчислення, як тільки досягне першого оператора **return.** Закриваюча фігурна дужка, що обмежує тіло функції, також приводить до припинення її виконання. Вона інтерпретується як оператор **return,** не зв'язаний ні з яким значенням. Якщо програміст не вкаже оператор **return** у функції, що повертає якесь значення, то її результат залишиться невизначеним.

Функція, визначена специфікатором **void,** може не містити жодного оператора **return,** пов'язаного з яким-небудь значенням. Оскільки такі функції по визначенню не мають значень, що повертають, безглуздо зв'язувати з ними оператор **return.**

Оператор goto

Оскільки в мові С/С++ існує багатий вибір керуючих структур на основі операторів **break** й **continue,** в операторі **goto** немає особливої необхідності. Вважається, що використання операторів **goto** знижує читабельність програм. Проте, незважаючи на те, що оператор **goto** протягом багатьох років піддається критиці, він як і раніше іноді застосовується. Важко уявити собі ситуацію, до якої не можна було б обійтися без оператора **goto.** І все-таки у вмілих руках і при обережному використанні оператор **goto** може принести користь, наприклад, при виході із глибоко вкладених циклів.

Для оператора **goto** необхідна *мітка* (label), що представляє собою ідентифікатор із двокрапкою. Мітка повинна перебувати в тій же функції, що й оператор goto, - перестрибувати з функції у функцію не можна. Оператор goto має такий вигляд.

goto *мітка:*

.

.

мітка;

Мітка може перебувати як до, так і після оператора goto. Наприклад, використовуючи оператор goto і мітку, можна організувати цикл від 1 до 100.

```
int x = 1;
loopl:
x++;
    if (x < 100)   goto   loopl;
```

Оператор break

Оператор **break** застосовується у двох ситуаціях. По-перше, він використовується для припинення виконання **case** усередині оператора **switch**. По-друге, за допомогою оператора **break** можна негайно вийти із циклу незалежно від істинності або хибності його умови.

Якщо в циклі зустрічається оператор **break**, ітерації припиняються й виконання програми відновляється з оператора, що випливає за оператором циклу. Розглянемо приклад.

```
#include<iostream>
#include"windows.h"
using namespace std;
void pad(char* s, int length);

int main()
{
      SetConsoleCP(1251);
      SetConsoleOutputCP(1251);

      int t;
      for (t = 0; t < 100; t++) {
            cout<< t;
            if (t == 10) break;
      }



      cout << endl;
      system("pause");
      return 0;
   }
```

Ця програма виводить на екран числа від 0 до 10, а потім цикл припиняється, оскільки виконується оператор **break**. Умова t < 100 при цьому ігнорується.

Оператор **break** часто застосовується в циклах, виконання яких варто негайно припинити при настанні певної події. У наведеному нижче прикладі натискання клавіші припиняє виконання програми:

```
#include<iostream>

#include"windows.h"
using namespace std;


int main()
{
      SetConsoleCP(1251);
      SetConsoleOutputCP(1251);

      do {
            /* пошук імен ... */
            if (getchar()) break;
      } while (1);
      /* точка виходу */
```

```
    cout << endl;
    system("pause");
    return 0;
}
```

Якщо клавіша натиснута, функція `getchar()` повертає код клавіши, і в такому випадку умова буде істина.

Якщо цикли вкладені один у одного, оператор **break** виконує вихід із внутрішнього циклу в зовнішній. Наприклад, наведена нижче програма 100 разів виводить на екран числа від 1 до 10, причому щораз, коли лічильник досягає значення 10, оператор **break** передає керування зовнішньому циклу **for**.

```
for (t = 0; t < 100; ++t) {
            count = 1;
            for (;;) {
                  cout<<count;
                  count++;
                  if (count == 10) break;
            }
    }
```

Якщо оператор **break** утримується усередині оператора **switch**, що вкладений у якийсь цикл, то вихід буде здійснений тільки з оператора **switch**, а керування залишиться в зовнішньому циклі.

Функція exit

Хоча функція **exit()** не ставиться до керуючих операторів, прийшов час її вивчити. Виклик стандартної бібліотечної функції **exit()** приводить до припинення роботи програми й передачі керування операційній системі. Її ефект можна зрівняти з катапультуванням із програми.

Функція **exit()** виглядає в такий спосіб.

void exit(int *код_повернення);*

Значення змінної *код_повернення* передається процесу, у ролі якого частіше за все виступає операційна система. Нульове значення коду повернення відповідає нормальному завершенню роботи. Інші значення аргументу вказують на вид помилки. Як код повернення можна застосовувати макроси **EXIT_SUCCESS** й **EXIT_FAILURE**. Для виклику функції **exit()** необхідний заголовний файл **stdlib.h**. У програмах мовою C++ можна також використати заголовний файл **<cstdlib>**.

Функція **exit()** часто використовується, коли обов'язкова умова, що гарантує правильну роботу програми, не виконується. Розглянемо, наприклад, віртуальну комп'ютерну гру, для якої потрібно спеціальний графічний адаптер. Функція **main()** у цій програмі може виглядати в такий спосіб.

```
if (!virtual_graphics()) exit(1);
      play;
      /* ... */
```

Тут функція **virtual_graphics ()** визначається користувачем і повертає істинне значення, коли в комп'ютері є необхідний графічний адаптер. Якщо ж його нема, вона повертає помилкове значення, і функція **exit()** припиняє роботу програми.

У програмі для перевірки правопису функція **menu()** може використати функцію **exit()** для виходу із програми й повернення керування операційній системі.

```
char ch;
      cout << "1.   Перевірка правопису" << endl;
```

```
	cout << "2.   Виправлення помилок" << endl;
	cout << "3.   Вивід помилок " << endl;
	cout << "4.   Вихід" << endl;
	cout << "      Виберіть пункт меню: ";

	do {
		ch = getchar(); /* Введення символу із клавіатури */
		switch (ch) {
		case '1':
			check_spelling();
			break;
		case ' 2 ':
			correct_errors();
			break;
		case '3':
			display_errors();
			break;
		case '4':
			exit(0); /* Повернення в операційну систему */
		}
	} while (ch != 'l' && ch != '2' && ch!='3');
```

Оператор continue

Оператор **continue** нагадує оператор **break**. Вони розрізняються тим, що оператор **break** припиняє виконання всього циклу, а оператор **continue** - лише його поточної ітерації, викликаючи перехід до наступної ітерації й пропускаючи всі оператори, що залишилися, у тілі циклу. У циклі **for** оператор **continue** викликає перевірку умови й збільшення лічильника циклу. У циклах **while** й **do-while** оператор **continue** передає керування операторам, що входять в умову циклу. Наведена нижче програма підраховує кількість пробілів у рядку, уведеної користувачем.

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


int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	char s[80], * str;
	int space;

	cout<<"Введіть рядок: ";
	gets_s(s);
	str = s;

	for (space = 0; *str; str++) {
		if (*str != ' ') continue;
		space++;
	}
	cout<<"пробілів"<<space<<endl;

	cout << endl;
	system("pause");
	return 0;
    }
```

Перевіряється кожен символ рядка. Якщо він не є пробілом, оператор **continue** перериває поточну ітерацію циклу **for** і починає нову, у протилежному випадку значення лічильника **space** збільшується на 1.

У наступному прикладі оператор **continue** виконує вихід із циклу **while,** передаючи керування операторові, що входить в умову циклу.

```
void code(void)
{
      char  done, ch;
      done = 0;
      while (!done) {
            ch = getchar();
            if (ch == '$') {
                  done = 1;
                  continue;
            }
            putchar(ch + 1);   /*   Перейти   до   наступної   букви   алфавіту   */
      }
   }
```

Функція **code** кодує повідомлення, додаючи одиницю до коду кожного символу. Наприклад, у повідомлені англійською мовою буква A замінюється буквою B и т. Функція припиняє свою роботу, якщо користувач увів символ $. Після цього вивід повідомлень на екран припиняється, оскільки змінна **done** приймає щире значення, і, відповідно, умова циклу стає помилковим.

Використана література

1. Г. Шилдт, Полный справочник по C++, 4-е видання, в-во «Вильямс», 2006
2. Х.М.Дейтел, Как програмировать на C++, 4-е видання, в-во «Бином-Пресс» 2009.
3. Р. Лафоре, Объектно-ориентированное программирование в C++, в-во «Питер», 2004, с 924.