Lesson 6

|  |  |
| --- | --- |
| Здравствуйте, коллеги. Я рад приветствовать вас на шестом занятии Основы Языка С. На этом видеоуроке мы всесторонне рассмотрим такое понятие языка С, как функции, и заглянем в ту часть программы, которая называется заголовочный файл.  Начнём с функций. |  |
| Функция - это такая обособленная часть кода, которую можно выполнять любое количество раз. У функций обязательно в таком порядке должны быть описаны тип возвращаемого значения, название, аргументы и так называемое тело, то есть собственно исполняемый код. Рассмотрим более детально функцию main():  - int это тип возвращаемого значения, то есть на том месте, где будет вызвана эта функция, в результате её работы по выполнении оператора return, появится некое целое число. Возвращаемые значения могут быть любых типов. В случае же когда функция не должна возвращать результат своей работы, указывается ключевое слово void (что переводится с английского пустота). То есть на месте вызова функции в результате её выполнения ничего не появится.  - main это название функции. Функция именно с таким названием, написанным с маленькой буквы, является точкой входа в программу. ОС ищет именно эту функцию, когда получает команду на выполнение программы. Названия функций в рамках одной программы не должны повторяться, никаких других ограничений на название функций не накладывается.  Конструкция в круглых скобках - (int argc, char\* argv[]) это аргументы функции. Аргументы функции - это такие переменные, которые создаются при вызове функции и существуют только внутри функции, с их помощью можно передать в функцию какие-то исходные данные и параметры. Аргументы пишутся в круглых скобках сразу после названия функции. В случае если функция не принимает аргументов необходимо поставить после названия пустые круглые скобки.  Весь код, содержащийся в фигурных скобках после аргументов функции – называется телом функции. Это те операторы и команды, которые будут последовательно выполнены при вызове функции. В теле функции мы можем вызывать другие функции, никаких ограничений на написание тела функции язык не накладывает. | #include <stdio.h>  int main (int argc, const char\* argv[]) {  return 0;  }  // void  // void somefunction ()    int main (int argc, const char\* argv[]) {  // more operations here  // somefunction ();  return 0;  } |
| Функции принято разделять на проверяющие, считающие и выводящие, и каждая из вышеописанных функций не должна нести дополнительной нагрузки. То есть, функция не должна знать откуда в программе появились её аргументы, и где будет использован результат её работы. Для примера опишем функцию суммирующую два числа. в качестве аргументов она будет принимать целые числа и возвращать целочисленный результат. Обратите внимание что функция не знает откуда взялись эти числа, мы можем их считать с консоли, можем задать в виде констант или получить в результате работы какой то другой функции.  *Внутри функции main мы вызываем нашу функцию sum суммирующую два числа и передаем в качестве аргументов эти числа.* | int sum(int x, int y) {  int result = x + y;  return result;  }  int main (int argc, const char\* argv[]) {  sum(50, 60);  return 0;  } |
| *Обратите внимание, что в качестве аргументов мы можем передавать константные значения, а также переменные. Значения переменных мы можем получить например из консоли, либо в результате выполнения какой-нибудь другой функции.* | int main (int argc, const char\* argv[]) {  int a;  scanf(“%d, &a”);  sum(50, a);  return 0;  } |
| Как уже было сказано - аргументы - это переменные, которые хранят в себе некоторые параметры вызова функции. Аргументы позволяют использовать одни и те же функции с разными исходными данными. Приглядимся повнимательнее к хорошо знакомой нам функции printf(). Строка, которую мы пишем в круглых скобках в двойных кавычках это и есть аргумент функции. То есть мы знаем, что функция умеет выводить на экран строки. Как именно - нам нет дела, а какие именно строки - мы указываем в качестве аргумента.  *Функция printf() примечательна еще и тем, что она может принимать в себя нефиксированное количество аргументов. Описание работы таких функций, а также их написание выходит далеко за пределы основ языка, нам важно помнить что мы можем это использовать. В аргументе* *функции printf() мы можем написать заполнитель соответствующего типа и вызвать нашу функцию sum.* | printf(“HelloWorld!”);  printf(“%d”, sum(50, a)); |
| Теперь мы без проблем можем оформить уже существующие у нас программы в виде функций. Например, оформим в виде функций программу проверки простоты числа.  *Для этого опишем функцию которая возвращает целое число, назовем ее isPrime, в качестве аргумента она принимает целое число, назовем его number. Найдем из предыдущих занятий программу определения простоты числа и скопируем в тело функции.*  *Внесем небольшие правки, уберем вывод т.к. это проверяющая функция, вывод оставим для основной части. И допишем если делителей 2 то число простое, возвращаем 1. Если же делителей больше – число не простое, возвращаем 0.*  *Такой вывод можно записать и другим способом, return (d == 2) – это выражение в случае истины вернет 1 в случае лжи 0.*  *Можно воспользоваться тернарным оператором, т.е. написать*  *return (d == 2) ? 1 : 0 – если условие в скобках истинно вернется 1, ложно – 0.;*  *Выйти из функции мы можем на любом этапе ее выполнения, например если делителей уже 3, то мы можем вернуть 0.* | int isPrime(int number){  int d = 0, i = 1;  while(i <= number){  if(number % i++ ==0)  d++;  else  continue;  if (d == 3) break;  }  //if (d == 2) return 1;  //else return 0;  return (d == 2)  *//return (d == 2) ? 1 : 0;*  }  if (d == 3) return 0; |
| *Немного подправим вывод, внесем в него вызов функции isPrime и объявим переменную int num, которую будем передавать в качестве аргумента в функцию isPrime.*  *Запустим нашу программу и убедимся что все работает – число 71 действительно является простым.*  Теперь мы можем написать программы любой сложности, содержащие функции isPrime. sum. О том, что мы работаем с консолью, в нашем случае должна знать только функция main, поэтому ввод значений и вывод на экран мы оставим в ней, а подсчёты значений положим в функции. | int main (int argc, const char\* argv[]) {  int a;  scanf(“%d, &a”);  printf(“%d”, sum(50, a));  int num = 71;  printf(“Введенное число %d %s является простым \n”, number, isPrime(num) ? “” : “не”);  return 0;  } |
| Пришло время поговорить про прототипы.  Зачастую возникают ситуации, когда функция не описана до точки входа в программу, или вовсе лежит в другом файле. В этом случае мы должны сообщить компилятору, что такую функцию придётся дополнительно поискать. Для этого необходимо указать всю информацию о функции, кроме её тела. Такой оператор называется прототип функции.  Опишем прототип функции isPrime, описав сигнатуру этой функции. | int isPrime(int number); |
| И пара слов о заголовочных файлах. Заголовочные файлы это мощный инструмент модульной разработки. Мы уже неоднократно видели подключение заголовочного файла stdio.h, давайте посмотрим, что же скрывает и как именно работает эта строка. Обнаружив данный файл на диске мы видим, что в нём содержатся другие подключения библиотек, директивы препроцессора (о которых более подробно мы будем говорить на следующих занятиях) и прототипы функций (например, так часто используемой нами  printf()). (Где-то вот здесь… на 259 строке) | // Здесь показ содержимого stdio.h |
| То есть, написав строку с ключевым словом include, мы попросили компилятор скопировать весь код заголовочного файла в нашу программу. Зная это, мы можем написать свой собственный заголовочный файл. Создадим в папке проекта новый заголовочный файл с каким-нибудь совершенно неожиданным названием “header.h” и скопируем наши сегодняшние  труды в него. Теперь для того чтобы воспользоваться этими функциями из любой программы, которую мы будем писать в дальнейшем, нам всего лишь нужно подключить наш вновь созданный заголовочный файл к нашему проекту. | \* header.h  #ifndef HEADER\_H\_  #define HEADER\_H\_  int isPrime (int number) {  int d = 0, i = 1;  while (i <= number) {  if (number % i++ == 0)  d++;  else  continue;  if (d == 3) return 0;  }  // if(d == 2) return 1; else return 0;  // return (d == 2) ? 1 : 0;  return (d == 2);  }  #endif /\* HEADER\_H\_\*/ |
| Заголовочные файлы, имена которых написаны в треугольных скобках среда разработки будет искать в папке где содержится компилятор, а написанные в двойных кавычках по пути, относительному папки проекта. Так мы можем создать некую общую библиотеку для наших проектов и при необходимости обращаться к ней.  *Скомпилируем наш проект, введем запрашиваемые данные и убедимся что все работает.*  Немного красивее оформим вывод в консоль, добавив текст запроса суммируемого числа, и  поправим сам код программы. Запустим наш проект ещё раз и убедимся что его не стыдно показывать окружающим.  *На этом предлагаю заканчивать занятие посвященное функциям.* | #include <stdio.h>  // copypasted all code from header  #include "header.h"  int sum (int x, int y) {  int result = x + y;  return result;  }  int isPrime(int number);  int main (int argc, const char\* argv[]) {  int a;  printf(“Enter a number”)// добавляется  scanf(“%d, &a”);  printf(“%d\n”, sum(50, a));  int num = 71;  printf(“Введенное число %d %s является простым \n”, number, isPrime(num) ? “” : “не”);  return 0;  } |
| На следующем уроке мы поговорим об указателях, и передаче аргументов в функцию по значению и по ссылке. До новых встреч. |  |