Тема 1.1 Функции

Функции — это отдельные независимые блоки кода, которые выполняют ряд предопределенных команд. В языке программирования Си вы можете использовать как встроенные функции различных библиотек так и функции, которые вы создали сами, то есть свои собственные функции.

Функции обычно требуют объявления прототипа. **Прототип** дает основную **информацию о структуре функции**: он сообщает компилятору, какое значение функция возвращает, как функция будет вызываться, а также то, какие аргументы функции могут быть переданы. Когда я говорю, что функция возвращает значение, я имею в виду, что функция в конце работы вернет некоторое значение, которое можно поместить в переменную. Например, переменная может быть инициализирована значением, которое вернет функция (Слайд 2)

Рассмотрим общий формат для прототипа функций:

где, returnedDataType — тип данных, возвращаемого функцией, значения;

functionName — имя функции

dataType — тип данных параметра функции, это тот же самый тип данных, что и при объявлении переменной

par1 ... parN — параметры функции.

Этот прототип сообщает компилятору, что функция принимает два аргумента, в качестве целых чисел, и что по завершению работы функция вернет целое значение. Обязательно в конце прототипа необходимо добавлять точку с запятой. Без этого символа, компилятор, скорее всего, подумает, что вы пытаетесь написать собственно определения функции.

Время жизни переменной может быть глобальным и локальным. Переменная с глобальным временем жизни характеризуется тем, что в течение всего времени выполнения программы с ней ассоциирована ячейка памяти и значение. Переменной с локальным временем жизни выделяется новая ячейка памяти при каждом входе в блок, в котором она определена или объявлена. Время жизни функции всегда глобально.

Область видимости объекта (переменной или функции) определяет, в каких участках программы допустимо использование имени этого объекта.

Область видимости имени начинается в точке объявления, точнее, сразу после объявителя, но перед инициализатором. Поэтому допускается использование имени в качестве инициализирующего значения для себя самого.

Рекурсия достаточно распространённое явление, которое встречается не только в областях науки, но и в повседневной жизни. Например, эффект Дросте, треугольник Серпинского и т. д. Самый простой вариант увидеть рекурсию – это навести Web-камеру на экран монитора компьютера,

естественно, предварительно её включив. Таким образом, камера будет записывать изображение экрана компьютера, и выводить его же на этот экран, получится что-то вроде замкнутого цикла. В итоге мы будем наблюдать нечто похожее на тоннель.

В программировании рекурсия тесно связана с функциями, точнее именно благодаря функциям в программировании существует такое понятие как рекурсия или рекурсивная функция. Простыми словами, рекурсия – определение части функции (метода) через саму себя, то есть это функция, которая вызывает саму себя, непосредственно (в своём теле) или косвенно (через другую функцию). Типичными рекурсивными задачами являются задачи: нахождения n!, числа Фибоначчи. Такие задачи мы уже решали, но с использованием циклов, то есть итеративно. Вообще говоря, всё то, что решается итеративно можно решить рекурсивно, то есть с использованием рекурсивной функции. Всё решение сводится к решению основного или, как ещё его называют, базового случая. Существует такое понятие как шаг рекурсии или рекурсивный вызов. В случае, когда рекурсивная функция вызывается для решения сложной задачи (не базового случая) выполняется некоторое количество рекурсивных вызовов или шагов, с целью сведения задачи к более простой. И так до тех пор пока не получим базовое решение. Разработаем программу, в которой объявлена рекурсивная функция, вычисляющая n!

В строках 7, 9, 21 объявлен тип данных unsigned long int, так как значение факториала возрастает очень быстро, например уже 10! = 3 628 800. Если не хватит размера типа данных, то в результате мы получим совсем не правильное значение. В коде объявлено больше операторов, чем нужно, для нахождения n!. Это сделано для того, чтобы, отработав, программа показала, что происходит на каждом шаге рекурсивных вызовов. Обратите внимание на выделенные строки кода, строки 23, 24, 28 — это рекурсивное решение n!. Строки 23, 24 являются базовым решением рекурсивной функции, то есть, как только значение в переменной f будет равно 1 или 0 (так как мы знаем, что 1! = 1 и 0! = 1), прекратятся рекурсивные вызовы, и начнут возвращаться значения, для каждого рекурсивного вызова. Когда вернётся значение для первого рекурсивного вызова, программа вернёт значение вычисляемого факториала. В строке 28 функция factorial() вызывает саму себя, но уже её аргумент на единицу меньше. Аргумент каждый раз уменьшается, чтобы достичь частного решения. Результат работы программы (см. Рисунок 1).

По результату работы программы хорошо виден каждый шаг и результат на каждом шаге равен нулю, кроме последнего рекурсивного обращения. Необходимо было вычислить пять факториал. Программа сделала четыре рекурсивных обращения, на пятом обращении был найден базовый случай. И как только программа получила решение базового случая, она порешала предыдущие шаги и вывела общий результат. На рисунке 1 видно всего четыре шага потому, что на пятом шаге было найдено частное решение, что в итоге вернуло конечное решение, т. е. 120. На рисунке 2 показана схема

рекурсивного вычисления 5!. В схеме хорошо видно, что первый результат возвращается, когда достигнуто частное решение, но никак не сразу, после каждого рекурсивного вызова.

Функция atof преобразует строку в значение типа double. Функция сначала отбрасывает пробелы по мере необходимости, до тех пор, пока не будет найден первый символ, отличный от символа пробела. Затем, начиная с этого символа, atof берет столько символов, сколько возможно. То есть, пока литерал в строке напоминает синтаксис чисел с плавающей точкой, функция его считывает и интерпретирует в числовое значение. Остальная часть строки, после последнего допустимого символа игнорируется и никак не влияет на поведение этой функции.

Допустимое число с плавающей точкой формируется функцией atof из следующих символов

Если первая последовательность не пробельных символов в строке string не формируют правильное число с плавающей точкой, или строка string содержит только пробельные символы, то преобразование строки в число не выполняется.

Указатель на функцию. Динамическое связывание

Указатель на функцию - переменная, которая содержит адрес некоторой функции. Соответственно, косвенное обращение по этому указателю представляет собой вызов функции.

Определение указателя на функцию имеет вид: (Слайд 7)

В соответствии с принципом контекстного определения типа данных эту конструкцию следует понимать так: pf - переменная, при косвенном обращении к которой получается функция с соответствующим прототипом, например int_F(int, char*), то есть pf содержит адрес функции или указатель на функцию. Следует обратить внимание на то, что в определении указателя присутствует прототип - указатель ссылается не на произвольную функцию, а только на одну из функций с заданной схемой формальных параметров и результата.

Перед началом работы с указателем его необходимо назначить на соответствующий объект, в данном случае - на функцию. В синтаксисе Си выражение вида &имя_функции имеет смысл - начальный адрес функции или указатель на функцию. Кроме того, по аналогии с именем массива использование имени функции без скобок также интерпретируется как указатель на эту функцию. Указатель может быть инициализирован и при определении. (Слайд7-2)

Естественно, что функция, на которую формируется указатель, должна быть известна транслятору - определена или объявлена как внешняя.

Синтаксис вызова функции по указателю совпадает с синтаксисом ее определения.

При создании консольного приложения в языке программирования C++, автоматически создается строка очень похожая на эту:

int main(int argc, char* argv[]) // параметры функции main()

Эта строка — заголовок главной функции main(), в скобочках объявлены параметры argc и argv. Так вот, если программу запускать через командную строку, то существует возможность передать какую-либо информацию этой программе, для этого и существуют параметры argc и argv[]. Параметр argc имеет тип данных int, и содержит количество параметров, передаваемых в функцию main. Причем argc всегда не меньше 1, даже когда мы не передаем никакой информации, так как первым параметром считается имя функции. Параметр argv[] это массив указателей на строки. Через командную строку можно передать только данные строкового типа. Указатели и строки — это две большие темы, под которые созданы отдельные разделы. Так вот именно через параметр argv[] и передается какая-либо информация. Разработаем программу, которую будем запускать через командную строку Windows, и передавать ей некоторую информацию.

После того как отладили программу, открываем командную строку Windows и перетаскиваем в окно командной строки экзэшник нашей программы, в командной строке отобразится полный путь к программе(но можно прописать путь к программе в ручную), после этого можно нажимать ENTER и программа запустится (см. Рисунок 1 слайда 9).

Так как мы просто запустили программу и не передавали ей никаких аргументов, появилось сообщение Not arguments. На рисунке 2 изображён запуск этой же программы через командную строку, но уже с передачей ей аргумента Open.

Аргументом является слово Open, как видно из рисунка, это слово появилось на экране. Передавать можно несколько параметров сразу, отделяя их между собой запятой. Если необходимо передать параметр состоящий из нескольких слов, то их необходимо взять в двойные кавычки, и тогда эти слова будут считаться как один параметр. Например, на рисунке изображен запуск программы, с передачей ей аргумента, состоящего из двух слов — It work.

А если убрать кавычки. То увидим только слово It. Если не планируется передавать какую-либо информацию при запуске программы, то можно удалить аргументы в функции main(), также можно менять имена данных аргументов. Иногда встречается модификации параметров argc и argv[], но это все зависит от типа создаваемого приложения или от среды разработки.