Операції в С++

Проф. Куссуль Н.М.

Вступ

- В мові C++ доступ до **объектів** і **функцій** забезпечується виразами
- **Вирази** складаються з **операцій** і **операторів**
  - **Операція –** дія, що виконується над об’єктом даних (загальне поняття)
  - **Оператор –** мінімальний виконуваний фрагмент програмного коду (поняття мови програмування)

Операції

- арифметичні
- логічні
- відношення
- побітові

Арифметичні операції

- Арифметичні операції застосовують до будь-яких комбінацій простих типів (bool, int, char, double):
  - \+ (плюс, унарний і бінарний)
  - \- (мінус, унарний і бінарний)
  - * (множення)
  - / (ділення)
  - % (остача від ділення)

Приклади арифметичних операцій

```
cout << "1 + 2 = " << 1 + 2 << endl;
cout << "5 - 3 = "  << 5 - 3 << endl;
cout << "4 * 6 = " << 4 * 6 << endl;
cout << "8 / 4 = "  << 8 / 4 << endl;
cout << "7 % 4 = " << 7 / 4 << endl;
```

Операції відношення

- Операції відношення застосовують до будь-яких комбінацій простих типів :
  - = = (перевірка на рівність)
  - ! = (не дорівнює)
  - > (більше)
  - < (менше)
  - < = (менше або дорівнює)
  - > = (більше або дорівнює)

Приклади операцій відношення

```
(a == 10) // Якщо два значення рівні
(a != b)  // Якщо два значення не рівні
    //між собою
(a > 5)   // Якщо перше значення
    //більше за друге
(b <= 30) // Якщо перше значення
    //менше або дорівнює другому
```

Логічні операції

- Логічні операції застосовні до будь-яких виразів:
  - ! (заперечення)
  - && (логічне І)
  - || (логічне АБО)

Приорітет і порядок виконання операцій

Пріоритет	Знаки операцій	Назви операцій	Порядок Виконання
1.	. ->[] () ++ --	вибір елемента за іменем вибір елемента за вказівником вибір елемента за індексом виклик функції або конструювання значення постфіксний інкремент постфіксний декремент	зліва-направо
2.	sizeof ++ -- ~ ! + - & new delete (ім'я_типу)	розмір операнда в байтах префіксний інкремент префіксний декремент інверсія (порозрядне заперечення) логічне заперечення унарний плюс унарний мінус адреса виділення пам'яті або створення звільнення пам'яті або знищення перетворення типу	справа-наліво
3.	.* ->*	вибір елемента по імені через вказівник вибір елементу по вказівнику через вказівник	зліва-направо

Приорітет і порядок виконання операцій

4.	* / %	множення ділення остача від ділення цілих (ділення по модулю)	зліва-направо
5.	+ -	додавання віднімання	зліва-направо
6.	<< >>	зсув вліво зсув вправо	зліва-направо
7.	< > <= >=	менше більше менше або дорівнює більше або дорівнює	зліва-направо
8.	== !==	дорівнює не рівне	зліва-направо

Приорітет і порядок виконання операцій

9.	&	порозрядне І	зліва-направо
10.	^	порозрядне виключаюче АБО	зліва-направо
11.	|	порозрядне АБО	зліва-направо
12.	&&	логічне І	зліва-направо
13.	||	логічне АБО	зліва-направо
14.	?:	умовна операція	справа-наліво
15.	= *=, /=, %=, +=, -=, &=, ^=, |=	присвоювання (просте і складене)	справа-наліво
16.	*throw*	генерація виключення	справа-наліво
17.	,	послідовність виразів	зліва-направо

Вираз з пріоритетом операцій

Розглянемо вираз

a =(1+2)*(2+3);

С ++ обчислює даний вираз наступним чином:

a = (1 + 2) * (2 + 3)= (3) * (2 + 3)=
= 3 * (5)= 3*5 = 15;

Побітові операції

- **Операції зсуву:**
  - << (зсув вліво на 1 розряд (перевантажений для выводу))
  - >> (зсув вправо на 1 розряд (перевантажений для вводу))
- **Побітові (порозрядні) операції:**
  - & (порозрядна кон'юнкція бітовых представлень значень цілочисельних операндів)
  - | (порозрядна диз'юнкція бітовых представлень значень цілочисельних операндів)
  - ^ (порозрядна виключаюча диз'юнкція бітовых представлень значень цілочисельних операндів)
  - ~ (операція інвертування або побітового заперечення)
  - **Для порівняння:**
    - ! (операція логічного заперечення)

Інші операції

- Для збільшення (зменшення) значення змінної в С++ використовують такі операції:
- Операції **автоінкременування**:
  - a++;
  - ++a;
- Операції **автодекременування** :
  - a--;
  - --a;
- **Приклад.** Операції автоінкременування і автодекременування
  - j =++i; //рівносильно i =i+1; j=i; (префіксна форма)
  - j =i++; //рівносильно j =i; i= i+1; (постфіксна форма)

Тернарна операція

- Операція з трьома операндами :
  - **<вираз 1> ? <вираз 2> : <вираз 3>**
  - 1) Обчислити вираз 1.
  - 2) Якщо **вираз 1** = true, тоді обчислюється значення **вираз 2**, яке і стає значенням всього виразу.
  - 3) Якщо **вираз 1** = false, тоді обчислюється **вираз 3**. Його результат стає результатом всього виразу.
- **Приклад**. Операція з трьома операндами

```
x  =(y<z)? y:z;   // x =min (y,z)
```

Оператори в С++

- **Оператор – найменша виконувана частина програми**
- **Оператори присвоєння (передача керування):**
  - Вираз присвоювання включає в себе оператор присвоєння "=".
- **Приклад. Вираз з оператором присвоєння**
  - a = b + 1;
  - y = z = 3.5;
  - a = b + (c=3); //рівносильно c=3 та a = b + c,
- **Оператори присвоєння можуть об'єднуватися з іншими операторами**
  - a+=b; // a=a+b
  - a*=a+b; // a=a*(a+b);

Умовний оператор

- **Умовний оператор** забезпечує виконання або невиконання деякого оператора або групи операторів в залежності від заданої умови
  - **if (умовний вираз) оператор1;**
  - **if (умовний вираз) оператор1;**
    else оператор2;
- **Приклад.** Знайти мінімум з двох чисел x та y

```
if (x<y)
   min=x;
else
   min=y;
cout<<"min="<<min;
```

Оператори switch і break

- Зручним засобом вибору з множини варіантів є оператор **switch**, який має наступну форму запису:

- switch (*вираз*)
  {
  case *константа1*: *оператор1*; break;
  ...
  case *константаN*: *операторN*; break;
  default : *оператор*; break;
  }

Приклад оператора switch

```
switch (rez)
   {
      case 5: cout<<"Оцінка — відмінно.";
      break;
      case 4: cout<<"Оцінка — добре."; break;
      case 3: cout<<"Оцінка — задовільно.";
      break;
      case 2: cout<<"Оцінка — незадовільно.";
      break;
      default: cout << "Невірне значення rez.";
   }
```

Типи операторів циклів

- При виконанні програми часто виникає необхідність неодноразового повторення однотипних обчислень над різними даними.
- Для цих цілей використовують так звані **цикли**.
- **Цикл** представляє собою частину програми, у якій одні й ті самі обчислення реалізуються неодноразово над різними значеннями одних й тих самих змінних (об’єктів).
- Для організації циклів в C++ використовуються наступні три оператора: **while**, **for** і **do-while**

Цикл типу while

- Цикл типу **while** є циклом з передумовою.
- Він використовується у випадку, коли
  - по-перше, не відома точна кількість повторів
  - по-друге, при цьому немає необхідності, щоб цикл неодмінно був виконаний хоча б один раз.
- Синтаксис

while (вираз)
оператор;

Цикл типу while

- В якості виразу зазвичай використовуються умовні вирази.
- В загальному випадку можна використовувати вирази довільного типу.
- На місці оператора може стояти простий оператор або сукупність операторів, об’єднаних у блок дужками **{}**.
- Якщо вираз істинний (не рівний нулю), то тіло циклу виконується один раз, далі вираз перевіряється знову.
- Ітерації (перевірка умови та тіло циклу) виконуються до тих пір, поки вираз не стане хибним (рівним нулю).

Приклад. Цикл while

Потрібно вгадати число 25 за 10 ітерацій

```
int  i=1, rez=1;
   while (i++<=10&&rez!=25) //перевірка
                            //умови перед циклом
   {
      cout << "\n Введіть число:";
      cin >> rez;
   }
   if (i==10)
      cout<<"\nВи не вгадали.";
   else
      cout<<"\nВітаю! Ви вгадали число.";
```

Цикл типу for

- **Цикл типу for** є циклом з параметрами і зазвичай використовується у випадку, коли відома точна кількість повторів обчислень.
- При цьому виконуються три операції:
  - ініціалізація лічильника циклу
  - порівняння його значення з деяким граничним значенням
  - зміна значення лічильника при кожному проходженні тіла циклу.
- Цикл **for** має наступну форму запису:

```
for (вираз1; вираз2; вираз3)
    оператор;
```

Приклад. Цикл for

```
for (i=1, rez=1; i<=T; i++)
      rez=rez*y;
   cout<<"rez="<<rez;
```

Порівняйте, чим відрізняється наступний цикл ?

```
for (i=1, rez=1; i<=T; i++)
      { rez=rez*y;
   cout<<"rez="<<rez;}
```

Цикл типу do-while

- Цикл типу **do-while** є циклом з постумовою і використовується у тих випадках, коли
  - невідома точна кількість повторів,
  - але водночас цикл необхідно виконати не менше одного разу
- Цикл типу **do-while** дуже схожий на цикл типу **while**
- Різниця тільки в тому, що перевірка істинності виразу в циклі **do-while** має місце після виконання тіла циклу

Цикл типу do-while

- Цей цикл має наступну форму запису:

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

- **Приклад**

```
do
   cin >> r;
while (r!=13);
cout << "Ви вгадали число.";
```

Вкладені цикли

- **Вкладеним циклом** називають конструкцію, в якій один цикл виконується всередині другого. Внутрішній цикл виконується повністю під час кожної ітерації зовнішнього циклу.
- **Приклад.** Заповнити екран символами '#'.

```
for ( i=1; i<=25; i++ )
        for ( k=1; k<=80; k++ )
            cout<<'#';
```

Керуючі оператори в циклах

- Виконання оператора **break** призводить до виходу з циклу, в якому він знаходиться, і переходу до наступного за циклом оператора.
- Якщо оператор **break** знаходиться всередині вкладених циклів, то його дія поширюється тільки на той цикл, в якому він безпосередньо заданий.
- **Приклад**. Треба вгадати число з 10 спроб

```
i=1;
while( i++<=10 )
{
    cin>>rez;
    if (rez==15) break;
    cout << "\nПощастить наступного разу.";
}
if ( i!=10 )  cout<<"\nВи вгадали!.";
```

Керуючі оператори в циклах

- Оператор **continue** може використовуватися тільки серед операторів тіла циклу. Цей оператор призводить до переходу до наступної ітерації без завершення поточної.
- **Приклад.** Вводяться числа місяця для обробки. Необхідно здійснити перевірку коректності вводу. Число 31 означає кінець обробки.

```
while (day!=31)
{
    cin >> day;
    if (day<1 || day>31) continue;
    // Обробка числа day
}
```

Оператори new і delete

- **Оператори керування вільною пам'яттю new і delete** виділяють пам’ять з купи (heap) та вивільняють її.
- **Купа** (heap) — область пам’яті для об'єктів, чий термін служби визначається *програмою*.
  - Замінюють бібліотечні функції C malloc(), calloc() і free().
- 3 способи використання оператора **new** :
  - new type_name

```
float* r= new float;
```

  - new(type_name)

```
float* r= new(float);
```

  - new type_name[*expression*] — виділення пам’яті **для масиву**.

```
float*r=new float[20];
```

Приклади

- Оператор new

```
r=new float(5);  // r=5
r=new float[5]; //r — масив з 5 елементів
```

  - new повертає базовий адрес об'єкта; якщо пам'ять недоступна — '0'

- Оператор delete

```
delete r;   // видаляє скаляр
delete [] r; // видаляє масив
```

  - тип значення, що повертається — void.