Курс с++

Царьков Олег

Описание тикета OTSG-3

Управляющие конструкции.

Управляющие конструкции — это $for, if, while$. Они предназначены для управления тем, в каком порядке совершаются команды языка.

• if пишется так:

```
if (condition) {
    ...
    ...
}
```

В скобках пишется условие. Если оно выполняется, то выполняются команды, написанные в фигурных скобках, а если нет — то ничего не выполняется.

Примеры условий:

$a < b$ (a меньше b)
$a == b$ (a равно b)
$a! = b$ (a не равно b)
$a <= b$ (a меньше, либо равно b)
$a >= b$ (a болбше, либо равно b)

Пример программы:

```
int main() {
    int a = 0;
    if (a == 0) {
        std :: cout << "a is zero";
    } else {
        std :: cout << "a is non - zero";
    }
    return 0;
}
```

• while пишется так:

```
while (condition) {
    ...
    ...
}
```

condition — условие, при котором действия внутри тела *while* будут продолжаться. Условие проверяется перед началом каждой итерации, и, если оно перестает быть верным, цикл кончается.

Пример:

```
int main() {
    int a = 5;
    while (a > 0) {
        std :: cout << a;
        a = a - 1;
    }
    return 0;
}
```

выведет на экран числа от 5 до 1.

• for пишется так:

```
for (pre_statement; condition; iteration_statement) {
    ...
    ...
}
```

Здесь *pre_statement* выполняется только один раз в самом начале. Перед каждой итерацей, как и в while, проверяется *condition* и если он не верен, то цикл завершается. Отличие от while лишь то, что можно часть команд тела цикла записать в *iteration_statement*.

Пример:

```
for (int n = 0; n < 10; ++n) {
    std :: cout << n;
}
```

Выведет все числа от 0 до 9-ти. Здесь $++n$ означает $n = n + 1$.

Рекурсивные развертки.

Из одной функции в программе всегда можно вызвать любую другую. При этом запоминается место, откуда вызвана функция, выполняется код функции, затем в место, откуда она вызвана, подставляется возвращаемое значение.

В языке отсутствует правило, запрещающее вызов функции самой из себя.

Пример:

```
int dumb_recursion() {
    return dumb_recursion();
}
```

В данном примере вызов $dumb\_recusrion()$ будет работать до бесконечности, так как программа будет все время запоминать точки возврата и пытаться снова пробежаться по коду функции. Когда память, в которой хранятся точки возврата, исчерпается, программа завершится с ошибкой.

Другой пример:

```
int sum(const int a, const int b) {
    if(b == 0) {
        return a;
    } else {
        return sum(a, b - 1) + 1;
    }
}
```

В данном случае, если вызвать $sum(2, 3)$, то программа попытается это посчитать как $1 + sum(2, 2) = 1 + 1 + sum(2, 1) = 1 + 1 + 1 + sum(2, 0)$. А в случае $sum(2, 0)$ вернется число 2. В итоге выражение свернется обратно и получится 5.

Почему теперь программа не зацикливается до бесконечности? Потому что внутри функции написано “условие выхода из рекурсии“ при $b = 0$. Однако, все равно этот вариант неправильный и может зациклиться.

Конечно, в данном случае лучше просто написать $return\ a + b;$, этот пример носит только теоретический характер

Задания.

1) Написать функцию, считающую факториал числа n с помощью цикла.

2) Исправить ошибку в примере рекурсивного написания $sum(a, b)$ так, чтобы функция работала для любых двух целых a, b.

3) Написать функцию, считающую факториал числа n с помощью рекурсии.

4) Написать функцию, считающую корень из числа

$double\ sqrt(const\ double\ arg)$

Алгоритм вычисления корня следующий: вначале $ans = 1$, а потом нужно повторить много раз $ans = (ans + arg/ans)/2$

5) Написать функцию, возводящую целое число a в целую степень b.

6) Придумать алгоритм для вычисления кубического корня из числа и написать такую функцию.