Курс с++

Царьков Олег

Описание тикета OTSG-2

Команды(директивы) препроцессора.

В языке есть много удобств, когда вместо того, чтобы писать 20 строчек, можно написать какую-то одну, и компилятор(то, что из текста програмы создает бинарный исполняемый файл) автоматически поймет, что имелось ввиду.

Preprocessing — первый шаг компиляции прграммы, когда код программы преобразуется, и все эти строчки, подразумевающие под собой более длинные записи, преобразуются в более длинные записи, которые под ними имелись ввиду.

Все директивы препроцессора пишутся начиная со знака #

Например,

$\#include < cstdlib >$

Означает, что надо весь текст из файла *cstdlib* вставить в данную программу.

Это нужно вот зачем: в файле cstdlib уже написано много чего полезного, чем можно пользоваться. И, чтобы не переписывать этого заново, можно просто его подключить этой командой.

Есть много других директив, но они все редко используются и сейчас не нужны.

Переменные и функции.

Языки делятся на два класса:

• функциональные. В них основную роль играют функции — короткие названия для большой совокупности действий. Если приходится повторять большую совокупность действий несколько раз, то лучше один раз как-то ее назвать, а потом писать это короткое название. Пример такого языка: perl. В нем есть только функции, а все переменные вообще являются строками!

• объектно-ориентированные. В них основную роль играют объекты. Объект(или тип, или переменная) — это совокупность функций, имеющих общую память. Например, банкомат — это объект, у которого есть функции снятия денег, отправления денег на счет, но у них всех есть общая память — количество денег, которое имеется в банке. Например, *python* — объектно-ориентированный язык. В нем нет даже функций, функция в нем — это объекты, значениями которых являются совокупности команд.

Люди отнесли язык $c++$ к категории объектно-ориентированных языков, но это не так, можно целиком писать программы в функциональном стиле, никто не запрещает этого делать(наедине, разумеется, а если кто-то будет смотреть потом этот код, то лучше не надо).

Вначале мы будем иметь дело лишь с простыми объектами: целыми и действительными числами. Чтобы создать такой объект, нужно набрать

$int\ a;$
$double\ b;$

При этом с именем a будет связано целое число, с именем b — действительное.

Теперь к делу. Во-первых, в программе выполняется лишь то, что написано в функции $main$ и больше ничего.

$int\ main()\ \{$
    $return\ 0;$
$\}$

Функция оформляется следующим образом

$type\_name\ function\_name(type\_name1\ var\_name1, type\_name2\ var\_name2, ...)\ \{$
    ...
    ...
    $return\ returned\_value;$
$\}$

Здесь $type\_name$ — тип возвращаемого значения($returned\_value$). Если возвращать значение не нужно, можно поставить тип $void$.

$type\_name1\ var\_name1, type\_name2\ var\_name2$ — аргументы, которые принимает функция, они используются для подсчета каким-либо образом $returned\_value$, либо просто, чтобы совершать какие-то действия с ними.

Вызов функции:

$function\_name(var\_name1, var\_name2, ...);$

Компилятор при этом ищет определение функции, в котором такое же название, такое же количество аргументов таких же типов, далее запускает код, написанный в теле функции, и вставляет вместо, где она вызвана, $returned\_value$.

Пример:

```
#include < iostream >

int add_one(int a) {
    a = a + 1;
    return a;
}

int main() {
    int b = 3;
    std :: cout << add_one(b);
}
```

Программа напечатает на экране 4. Заметим, что неважно, что a и b называются по-разному, потому что все аргументы, передаваемые в функцию, копируются. После работы функции вместо $add\_one(b)$ вставится возвращаемое значение, которое будет 4, и $std :: cout$ выведет его на экран.

Очень важно понимать, что b при этом останется равным тройке, потому что аргумент при передаче в фукнцию скопировался, и функция работает с его копией a, что не влияет на значение b.

$std :: cout$ это уже не функция, а сложный объект, описание которого содержится в $iostream$, поэтому мы подключили этот файл. $<<$ — это операция побитового сдвига для чисел. Она, вообще говоря, делает следующее.

Написано, например $3 << 1$. Она записывает 3 в двоичном виде — 11, потом сдвигает на 1 побитово, получая 110, и получается число 6.

Но в данном примере это вообще не при чем, потому что $std :: cout$ — это не целое число типа int, а объект типа $std :: ostream$, и для него операция $<<$ определена по-другому. Она не делает никакого побитового сдвига, она просто печатает на экран.

Задания.

1) Написать функцию, суммирующую два числа типа int и возвращающую результат суммирования.

2) Написать функцию, прибавляющую к числу 100 и возвращающую ответ.

3) Написать функцию, печатающую сумму двух чисел на экран и ничего не возвращающую.

4) Выше было написано про $std :: cout$ и операцию $<<$. Операция — это тоже функция $operator << (std :: cout, b)$, где b — это то, что выводится на экран. Вопрос — что она возвращает?

Надо учесть, что если написать

$std :: cout << “something“ << “different“;$

То на экран напечатается $something\ different$