Functions

Topics

- Using Predefined Functions
- Programmer-Defined Functions
- Using Input Parameters

Review of Structured Programming

- Structured programming is a problem solving strategy and a programming methodology that includes the following guidelines:
  - The program uses only the sequence, selection, and repetition control structures.
  - The flow of control in the program should be as simple as possible.
  - The construction of a program embodies top-down design.

Review of Top-Down Design

- Involves repeatedly **decomposing** a problem into smaller problems
- Eventually leads to a collection of small problems or tasks each of which can be easily coded
- The **function** construct in C is used to write code for these small, simple problems.

Functions

- A C program is made up of one or more functions, one of which is main( ).
- Execution always begins with main( ), no matter where it is placed in the program. By convention, main( ) is located before all other functions.
- When program control encounters a function name, the function is **called** (**invoked**).
  - Program control passes to the function.
  - The function is executed.
  - Control is passed back to the calling function.

Sample Function Call

```
#include <stdio.h>

 int main ( )
{
      printf (“Hello World!\n”) ;
      return 0 ;
}
```

printf is the name of a **predefined** **function** in the stdio library

this statement is is known as a **function call**

this is a string we are **passing** as an **argument** (**parameter**) to the printf function

Functions (con't)

- We have used three predefined functions so far:
  - printf
  - scanf
  - getchar
- There are 3723 predefined functions on the gl system. Don't worry, nobody knows them all!

Predefined Functions

- You can use the man pages for help with using the predefined functions.
- Some of the functions are also Linux commands or system calls, so to get the information on how to use them in your program, use:

  man 3 abs
- You will also need to use the man page to find out what you have to #include. There are over 900 libraries, so don't expect to learn them all!
- The man page will also tell you what the arguments and return type are.

Man Page Example

ABS(3) Linux Programmer's Manual ABS(3)

NAME

abs, labs, llabs, imaxabs - compute the absolute value of an integer.

SYNOPSIS

#include <stdlib.h>

==========

To use the library function abs( ), you must include the stdlib.h file.

Man Page Example (cont'd)

- In the case of the abs function, you will see:

int abs(int j);

- This means that you must give it one integer argument and it will return an integer value.

Programmer-Defined Functions

- Programmers can write their own functions.
- Typically, each module in a program's design hierarchy chart is implemented as a function.
- C function names follow the same naming rules as C variables.
- All non-trivial programs use functions.
- You define all the parts of a programmer-defined function, the name, the behavior, the parameters (or arguments) and the return type.

Sample Programmer-Defined Function

```
#include <stdio.h>

void printMessage ( void ) ;

int main ( void )
{
     printMessage ( ) ;
     return 0 ;
}

void printMessage ( void )
{
     printf (“A message for you:\n\n”) ;
     printf (“Have a nice day!\n”) ;
}
```

Examining printMessage

```
#include <stdio.h>

void printMessage ( void ) ;

int main ( void )
{
    printMessage ( ) ;
    return 0 ;
}

void printMessage ( void )
{
    printf ("A message for you:\n\n") ;
    printf ("Have a nice day!\n") ;
}
```

function **prototype**

function call

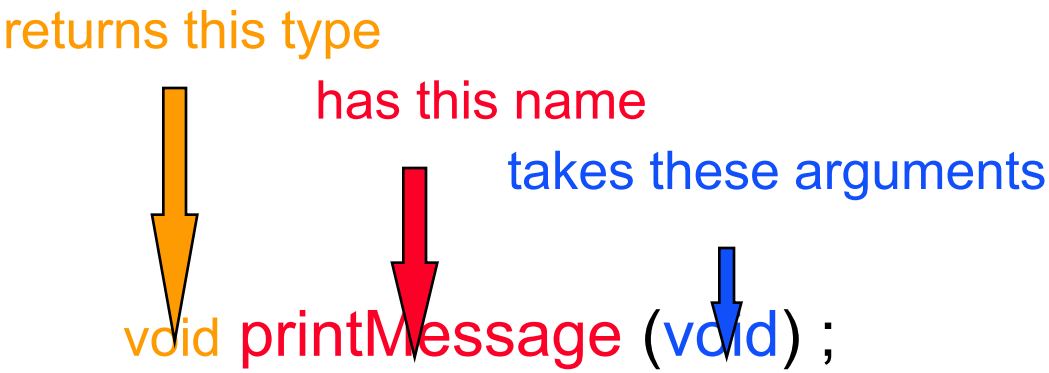
function **header**

function **body**

function **definition**

The Function Prototype

- Informs the compiler that there will be a function defined later that:



- Needed because the function call is made before the definition -- the compiler uses it to see if the call is made properly

The Function Call

- Passes program control to the function
- Must match the prototype in name, number of arguments, and types of arguments

```
void printMessage (void) ;

int main ( void ) same name      no arguments
{
        printMessage ( ) ;
           return 0 ;
}
```

The Function Definition

- Control is passed to the function by the function call. The statements within the function body will then be executed.

```
void printMessage ( void )
{
        printf (“A message for you:\n\n”) ;
        printf (“Have a nice day!\n”) ;
}
```

- After the statements in the function have completed, control is passed back to the **calling function**, in this case main( ) . Note that the calling function does not have to be main( ) .

General Function Definition Syntax

type functionName (*parameter*$_1$, . . . , *parameter*$_n$)
{
    variable declaration(s)
    statement(s)
}

- If there are no parameters, either
  functionName( ) OR functionName(void)
  is acceptable.
- There may be no variable declarations.
- If the **function type** (**return type**) is void, a return statement is not required, but the following are permitted:
  return ; OR return( ) ;

Possibilities

- A function may:
  - Have no arguments and return nothing.
  - Have arguments and return nothing.
  - Have arguments and return one value.
  - Have no arguments and return one value.
- A function can never return more than one value!

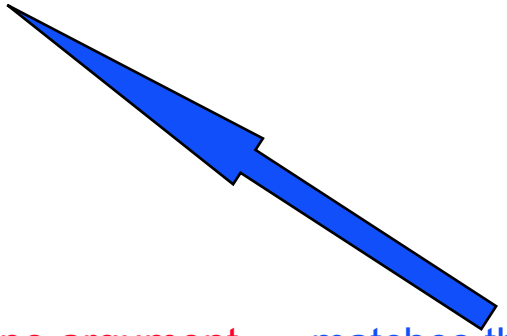
Parameter List

- A function can have more than one parameter:
  int functionA(int a, int b, float c)
- When the function is invoked, the parameters can be a variable, constant, or expression:
  result = functionA(7, 3 + a, dollars);
- The value 7 is put into to local variable a, 3 + a is put into local variable b, and a copy of the value in dollars is put into c in this example.

Using Parameters

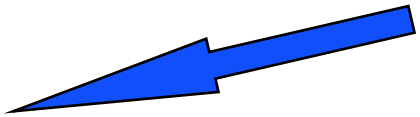
```
void printMessage (int counter) ;
int main ( void )
{
      int num;
      printf (“Enter an integer: “) ;
      scanf (“%d”, &num) ;
      printMessage (num) ;
      return 0 ;
}

void printMessage (int counter)
{
      int i ;
      for ( i = 0; i < counter; i++ )
      {
          printf (“Have a nice day!\n”) ;
      }
}
```



one argument of type int

matches the one **formal parameter** of type int



Using Parameters (cont'd)

- In this example, we do not know in advance what the user will enter for the value of num, however, it is copied into the variable **counter** in the function printMessage( ).
- Both the variables **counter** and **i** are considered to be local variable, because they are only visible inside (or local to) the function printMessage( ).

Final “Clean” C Code

```
#include <stdio.h>

void printMessage (int counter) ;

int main ( void )
{
    int num ;      /* number of times to print message */

    printf (“Enter an integer: “) ;
    scanf (“%d”, &num) ;
    printMessage (num) ;

     return 0 ;
}
```

Final “Clean” C Code (con’t)

```
/*****************************************************************
** printMessage - prints a message a specified number of times
** Inputs:  counter - the number of times the message will be
**                    printed
** Outputs:  None
*****************************************************************/
void printMessage ( int counter )
{
    int i ;   /* loop counter */

    for ( i = 0; i < counter; i++ )
    {
        printf (“Have a nice day!\n”) ;
    }
}
```

Good Programming Practice

- Notice the **function header comment** before the definition of function printMessage.
- This is a good practice and is required by the 104 C Coding Standards.
- Your header comments should be neatly formatted and contain the following information:
  - function name
  - function description (what it does)
  - a list of any input parameters and their meanings
  - a list of any output parameters and their meanings
  - a description of any special conditions

Header Files

- Header files contain function prototypes for all of the functions found in the specified library.
- They also contain definitions of constants and data types used in that library.

Commonly Used Header Files

Header File	**Contains Function Prototypes for:**
<stdio.h>	standard input/output library functions and information used by them
<math.h>	math library functions
<stdlib.h>	conversion of numbers to text, text to numbers, memory allocation, random numbers, and other utility functions
<time.h>	manipulating the time and date
<ctype.h>	functions that test characters for certain properties and that can convert case
<string.h>	functions that manipulate character strings
others	see Chapter 5 of text

Using Header Files

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main ( )
{
    float side1, side2, hypotenuse ;
    printf("Enter the lengths of the right triangle sides:  ") ;
    scanf("%f%f", &side1, &side2) ;
    if ( (side1 <= 0) || (side2 <= 0) {
        exit (1) ;
    }
    hypotenuse = sqrt ( (side1 * side1) + (side2 * side2) ) ;
    printf("The hypotenuse = %f\n", hypotenuse) ;
    return 0 ;
}
```