3. ОПЕРАТОРЫ

В C++ точка с запятой является признаком конца оператора.

3.1. Пустой оператор

Пустой оператор состоит из знака '**;**'. Он используется там, где по правилам языка должен находиться какой-либо оператор, а по логике программы там ничего выполнять не надо.

3.2. Оператор-выражение

Любое выражение, за которым следует знак '**;**', является оператором. Такой оператор называется **оператор-выражение**. Примеры:

```
i ++; a = b + c; c +
= (a <b) ? a: b;
x + y;                    // здесь результат не используется
                          // и будет выдано предупреждение.
```

3.3. Составной оператор

Составной оператор иначе называют блоком. Он представляет фрагмент текста программы, заключенный в фигурные скобки и, как правило, объединяющий несколько операторов. Составной оператор должен использоваться там, где синтаксис языка требует наличия лишь одного оператора, а логика программы – сразу нескольких:

```
{i = 5; c = sin(i □ x); c++;}          // Это блок.
```

3.4. Объявления

В C++ объявления являются операторами языка и могут стоять там, где возможен любой другой оператор C++:

```
s = 0.3; d /= s; int k = 5;
d = s + 2 * k; double f
= s + d; f □= k;
```

3.5. Условный оператор

Имеется две формы условного оператора:

1) **if(выражение) оператор1**
2) **if(выражение) оператор1 else оператор2**

Оператор1 выполняется в случае, если **выражение** принимает ненулевое значение. Если **выражение** принимает значение 0 (или указатель NULL), то выполняется **оператор2**. Примеры:

```
if(a > b) c = a - b; else c = b - a;
if(i < j) i++; else{ j = i - 3; i++;
}
```

При использовании вложенных операторов **if** текущий **else** всегда относится к самому последнему **if**, с которым еще не сопоставлен ни один **else**.

```
void main( ){ int a =
2, b = 7, c = 3; if(a >
b){ if(b < c) c = b;
} else c =
a;
cout << "c =" << c <<".\n"; }
```

Здесь результатом будет вывод строки **c=2**.

Если опустить фигурные скобки в операторе **if**, то программа примет вид void main( ){ int a = 2, b = 7, c = 3; if(a > b) if(b < c) c = b; else c = a;
cout << "c = " << c<< ".\n"; }

Здесь **else** относится ко второму **if**.
В результате выведется строка **c = 3**.

3.6. Оператор выбора *switch*

Этот оператор позволяет передать управление одному из нескольких помеченных метками операторов в зависимости от значения целочисленного выражения. Метки оператора **switch** имеют специальный вид:

case целая_константа**:** Вид

оператора **switch**:

```
switch(целое_выражение){
[объявления]
[case константное_целое_выражение1:]
```

. . .

[case константное_целое_выражение2:]
[операторы]
. . .
[case константное_целое_выражение m:]

. . .

[case константное_целое_выражение n:]
[операторы]
[default:] [операторы]
}

Здесь [ ] означают необязательную часть оператора, а **...** говорит о том, что указанная конструкция может применяться сколько угодно раз. Блок после **switch**( ) называют телом оператора **switch**.

Приведем схему выполнения оператора.

Сначала вычисляется выражение в круглых скобках (назовем его селектором).

Затем вычисленное значение селектора последовательно сравнивается с константным выражением, следующим за **case**.

Если селектор равен какому-либо константному выражению, стоящему за **case**, то управление передается оператору, помеченному соответствующим оператором **case**.

Если селектор не совпадает ни с одной меткой варианта, то управление передается оператору, помеченному словом **default**.

Если **default** отсутствует, то управление передается следующему за **switch** оператору.

Отметим, что после передачи управления по какой-либо одной из меток дальнейшие операторы выполняются подряд. Поэтому если необходимо выполнить только часть из них, нужно позаботиться о выходе из тела оператора **switch**. Это обычно делается с помощью оператора **break**, который осуществляет немедленный выход **switch**.

Пример 1:

```
int i, d;
cout<<"Задайте целое значение  i\n"; cin
>> i;
```

```
switch(i){ case 1: case 2: case 3: cout << " i=" << i <<
"\n"; case 4: cout << "i = " << i <<"  i^2 = " << i □ i
<< "\n"; d=3 □ i - 4; cout << " d = " << d << ".\n";
break;
case 5: cout << "i = 5.\n"; break;
default: cout << "Значение  i меньше 1 или больше 5.\n"; }
```

Если ввести число 2, то будет напечатано

```
i = 2 i = 2
i^2 = 4 d=2.
```

Если **i** равно 4, то будет выведено

```
i = 4  i^2=16 d
= 8.
```

При **i** = 5 будет выведено i

= 5.

При всех остальных значениях **i** будет напечатано

```
Значение  i  меньше 1 или больше 5.
```

Пример 2:

```
char   sign;
int x, y, z;
cout<<"Задайте знак операции  +    -    *    / \n"; cin
>> sign;
cout << "Задайте  x  и  y \n";
cin >> x>> y; switch(sign){
case '+':  z = x + y;            break;
case '-':  z= x - y;             break;
case '*':  z = x * y;            break;
case '/': if( y == 0 ){cout << "Делить на нуль нельзя!\n"; exit(1);}
else  z = x / y;  break; default: cout << "Неизвестная
операция!\n"; exit(1); }
```

Здесь exit(1) – вызов функции, который приводит к немедленному прекращению выполнения программы.

3.7. Оператор цикла *while*

Оператор цикла с **предусловием** имеет вид

while **(выражение) оператор Оператор**

называют телом цикла.

При выполнении такого оператора сначала вычисляется значение **выражения**. Если оно равно 0, то **оператор** не выполняется и управление передается оператору, следующему за ним. Если значение **выражения** отлично от 0, то выполняется **оператор**, затем снова вычисляется **выражение** и т. д.

Возможно, что тело цикла не выполнится ни разу, если **выражение** сразу будет равно 0.

Пример 1: char

c;

while (cin.get(c))cout << c;

В результате выполнения этого фрагмента происходит копирование символов, включая пробельные, из потока **cin** (в данном случае из буфера клавиатуры) в поток **cout** (в данном случае на экран дисплея). Использованная выше функция **get(c)** (член класса) извлекает один символ из входного потока, включая пробельные символы, и возвращает ненулевое значение, пока не будет достигнут конец файла (признак конца файла – ctrl-z).

Пример 2:

while (1){ операторы ... }

Это – бесконечный цикл.

Пример 3: char c;

while ((c = cin.get (c)) == ' ' || c == '\n' || c == '\t');

Этот оператор цикла пропускает при считывании из потока **cin** так называемые пробельные символы. Здесь **get( )** – другая форма функции, считывающей из потока один символ. Она возвращает целое число – код символа, или число -1, если встретился признак конца файла.

3.8. Цикл с постусловием *do-while*

Этот оператор проверяет условие окончания цикла после каждого прохода через его тело, поэтому тело цикла всегда выполняется по крайней мере один раз.

Вид оператора:

do **оператор** while **(выражение)**

Сначала выполняется **оператор**, затем вычисляется выражение, и если оно отлично от нуля, то оператор выполняется снова и т. д.

Если выражение становится равным нулю – цикл завершается.

Такой цикл удобно, например, использовать при проверке вводимых пользователем данных:

```
int input = 0;
int minvalue = 10, maxvalue = 150; do{
cout << "Введите значение input \n"; cin
>> input;
cout << " input = " << input << "\n";
} while( input < minvalue || input > maxvalue
);
```

3.9. Оператор *for*

Этот оператор цикла имеет вид

for(**оператор1 выражение1; выражение2**) **оператор2**

Оператор1 может быть объявлением, пустым оператором или оператором-выражением.

Наиболее распространенным является случай, когда **оператор1** и **выражение2** являются присваиваниями или обращениями к функциям, а **выражение1** – условным выражением. Этот цикл эквивалентен следующей конструкции:

оператор1

while (**выражение1**){ **оператор2 выражение2**; }

Иногда **оператор1** называют инициализатором цикла, а **выражение2** – реинициализатором.

Любая из трех частей может быть опущена, хотя точка с запятой обязательно должна оставаться. Если отсутствует проверка, то есть

выражение1, то считается, что **выражение1** отлично от 0, так что for(; ;){ . . . } – бесконечный цикл и его надо каким-либо образом прервать. Пример

1:

```
int n = 20, s=0;
for( int i = 1; i <= n; i++ ) s+ = i  i;
```

Здесь вычисляется сумма квадратов целых чисел от 1 до 20. Пример

2:

```
double s, sum, den = 0.85, eps = 1e-10; for( s =
1, sum=0; s > eps; s = den ) sum += s;
```

Здесь вычисляется сумма геометрической прогрессии 1, 1×0.85, 1×0.85×0.85 и т. д., пока ее очередной член не станет меньше 10^{-10}.

В следующем примере вычислим и выведем на экран дисплея таблицу функции $y=\sin(x^2)$ для $x\in[0, \pi/2]$ с шагом $\pi/20$.

```
#include <iostream>
#include <cmath>
void main( ){
const double pi_2= 3.14159265358979/2; int
n = 10;
double x0 = 0, xk = pi_2, y;
double h = (xk - x0)/n;  double
xt = xk + h/2;
cout << "    x           y\n" for
(double x = x0; x < xt; x+ = h){  y
= sin(x * x);
cout.width (4);  cout cout.precision(2);
<< x;
cout.width (10);       cout.precision(4);
cout << y << '\n';
}
}
```

Обращение к функции cout.width(**k**) устанавливает ширину поля следующего вывода в **k** позиций, что позволяет произвести выравнивание таблицы. Функция cout.precision(**k**) задает число цифр, выводимых после десятичной точки. Использовать ли циклы while или for – это в основном дело вкуса. Цикл for предпочтительнее там, где имеется простая инициализация и реинициализация, поскольку при этом

управляющие циклом операторы наглядным образом оказываются вместе в начале цикла. Это наиболее очевидно в конструкции

for(i = 0; i < n; i++),

которая применяется для обработки первых n элементов массива. Отметим, что границы цикла могут быть изменены внутри цикла, а переменная i сохраняет свое значение после выхода из цикла, какова бы ни была причина этого выхода.

3.10. Оператор безусловного перехода

Оператор безусловного перехода имеет вид **goto метка**;

Метка – это имя, за которым следует знак '**:**'. Этот оператор передает управление оператору, помеченному указанной меткой. С его помощью удобно выходить сразу из нескольких вложенных циклов:

```
for ( i = 0; i < l; i++) for
( j = 0; j < m; j++) for (
k = 0; k < n; k++){
…
операторы;
…
if( условие ) goto lab; операторы;
} lab:; . .
.
```

С помощью оператора **goto** можно переходить извне в тело блока, если при этом управление не передается через объявления имен, которые присутствуют в этом блоке.

3.11. Оператор *break*

Этот оператор осуществляет выход из тела цикла **for**, **while**, **dowhile** или оператора **switch**, в котором он появился. При этом управление передается на первый оператор после цикла.

Оператор не может обеспечить выход сразу из двух или более вложенных циклов.

3.12. Оператор *continue*

Этот оператор осуществляет переход на точку сразу за последним оператором тела цикла без выхода из цикла, так что дальнейшие итерации в цикле будут продолжаться.

Пример вывода четных чисел:

```
for ( int num = 0; num < 100; num++ ){
if( num % 2 ) continue; cout << num
<< "\n";
}
```

Когда **num** становится нечетным, выражение num % 2 получает значение 1 и выполняется оператор, который передает управление на следующую итерацию цикла **for**, не выполняя вывода.

Отметим, что этот фрагмент программы лучше написать так:

```
for ( int num = 0; num < 100; num += 2 ){
cout << num << "\n"; }
```

3.13 Оператор *return*

Этот оператор завершает выполнение функции, в которой он задан, и возвращает управление в вызывающую функцию.

Управление передается в вызывающую функцию в точку, непосредственно следующую за вызовом.

Если **return** присутствует в функции main( ), то он вызывает прерывание выполнения программы.