2008/11/05: Lecture 15
CMSC 104, Section 0101
John Y. Park

Functions: Part 1 of 3

Functions, Part 1 of 3

Topics

- Review of C program
- Functions - Overview
- Using Predefined Functions
- Programmer-Defined Functions
- Using Input Parameters
- Function Header Comments

Review

```
 int main()
{
        float  radius;        /* input    - radius of a circle */
        float  area;          /* output - area of a circle */
        float  circum;        /* output - circumference of a circle */

        /* Ask the user to input the radius */
        printf("Enter radius > ");
        scanf("%f", &radius);

        /* Calculate the circumference */
        area = PI * radius * radius;

        /* Calculate the circumference */
        circum = 2 * PI * radius;

        /* Display the area and circumference */
        printf("The area is %.4f \n", area);
        printf("The circumference is %.4f \n", circum);

        return(0);
}
```

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.

Recipes

- Each recipe has the same types of inputs and outputs as a meal
  - Raw food is the input data
  - The steps to follow is the code
  - The dish is the output
- Most meals have more than one part, so they have more than one recipe
  - 1 for main dish
  - 1 for each side
  - 1 for dessert

A Function is Like a Recipe

- A function has the same parts as a recipe
  - The parameters are the input
  - Executable steps implement the logic
  - The return value is the output
- An application is broken into many functions
  - Each function implements a small, logical amount of work
  - Functions are meant to be reused

Functions

- C programs are made up of one or more functions
  - The main( ) function is required in all C programs
- Execution always begins with main( )
  - 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 to the next statement after the call.

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** to the printf () function

Functions (con't)

- We have used two predefined functions so far:
  - printf
  - scanf
- Programmers can write their own functions.

What Makes up a Function?

- Name
- Input - Parameters
- Output - Value
- Statements

What Makes up a Function?

- Function Name
  - Should describe what the function does
    - Good: print_date, send_email
    - Bad: dostuff, reallylongnamethatsaysnothingandishardtoremember
  - Can contain only letters, numbers, and underscores
    - Start with a lowercase letter
    - "words" in the name are separated by underscores

Function Input

- Function Input (parameter list)
  - The data that the function will need to do its job
  - Parameters are listed inside the parentheses - to the right of the function name
  - Each parameter is specified by its **data type** and a **name**
  - Multiple parameters are separated by a comma
  - If there are no parameters, you can either leave the list blank or write **void**

Function Input

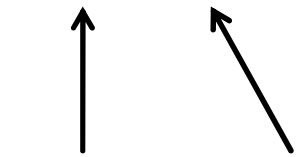
- Examples:

print_newline(**void**);

print_newline();

go(**double miles**);

add_these(**int num1, float num2**);



Parameter Type

Name

Function Output

- Function output (Return value)
  - Each function can return one piece of data
  - Specified by a **data type** to the left of the function name
  - Typically, the return value is the result of the operation or indicates success/failure
    - If there is nothing to return, the return value is **void**
  - The value is returned (and the function exited) using the **return** keyword

Function Output

- Examples:

```
int generate_int();
double guess_weight();
void print_hello();
```

Anatomy of a Function

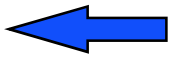
```
#include <stdio.h>

void PrintMessage ( int  numdays) ;

int   main ( )
{
     PrintMessage ( 5 ) ;
     return 0 ;
}

void PrintMessage ( int   numdays)
{
     printf ("A message for you:\n\n") ;
     printf ("Have a nice %d days! \n", numdays) ;

     return;
}
```



function prototype

function call

function definition

A Function Is Implemented in Two Parts

```
#include <stdio.h>

void PrintMessage (int numdays) ;

int main ( )
{
     PrintMessage ( 5 ) ;
     return 0 ;
}
```

function prototype definition

function definition

```
void PrintMessage (int numdays)
{
     printf (“A message for you: \n”) ;
     printf ((“Have a nice %d days! \n”, numdays) ;
}
```

function **header** (same as proto)

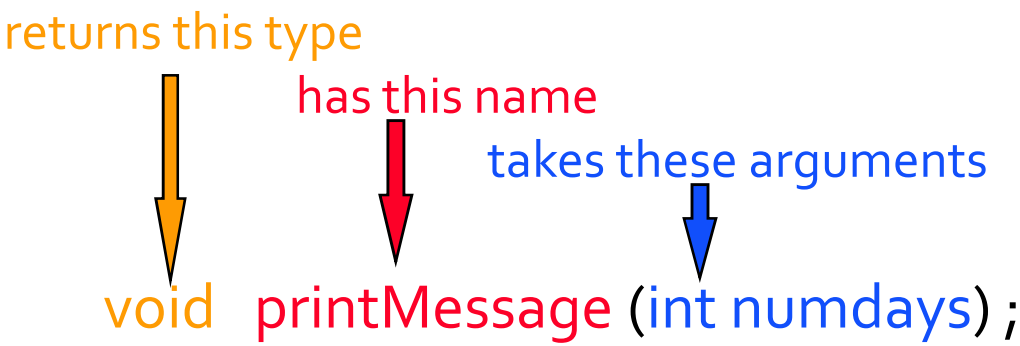
function **body**

Why Two Parts?

- Prototypes are placed at the beginning of the file
  - Makes them easy to find (to see what can be called)
  - Must be listed before the function is called
    - The compiler needs to know how the function is defined before it is can be used.
- Definitions are listed at the end of the file
  - Makes them easy to find if you need read/modify them
  - Makes them easy to ignore if you don't want to read them

The Function Prototype

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



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

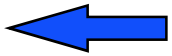
Creating a Function Definition

- Steps:
  1. Copy the function prototype below the main() function
  2. Remove the semicolon
  3. Add an open brace '{' on the next line
  4. Add a close brace '}' below the open brace
  5. Implement the function between the two braces (in the body)
     - Don't forget to indent the body of the function

The Function Definition

- Function **definition** must match the prototype in

```
void PrintMessage (int numdays) ;
...
void PrintMessage (int numdays)
{
    printf ("A message for you: \n") ;
    printf (("Have a nice %d days! \n", numdays) ;
}
```



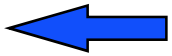
function prototype

function definition

The Function Call

- Function **call** must match the prototype in
  - name
  - number and data types of arguments

```
void PrintMessage (int numdays) ;
```



function prototype

```
int main ( )
{
   PrintMessage (5) ;
   return 0 ;
}
```

function call

The Function Call

- Control is passed to the function by a function call.
  - The value of the arguments are passed from the caller to the function.
    - Values are assigned to the variables in the argument list
  - The statements within the function body will then be executed.
  - After the statements in the function have completed, control is passed back to the calling function
    - The value in the function's return statement is sent back to the caller.

The Function Call

```
#include <stdio.h>

void PrintMessage ( int  numdays) ;

int   main ( )
{
     PrintMessage ( 5 ) ;
     return 0 ;
}

void PrintMessage ( int   numdays)
{
     printf ("A message for you:\n\n") ;
     printf ("Have a nice %d days! \n", numdays) ;

     return;
}
```

function prototype definition

function call

function definition

Function With Parameters and Return Value

int return value

Two int parameters named num1 and num2

```
int add_these( int num1, int num2 )
{
   int sum = num1 + num2;
   return sum;
}
```

The value of the variable ‘sum’ is returned

The variables ‘num1’ and ‘num2’ only exist within this function

Final "Clean" C Code

```
/* ************************************************
   File: kpm.c
   Name: A. Student
   Username: astudent1
   Date: 10/24/07
   Description:    Gets miles from the user and displays that distance in
                   kilometers
************************************************** */
#include <stdio.h>

/* Conversion factor for miles to kilometers */
#define  KM_PER_MILE   1.609344

float  convertToKm( int  miles) ;
```

Final "Clean" C Code

```
int main ( )
{
      int miles = 0;        /* miles entered by the user */
      float  km = 0;        /* distance in kilometers      */

      /* Get distance in miles from the user */
      printf ("Enter  the number of miles: ") ;
      scanf ("%d", &miles) ;
      printf("You said %d  miles \n" , miles) ;

      /* Convert miles to kilometers and print it out */
      km = convertToKm(miles);
      printf("%d  miles is  %f  kilometers  \n" , miles, km) ;
      return 0 ;
}
```

Final "Clean" C Code (con't)

```
/*****************************************************************
** convertToKm  -  converts distance from miles to km
** Inputs:  miles – distance in miles
** Outputs:  float – distance in kilometers
*****************************************************************/

float convertToKm( int  miles)
{
        /* Do the conversion */
        float  k  = miles * KM_PER_MILE;

        return k;
}
```

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

Classwork 1

- Create a **cw1** subdirectory in your CMSC104 directory
- Use the code from the kilometers.c program to write a new program called celsius.c
  - Ask the user to input a temperature value in fahrenheit
  - Write a function that will convert that value to celcius
    - Deduct 32, then multiply by 5, then divide by 9
  - Call the function from main, return the converted value from the function, and print out the conversion
- Use the gcc command the compile and run the code
- Submit your source code and the executable to
  - **cw01** project for our class