Лекція 3. Інструкції мови С++

Послідовні та галужені алгоритми.
Інструкції if, if else, switch, break. Задача min(a,b,c). Задача "Ми знайшли k грибів".
Повторювані алгоритми. Інструкції циклу for, while, do while. Оголошення в інструкціях.
Оператор ?:. Задача про кількість цифр числа.
Введення послідовностей чисел (стан потоку введення). Інструкції break, continue, goto.
Модифікатор const, тип перелік.

Послідовні та розгалужені алгоритми

Приклад послідовного алгоритму – програма *second.cpp*. Інструкції цієї програми завжди виконуються одна за одною, послідовно, незалежно від уведених вхідних даних. Послідовні ділянки зазвичай є частинами більших складніших алгоритмів. Послідовність інструкцій не потребує спеціального синтаксичного оформлення – достатньо записати інструкції одна за одною. Проте, якщо послідовність є частиною чогось більшого, то її виділяють за допомогою фігурних дужок і утворюють *блок* (інструкцій).

Розгалужений алгоритм містить принаймні одну ділянку, яка може бути виконана або ні, залежно від істинності деякого твердження. Зазвичай розгалужений алгоритм містить дві або більше альтернативні інструкції (чи блоки інструкцій). Галуження в програмах мовою С++ задають за допомогою інструкції *if*.

Різновиди інструкції галуження *if*

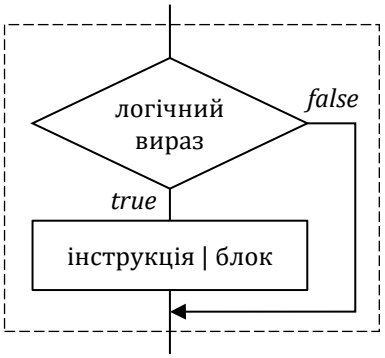
Вкорочена

```
if (логічний вираз) інструкція
```

або

```
if (логічний вираз)
{
   блок
}
```

алгоритм виконання



Приклад 1. Задано два цілих числа. Розташувати їх в змінних *a* і *b* так, щоб в *a* було більше з них.

```
// фрагмент коду
// введення заданих величин
cout << "Input a, b: ";
int a, b;
cin >> a >> b;
if (a < b)
{
      // порядок значень неправильний - поміняємо їх місцями
      // для обміну використаємо третю змінну
      int c = a;
      a = b;
      b = c;
}
cout << "a = " << a << "\tb = " << b << endl;
```

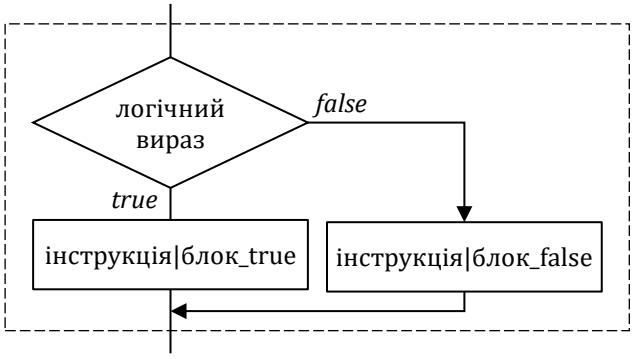
Повна

```
if (логічний вираз) інструкція_true
else інструкція_false
```

або

```
if (логічний вираз)
{
   блок_true
}
else
{
   блок_false
}
```

алгоритм виконання



Приклад 2. Задано два цілих числа. Знайти значення більшого з них.

```
// введення заданих величин
cout << "Input a, b: ";
int a, b;
cin >> a >> b;
int max;
// щоб знайти більше з двох, достатньо їх порівняти між собою
if (a > b) max = a;
else max = b;
cout << "max(" << a << ';' << b << ")=" << max << endl;
```

Інші різновиди отримують комбінуванням двох попередніх.

Продовжена

Приклад 3. Задано ціле число *x*. Обчисліть sign(*x*), де sign – функція знаку набуває значення –1 для від'ємних чисел, 1 для додатних і 0 для нуля.

```
// введення заданих величин
cout << "Input x: ";
int x;
cin >> x;
int sign;
// послідовна перевірка двох пов'язаних умов
if (x < 0) sign = -1;
else if (x > 0) sign = 1;
else sign = 0;
cout << "x = " << x << "\tsign = " << sign << endl;
```

Вкладена

Приклад 4. Задано номер дня за календарем і його назву в тижні. Визначимо умовний колір дня за правилом: п'ятниця 13-те – «чорний», 13-те будь-який інший день тижня – «червоний», всі інші дні – «зелений».

```
int dayNo;
cout << "Input number of the day: "; cin >> dayNo;
string nameInWeek;
cout << "Input name of the day: "; cin >> nameInWeek;
string color;
if (dayNo == 13)
{
	if (nameInWeek == "Friday") color = "Black";
	else color = "Red";
}
else
	color = "Green";
cout << nameInWeek << ' ' << dayNo << " has color " << color << endl;
```

Приклад 5. Задано три різні цілі числа. Знайдіть значення найменшого з них.

```
cout << "Input a, b, c: ";
int a, b, c;
cin >> a >> b >> c;
int min;
if (a < b)
{
	if (a < c) min = a;
	else min = c;
}
else if (b < c) min = b;
else min = c;
cout << "min = " << min << endl;
```

Інструкція вибору *switch*

Використовують для організації галуження на багато альтернатив замість багатьох продовжень інструкції *if*.

```
switch (вираз_дискримінант)
{
   case константний_вираз1: інструкція1 | інструкції; break;
   case константний_вираз2: інструкція2 | інструкції; break;
   case константний_вираз3: інструкція3 | інструкції; break;
   ...
}
```

або

```
switch (ініціалізація; вираз_дискримінант)
{
   оголошення_локальних_змінних
   case конст_вираз11: case конст_вираз12:
      інструкція1 | інструкції; break;
   case конст_вираз21: case конст_вираз22: case конст_вираз23:
      інструкція2 | інструкції; break;
   case конст_вираз3: інструкція3 | інструкції; break;
   ...
   default: інструкція|інструкції_за_замовчуванням;
}
```

Алгоритм виконання подібний до алгоритму виконання продовженої *if*:

- виконати *ініціалізацію* (якщо є) –будуть створені локальні змінні, потрібні для обчислення виразу-дискримінанта;
- обчислити *вираз-дискримінант* – він має бути цілого типу, або такого, що перетворюється до цілого;
- *оголосити локальні змінні* (якщо є);
- порівняти значення виразу-дискримінанта з *константним_виразом1* (при цьому константний вираз перетворюється до типу дискримінанта)
  - якщо ==, то виконати *інструкцію*1, завершити;
  - у протилежному випадку йти далі
- порівняти значення виразу-дискримінанта з *константним_виразом2*:
  - якщо ==, то виконати *інструкцію*2, завершити;
  - у протилежному випадку йти далі…
- і так далі до першого збігу чи до вичерпання константних виразів
- якщо є альтернатива *default*, то виконати *інструкції за замовчуванням*, завершити.

Приклад 6. Задано ціле число $k \in [1, 99]$. Надрукувати фразу «Ми знайшли *k* грибів», узгоджуючи закінчення слів.
Пояснення. Для значень 1, 21, 31, …, 91 потрібно надрукувати «гриб», для 2, 3, 4, 22, 23, …, 94 – «гриби», для інших значень – «грибів». Щоб не перелічувати занадто велику кількість варіантів, можна аналізувати останню цифру числа *k*, вилучивши попередньо всі «надцятки».

```
SetConsoleOutputCP(1251);
cout << "Уведіть число k між 1 та 99: ";
int k; cin >> k;
cout << "Ми знайшли " << k;
if (k >= 11 && k <= 19) cout << " грибів\n";
else
      switch (k % 10) // остання цифра – це остача від ділення на 10
      {
      case 1: cout << " гриб\n"; break;
      case 2: case 3: case 4: cout << " гриби\n"; break;
      default: cout << " грибів\n";
      }
```

Тернарний оператор ?:

Має три операнди, відокремлених знаками ? та :. Загальна схема виразу

```
(логічний_вираз) ? вираз_true : вираз_false
```

Такий оператор можна використовувати в простих випадках як дієву альтернативу інструкції *if*. Наприклад, обчислення більшого з двох цілих (див. приклад 2) можна за допомогою одної інструкції:

```
int max = (a > b) ? a : b;
```

Дужки не обов'язкові, але бажані заради читабельності. Тернарні оператори можна вкладати один в одного.

Алгоритми з повтореннями

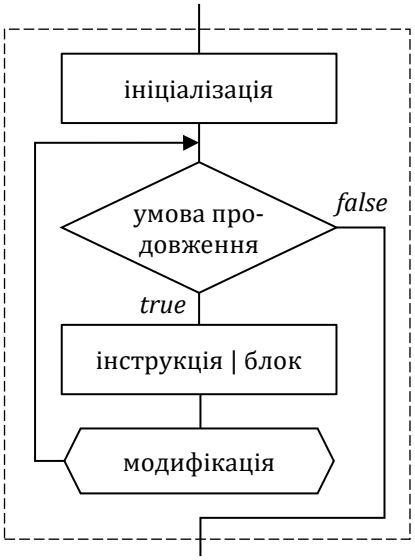
У мові C++ є три різні інструкції для організації циклів. Усі вони взаємозамінні – можна було б обійтися і однією, але кожна з інструкцій має свою традиційну галузь застосування.

Арифметичний цикл

Якщо кількість повторень циклу відома наперед, використовуйте інструкцію *for*.

```
for (ініціалізація; умова_продовження; модифікація)
   тіло_циклу
```

алгоритм виконання



- *ініціалізація* оголошує і налаштовує параметр (або параметри) циклу;
- *умова продовження* – логічний вираз; цикл працюватиме тільки тоді, коли він має значення *true*;
- *модифікація* змінює значення параметра циклу;
- *тіло циклу* – інструкція або блок інструкцій;
- ініціалізацію, умову та модифікацію можна пропустити, якщо вони не потрібні, або визначені в іншому місці.

Приклад 7. Задано натуральне *n*. Обчисліть *n*!

```
cout << "Input n (n > 0): ";
unsigned n; cin >> n;
unsigned long long f = 1ULL;
// факторіал – добуток усіх від 1 до n
for (unsigned i = 2; i <= n; ++i) f *= i;
cout << n << "! = " << f << endl;
```

Приклад 8. Задано натуральне *n* і *n* дійсних чисел. Обчисліть їхню суму.

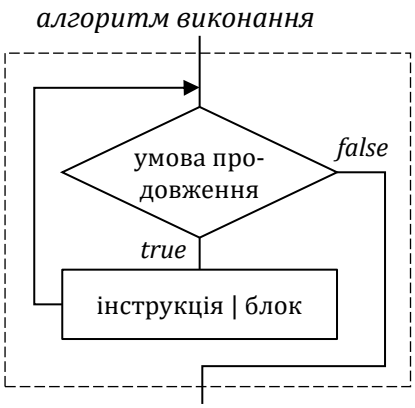
```
cout << "Input n (n > 0): ";
unsigned n; cin >> n;
cout << "Input " << n << " numbers: ";
double total = 0.0; // підсумок
for (unsigned i = 0; i < n; ++i)
{
      double price; // чергове задане число
      cin >> price;
      total += price;
}
cout << "sum = " << total << endl;
```

Приклад 9. Задано натуральне *n*. Обчисліть значення виразу $1\times\sin(n)+2\times\sin(n-1)+\ldots+n\times\sin 1$.

```
cout << "Input n (n > 0): ";
unsigned n; cin >> n;
double expr = 0.0;
// цикл використовує два параметри: зростаючий і спадний
// перевіряти достатньо один з них
for (int i = 1, j = n; i <= n; ++i, --j)
      expr += i * sin(j);
cout << "Expression = " << expr << endl;
```

Приклад 10. Задано натуральне *n*. Обчисліть значення виразу $\sqrt{3+\sqrt{6+...+\sqrt{3(n-1)+\sqrt{3n}}}}$.

```
cout << "Input n (n > 0): ";
int n; cin >> n;
double root = 0.0;
// обчислення почнемо з кінця – з внутрішнього кореня
// крок параметра циклу підібрано відповідно до задачі
for (int i = 3*n; i >= 3; i -= 3)
      root = sqrt(i + root);
cout << "Expression = " << root << endl;
```



Ітераційний цикл з передумовою

Застосовують тоді, коли кількість повторень наперед невідома.

```
while (умова_продовження) тіло_циклу
```

Тут інструкції в тілі циклу або оператори в умові продовження повинні впливати на операнди логічного виразу так, щоб він зрештою набув значення хиба, і цикл припинив виконання. Якщо умова має значення *false* перед початком виконання циклу, то він не виконається жодного разу.

Приклад 11. Задано непорожню послідовність чисел, що закінчується нулем. Обчисліть їхнє середнє арифметичне (нуль не враховувати).

```
double average = 0.0; // сума значень
int count = 0;        // кількість значень
double num;
cout << "Введіть послідовність чисел, що закінчується нулем: ";
cin >> num;           // перше число
while (num != 0.0)
{
      average += num; ++count;
      cin >> num;    // чергове число
}
average /= count;
cout << "Середнє арифметичне " << count << " чисел = " << average << endl;
```

Альтернативна реалізація використовує інструкцію *break* для примусового завершення виконання тіла циклу.

```
double average = 0.0; // сума значень
int count = 0;        // кількість значень
double num;
cout << "Введіть послідовність чисел, що закінчується нулем: ";
while (cin >> num)    // після успішного введення cin приводиться до true
{
      if (num == 0.0) break; // знайшли 0 – завершуємо обчислення
      average += num;
      ++count;
}
average /= count;
cout << "Середнє арифметичне " << count << " чисел = " << average << endl;
```

Інструкція *break* перериває виконання блока, в якому записана. Ми вже бачили її застосування в інструкції *switch*, використання в тілі циклу – ще один поширений випадок. Якщо виникає потреба завершити достроково не весь цикл, а тільки одну поточну ітерацію (і перейти до наступної), використовують інструкцію *continue*, зазвичай, після виконання певної умови.

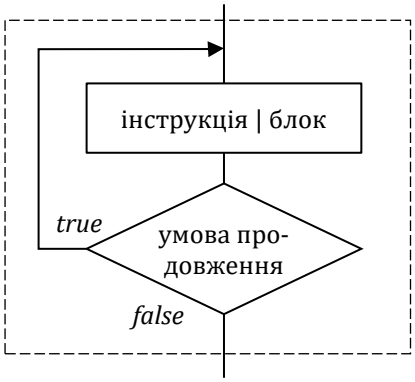
```
// Цикл можна перервати не тільки нулем, а й довільним нечисловим значенням.
// Якщо набрати літеру, спроба введення числа завершиться невдачею, cin «зіпсується»
// і приведення дасть false – цикл завершиться
// Зіпсутий cin потрібно відновити:
      if (cin.fail())
      {
            cin.clear(); // скинути аварійні прапорці
            // вилучити з потоку введення всі символи аж до кінця рядка
            while (cin.get() != '\n') continue;
      }
```

У цій програмі використано інструкції *break* та *continue*. Як і раніше, *break* припиняє виконання блока, тобто перериває виконання циклу опрацювання чисел, як тільки введено нуль. Інструкцію *continue* використовують лише в циклах для того, щоб перейти до наступної ітерації циклу. У нашому прикладі всю роботу в циклі виконує умова продовження: метод *cin.get()* забирає з потоку введення одну літеру, тому тіло циклу порожнє.

Приклад 12. Задано натуральне число. Скільки десяткових цифр є в його записі?
Ідея розв'язування базується на тому, що цілочислове ділення на 10 «забирає» в числа його молодшу цифру.

```
      cout << "Input n (n > 0): ";
      long long n; cin >> n;
      long long copy_n = n; // копія для змістовного виведення
      unsigned digits = 0;  // лічильник цифр
      while (n > 0)
      {
            ++digits;
            n /= 10;
      }
      cout << copy_n << " has " << digits << " digits\n";
```

алгоритм виконання



Ітераційний цикл з постумовою

Відрізняється від попереднього взаємним розташуванням умови продовження і тіла циклу. Тіло буде виконане принаймні один раз.

```
do
   інструкція | блок
while (умова_продовження);
```

Зручно використовувати, якщо перед перевіркою умови потрібно виконати якісь дії.

Приклад 13. Задано дійсне додатне число *x*. Обчисліть $\sqrt[3]{x}$ за допомогою алгоритму Ньютона з точністю 10^{-6}.
Нагадаємо, що цей алгоритм будує послідовність наближених значень y_k за формулами:

$y_0 = 1;\ y_k = \frac{1}{3}\left(\frac{x}{y_{k-1}^2} + 2y_{k-1}\right), k = 1,2,\ldots$ Точності ε досягнуто, якщо $|y_k - y_{k-1}| \le \varepsilon$.

```
      cout << "Введіть додатне x: ";
      double x; cin >> x;
      double y = 1.0; // y_k
      double z;       // y_(k-1)
      do
      {
            z = y;
            y = (x / y / y + 2.*y) / 3.;
      }
      while (abs(y - z) > 1.e-6);
      cout << "Корінь кубічний з " << x << " = " << y << '\n';
```

Усі описані вище інструкції керування мають спільну властивість: один вхід і один вихід. Такі структури керування використовують для побудови добре структурованих програм з ясними шляхами виконання, або для *структурного програмування*.

Інструкція безумовного переходу

У окремих випадках може виявитися корисною інструкція передавання керування у визначену точку програми: інструкція безумовного переходу. Інструкцію, якій передають керування, позначають окремим іменем – позначкою – відокремленим від інструкції двокрапкою.

```
goto позначка; // позначка – ім’я (ідентифікатор)
позначка: інструкція
```

За допомогою багатьох інструкцій *goto* легко написати заплутану програму, шляхи виконання якої годі буде простежити, тому використання *goto* не заохочується (а в структурному програмуванні – заборонене). Для зменшення ризиків «заплутаності» рекомендують передавати керування тільки вперед по ходу програми і не далі, ніж на один екран тексту програми.

У програмі мовою C++ інструкція *goto* стане в пригоді для того, щоб швидко вийти з декількох вкладених блоків: декількох вкладених *for* або *switch*. Відомо, що інструкція *break* припиняє виконання блока. З її допомогою можна достроково вийти з циклу, але тільки з одного (у тілі якого вона записана). Щоб припинити виконання одразу двох (або більше) вкладених циклів, можна використати *goto ім’я_зовні_циклів;*.

Приклад 14. Знайдіть найменше трицифрове число, що має парну кількість десятків не меншу за 2 і ділиться на 67.
Один з підходів до побудови розв’язку – перебрати допустимі цифри, скласти з них число і перевірити, чи ділиться воно на 67. Перебір можна припинити, як тільки знайдеться перше таке, що ділиться.

```
int main()
{
	SetConsoleOutputCP(1251);
	// перебираємо сотні, десятки, одиниці
	for (int hundreds = 1; hundreds <= 9; ++hundreds)
	{
		for (int tens = 2; tens <= 8; tens += 2)
		{
			for (int ones = 0; ones <= 9; ++ones)
			{
				// конструюємо шукане число
				int number = hundreds * 100 + tens * 10 + ones;
				if (number % 67 == 0)
				{
					cout << "Шукане число " << number << endl;
					goto out_of_loops;
				}
			}
		}
	}
	cout << "Потрібного числа не знайшли";
out_of_loops:
	system("pause");
	return 0;
}
```

Література

1. Бублик В.В. Об'єктно-орієнтоване програмування мовою C++.
2. Стивен Прата Язык программирования C++.
3. Дудзяний І.М. Програмування мовою C++.

4. Брюс Эккель, Чак Эллисон Философия C++.
5. Бьерн Страуструп Язык программирования C++.
6. Герберт Шилдт C++ Базовый курс.