Учебный Центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

Язык программирования C

В.Г.Тетерин – Microsoft Solution Developer (Visual C++)
teterin@specialist.ru

компьютерного Центр обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

МОДУЛЬ 7

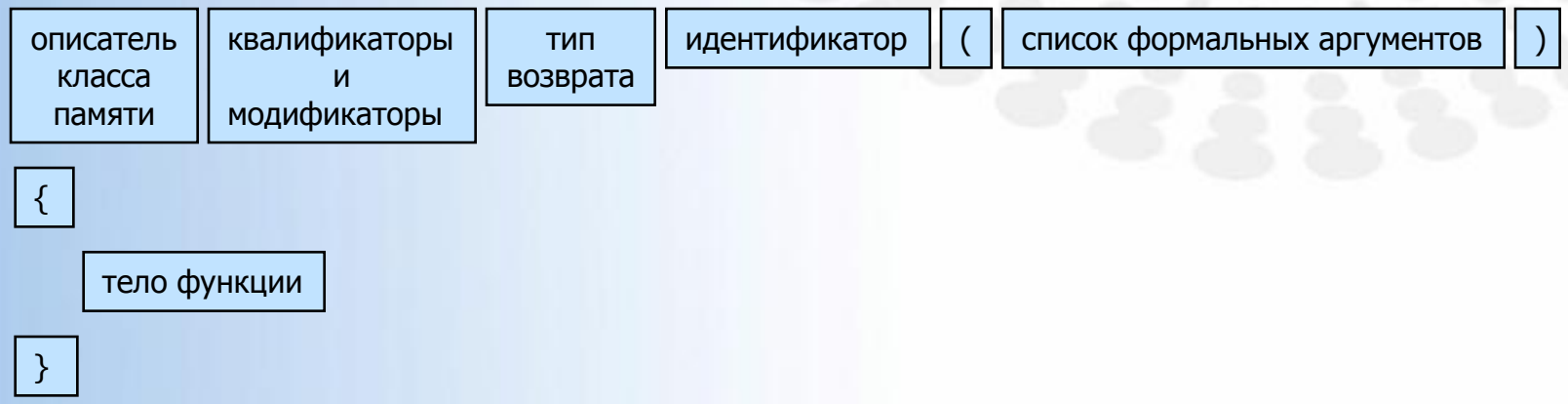
ФУНКЦИИ

Модуль 7. Функции

- Аргументы и параметры
- Прототип функции
- Возвращение значения функцией
- Рекурсия
- Программные проекты в Visual C++

Определение функции

- **Определение функции** состоит из *заголовка* функции и *тела* функции
  - *Заголовок* функции описывает класс памяти функции, тип возвращаемого функцией значения, имя функции и совокупность формальных аргументов.
  - *Тело функции* - это составной оператор (блок), описывающий выполняемые функцией действия.



  - Символ точки с запятой(;) не ставится ни после круглой скобки, заканчивающий список формальных аргументов, ни после фигурной скобки, завершающей тело функции.
  - Ни одна функция не может содержать внутри себя определений других функций - функции в языке С являются ***внешними*** объектами.

Определение функции (продолжение)

- В языке С возвращаемое функцией значение может иметь любой из простых типов:
  - **`char`**, **`int`**, **`float`**, **`double`**, **`указатель`** (рассматривается далее),
  - а также **`структура`** или **`смесь`** (рассматриваются далее).
- Значение функции не может быть составным объектом - *массивом*
  - однако функция может возвращать значение указателя на такой объект.
- В качестве типа возвращаемого значения может быть указан описатель **`void`**
  - это означает, что данная функция *не возвращает* никакого значения, т.е. является *процедурой* в терминологии других языков.
- Если тип возвращаемого значения не указан, то *по умолчанию* считается, что функция возвращает значение типа **`int`**
  - такое использование “**`int`** по умолчанию” стандартом С89 не рекомендуется, а стандартом С99 запрещается.
- Если в качестве списка формальных параметров указан описатель **`void`**, то он сообщает о том, что данная функция *не получает* никаких аргументов.

Определение функции (продолжение)

- *Программа* на языке С обычно состоит из десятков или даже сотен функций.

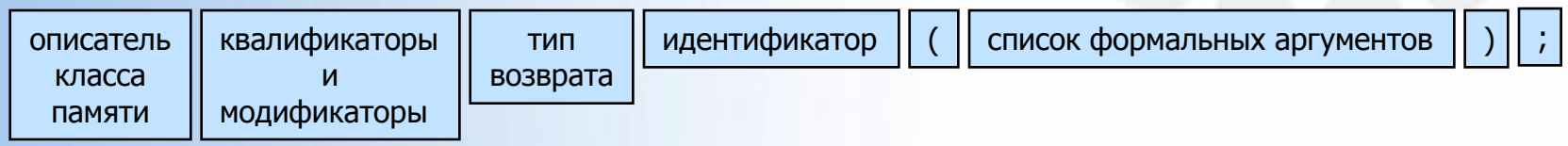
- Ровно одна из функций, составляющих всю программу, должна иметь имя **`main`**:
  - выполнение программы начинается с выполнения функции **`main`**
  - и заканчивается по ее завершению,
  - т.е. функция **`main`** является точкой входа в программу.

- Функция **`main`** вызывает другие функции, те - третьи и т.д.

- Допускается, но *не рекомендуется*, совпадение имени функции, определенной пользователем, с именем библиотечной функции.
  - Такое переопределение библиотечной функции может привести к ошибочному исполнению программы: даже если эта библиотечная функция не используется в программе явно, она иногда может косвенно вызываться другими используемыми библиотечными функциями.

Объявление функции

- **Объявление функции. Прототип функции.**
  - Так как функции в языке Си являются *внешними объектами*, то оператор вызова функции в программе может встречаться *до фактического определения* этой функции.
    - Например, так всегда бывает, когда функция **`main`** располагается в программе первой.
  - Для того, чтобы в этой ситуации обеспечить получение правильного результата, необходимо перед вызовом функции ***объявить ее тип***.
  - **Объявление функции** состоит из *заголовка функции* и заканчивается *точкой с запятой (;)*.



- Например: `double pow(double value, double power);`
- В прототипе можно опустить имена аргументов:

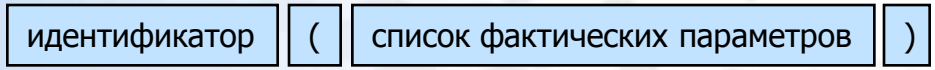
```
double pow(double,double);
```

  но их присутствие улучшает самодокументируемость программы.
- Если функция может принимать переменное число аргументов, то это отмечается в ее заголовке и в прототипе многоточием, расположенным на месте последнего аргумента:

```
int scanf(const char *format, ...);
```

Вызов функции

- **Вызов функции** - это *выражение*, значением которого является результат, возвращаемый функцией.
- В языке С существуют два способа вызова функции:
  - по имени:



  - по указателю: (рассматривается далее)
- Если функция не имеет параметров, то *круглые скобки* все равно должны присутствовать - это отличительный *признак* функции,
  - например, **`getchar()`**.
- Фактические параметры в общем случае могут быть *выражениями*, тогда их значения будут предварительно вычислены перед вызовом функции.
  - например, **`c = sqrt( x*x + y*y )`**.
- Порядок, в котором будут вычисляться выражения-параметры, в языке *не фиксируется*, а оставляется на усмотрение компилятора.
  - Поэтому не рекомендуется использовать конструкции, зависящие от фактического порядка вычисления выражений

Вызов функции (продолжение)

- При вызове функции компилятор выполняет контроль
  - за *количеством* передаваемых параметров
    - их должно быть ровно столько, сколько зафиксировано в прототипе, если только прототип не содержит многоточия - признака переменного числа параметров
  - за *соответствием типов* формальных и фактических параметров
    - при несовпадении типов параметров выполняется преобразование к требуемому типу, если это возможно. Если типы не преобразуемы один в другой, или преобразование кажется компилятору "подозрительным", он выдает соответствующее сообщение.

- Таким образом, применение прототипа для объявления функции позволяет обеспечить полный контроль над ее *корректным использованием* в программе.
  - Особую важность это приобретает при вызове библиотечных или других ранее откомпилированных функций.
  - Часто все прототипы функций (вместе с другими глобальными определениями) собирают в отдельный файл-заголовок (header file) и включают его в каждый программный файл при помощи директивы препроцессора **`#include`**.

Вызов функции (продолжение)

- **Передача параметров по значению**
  - В языке С при вызове функции все ее параметры передаются *"по значению":*
    - вызванная функция получает в промежуточных переменных (фактически в стеке) *копии значений* своих аргументов.
    - если функция изменяет свои аргументы, то все эти изменения касаются лишь значений локальных копий и никак не отражаются на значениях переменных в вызвавшей ее функции.
  - Такое свойство языка обеспечивает разработку функций без *"побочного эффекта"*,
    - то есть вызов функции не может привести к неожиданному изменению переменных в вызывающей функции

- **Передача параметров по адресу**
  - Вызывающая функция может передавать вызываемой функции не значения, а *адреса* своих переменных. В этом случае вызванная функция получает доступ к значениям этих объектов и возможность их изменения (рассматривается далее).

Функции и массивы

- Если аргументом функции является имя массива, то никакого копирования его элементов не производится, а в функцию передается *копия адреса* начала массива.
  - В этом случае, индексируя его, функция получает возможность изменять элементы исходного массива, следовательно, при передаче в функцию массивов возможен "побочный эффект".
- В описании формальных параметров размер массива *не указывается*, поскольку он уже где-то определен.

Пример:

```
/* копирование строки s в t */
void strcopy(char t[], const char s[])
{
    int i;
    for(i=0; t[i]=s[i]; i++);
}
```

- При вызове функции с параметром, являющимся массивом, операция **&** получения адреса к имени массива не применяется, так как имя массива само и есть *адрес* его начала:

```
#define MAXLEN 81
char str1[MAXLEN]="Hello", str2[MAXLEN];
. . . . .
strcopy(str2, str1);
```

Функции и массивы (продолжение)

- В описании формальных параметров размер массива *должен передаваться* отдельным аргументом (если только это не массив типа **`char`**).
- При описании аргументов, являющихся многомерными массивами, так же, как и при их объявлении, можно опускать размер по первому измерению, но размеры по остальным измерениям должны быть указаны, например:

```
#define MAXLEN 100
int func(int array[][MAXLEN], int rows, int columns)
{ /* тело функции */ }
```

- Возможны и более изощренные способы работы функций с массивами. Например, функции может быть передан одномерный массив, а она может трактовать его как многомерный, и наоборот.

Значение, возвращаемое функцией

- Вызванная функция возвращает значение, являющееся результатом ее выполнения, с помощью оператора **return**:

```
/* перевод символа с нижнего на верхний регистр */
char toupper(char c)
{ return (c >= 'a' && c <= 'z') ? c+'A'-'a' : c; }
```

  - Оператор **return** вызывает прекращение выполнения функции и возвращает управление в вызвавшую ее функцию с передачей значения указанного в нем выражения.
  - При необходимости значение выражения автоматически преобразуется к типу, указанному в заголовке функции.

- Если выполнение функции заканчивается оператором **return** без последующего выражения или по достижении правой фигурной скобки, завершающей описание тела функции, то значение функции считается неопределенным (**void**).

- Возвращаемое значение (не **void**) может использоваться в любых выражениях:

```
if(toupper(getchar()) == 'S') goto stop;
```

- Даже если вызванная функция возвращает некоторое значение, вызывающая функция *не обязана* его использовать, и в таком случае оно просто игнорируется:

```
puts("press any key to continue");
getch();
```

Рекурсия

- Язык С допускает *рекурсию* - любая функция, в том числе и **main**, может обращаться сама к себе непосредственно или косвенно.
  - Традиционным примером рекурсии является вычисление факториала:

```
int factorial(int n)
{ return((n>1) ? n*factorial(n-1): 1); }
```

- В случае рекурсивного обращения функции к самой себе при каждом вызове создается (в стеке) *новое множество* всех автоматических переменных и формальных аргументов, и оно совершенно не зависит от предыдущего.
- По выходе из функции это множество значений выталкивается из стека и выполнение продолжается с прерванного места с предыдущим множеством всех локальных значений.
- Рекурсии не предусматривают никаких механизмов *защиты памяти*, так что в случаях слишком "глубоких" рекурсий возможно переполнение стека и порча других элементов данных или даже кода программы.
- Обычно рекурсивные функции работают несколько медленнее своих нерекурсивных версий.
- Однако, рекурсивные программы отличаются компактностью записи и часто их легче писать и понимать, особенно в случае обработки данных с *рекурсивно определенной структурой*, например, деревьев.

Итоги

- В этом модуле Вы изучили:
  - Синтаксис определения и объявления функции
  - Роль объявления (прототипа) функции в программе
  - Семантику вызова функции
  - Понятие рекурсивной функции

Вопросы?

- В.Г.Тетерин – Microsoft Solution Developer (Visual C++)
- teterin@specialist.ru

ПРИЛОЖЕНИЕ

ЗАДАЧИ

Задачи

1. Написать функцию, печатающую строку-вопрос (ее аргумент), принимающую с клавиатуры только ответ “Yes” или “Now” (в форме Y, N или y, n), и возвращающую 1 – при ответе “Yes” и 0 - при “Now”
2. Написать функцию, печатающую запрос на ввод целого числа, принадлежащего диапазону [a,b](ее аргументы) и возвращающую только допустимый ввод пользователя
3. Написать функцию, печатающую запрос на ввод строки, длина которой не превышает заданной длины и возвращающую только допустимый ввод пользователя
4. С помощью функций из п.1-3 ввести фамилии студентов (не более N) и их оценки по M экзаменам

Задачи

5. Написать функцию, вычисляющую расстояние между двумя точками на плоскости, заданными их координатами
6. Написать функцию, вычисляющую площадь треугольника по длинам трех его сторон
7. С помощью функций из п.5-6 вычислить площадь выпуклого N-угольника, заданного его координатами на плоскости
8. Написать функцию, вычисляющую расстояние между двумя точками в пространстве, заданными их координатами
9. С помощью функций из п.7-8 вычислить площадь поверхности треугольной пирамиды

Задачи

10. Написать функцию, вычисляющую скалярное произведение двух N-мерных векторов
11. Написать функцию, меняющую местами две строки матрицы
12. Написать функцию, заменяющую заданную строку матрицы суммой ее с другой строкой, умноженной на заданный коэффициент
13. Написать функцию, определяющую в матрице индекс такой строки, которая содержит в заданном столбце наибольший по модулю элемент, расположенный не выше главной диагонали
14. С помощью функций из п.10-13 решить систему линейных уравнений методом Гаусса с выбором ведущего элемента