1.4. Условный оператор и циклы

1.4.1. Условный оператор if

Условный оператор if позволяет указать операции, которые должны выполниться при соблюдении некоторого условия, либо не выполниться, если это условие неверно.

Синтаксис оператора if в простейшем случае имеет вид:

```
if ( <условие> )
  <команда если верно>
```

Пусть пользователь вводит с консоли два числа, которые потом сравниваются между собой.

```
int a, b;

cin >> a >> b;

if (a == b)
  cout << "equal";
```

Если ввести два одинаковых числа, программа выводит «equal», иначе — ничего не выводит.

Оператор else позволяет указать утверждение, которое будет выполнено в случае, если условие не верно. Оператор else всегда идет в паре с оператором if и имеет следующий синтаксис:

```
if ( <условие> )
  <команда если верно>
else
  <команда если неверно>
```

В результате, программу можно дополнить следующим образом.

```
int a, b;

cin >> a >> b;

if (a == b)
  cout << "equal" << endl;
else
  cout << "not equal" << endl;
```

Если ввести два одинаковых числа, программа выводит «equal», иначе — «not equal».

Если необходимо выполнить больше одной операции при выполнении условия, нужно использовать фигурные скобки:

```
if ( <условие> ) {
  ...
}
```

Например, можно вывести значения чисел: оба значения, если числа различны, и одно, если совпадают.

```
int a, b;

cin >> a >> b;

if (a == b) {
  cout << "equal" << endl;
  cout << a;
}
else {
  cout << "not equal" << endl;
  cout << a << " " << b;
}
```

Здесь endl (end of line) — оператор, который делает перенос строки.

При работе с оператором if следует иметь в виду следующую особенность. Пусть дан такой код:

```
int a = -1;

if (a >= 0)
  if (a > 0)
    cout << "positive";
else
  cout << "negative";
```

Из-за отступов могло показаться, что оператор else относится к внешнему if, а на самом деле в такой записи он относится к внутреннему if. В C++, в отличие от Python, отступы не определяют вложенность. В итоге программа ничего не выводила в консоль.

Если явно расставить скобки, получится:

```
int a = -1;

if (a >= b) {
  if (a > 0)
    cout << "positive";
}
else {
```

```
  cout << "negative";
}
```

В данном случае, как и ожидается, выведено «negative».

Из последнего примера можно сделать вывод, что следует всегда явно расставлять фигурные скобки, даже если выполнить необходимо всего одну команду.

1.5. Цикл while

Цикл while может быть полезен, если необходимо выполнять некоторые условия много раз, пока истинно некоторое условие.

```
while ( <условие> )
  <команда>
```

Пусть пользователь вводит число n. Требуется подсчитать сумму чисел от 1 до n.

```
int n = 5;
int sum = 0;
int i = 1;
while (i <= n) {
  sum += i;
  i += 1;
}
cout << sum;
```

Аналогом цикла while является так называемый цикл do-while, который имеет следующий синтаксис:

```
do {
  <команда>
} while ( <условие> );
```

Следующая программа является интерактивной игрой, в которой пользователь пытается угадать загаданное число.

```
int a = 5;
int b;

do {
  cout << "Guess the number: ";
  cin >> b;
} while (a != b);

cout << "You are right!";
```

1.6. Цикл for

Цикл for используется для перебора набора значений. В качестве набора значений можно использовать некоторые типы контейнеров:

vector

```
vector<int> a = {1, 4, 6, 8, 10};

int sum = 0;
for (auto i : a) {
  sum += i;
}

cout << sum;
```

map

```
map<string, int> b = {{"a", 1}, {"b", 2}, {"c", 3}};

int sum = 0;
string concat;
for (auto i : b) {
  concat += i.first;
  sum += i.second;
}

cout << concat << endl;
cout << sum;
```

string

```
string a = "asdfasdfasdf";

int i = 0;
for (auto c : a) {
  if (c == 'a') {
    cout << i << endl;
  }
  ++i;
}
```

Простой цикл for позволяет создавать цикл с индексом:

```
string a = "asdfasdfasdf";

for (int i = 0; i < a.size(); ++i) {
  if (a[i] == 'a') {
    cout << i << endl;
  }
}
```

С помощью оператора **break** можно прервать выполнение цикла:

```
string a = "sdfasdfasdf";

for (int i = 0; i < a.size(); ++i) {
  if (a[i] == 'a') {
    cout << i << endl;
    break;
  }
}

cout << "Yes";
```

```
3
Yes
```