Chapter 5 - Functions

Outline

Reading list

- From Teach yourself book
  - Section 1.7, 1.8 and 1.9
  - Class lecture

5.1 Introduction

- Divide and conquer
  - Construct a program from smaller pieces or components
  - Each piece more manageable than the original program

5.2 Program Modules in C

- Functions
  - Modules in C
  - Programs written by combining user-defined functions with library functions
    - C standard library has a wide variety of functions
    - Makes programmer's job easier - avoid reinventing the wheel

5.2 Program Modules in C (II)

- Function calls
  - Invoking functions
    - Provide function name and arguments (data)
    - Function performs operations or manipulations
    - Function returns results
  - Boss asks worker to complete task
    - Worker gets information, does task, returns result
    - Information hiding: boss does not know details

5.3 Math Library Functions

- Math library functions
  - perform common mathematical calculations
  - `#include <math.h>`

- Format for calling functions

  `FunctionName (argument);`

  - If multiple arguments, use comma-separated list
  - `printf( "%.2f", sqrt( 900.0 ) );`
    - Calls function **sqrt**, which returns the square root of its argument
    - All math functions return data type **double**
  - Arguments may be constants, variables, or expressions

5.4 Functions

- Functions
  - Modularize a program
  - All variables declared inside functions are local variables
    - Known only in function defined
  - Parameters
    - Communicate information between functions
    - Local variables
- Benefits
  - Divide and conquer
    - Manageable program development
  - Software reusability
    - Use existing functions as building blocks for new programs
    - Abstraction - hide internal details (library functions)
  - Avoids code repetition

5.5 Function Definitions

- Function definition format

return-value-type *function-name* **(** *parameter-list* **)**
{
declarations and statements
}

  - Function-name: any valid identifier
  - Return-value-type: data type of the result (default **`int`**)
    - **`void`** - function returns nothing
  - Parameter-list: comma separated list, declares parameters (default **`int`**)

5.5 Function Definitions (II)

- Function definition format (continued)

  return-value-type function-name **(** *parameter-list* **)**

  {

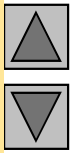
  declarations and statements

  }

  – Declarations and statements: function body (block)
    - Variables can be declared inside blocks (can be nested)
    - Function can not be defined inside another function
  – Returning control
    - If nothing returned
      – **`return;`**
      – or, until reaches right brace
    - If something returned
      – **`return`** *expression* **`;`**

```c
/* Fig. 5.4: fig05_04.c
   Finding the maximum of three integers */
#include <stdio.h>

int maximum( int, int, int );   /* function prototype */

int main()
{
   int a, b, c;

   printf( "Enter three integers: " );
   scanf( "%d%d%d", &a, &b, &c );
   printf( "Maximum is: %d\n", maximum( a, b, c ) );

   return 0;
}

/* Function maximum definition */
int maximum( int x, int y, int z )
{
   int max = x;

   if ( y > max )
      max = y;

   if ( z > max )
      max = z;

   return max;
}
```

```
Enter three integers: 22 85 17
Maximum is: 85
```



Outline

1. Function prototype (3 parameters)

2. Input values

2.1 Call function

3. Function definition

Program Output

5.6 Function Prototypes

- Function prototype
  - Function name
  - Parameters - what the function takes in
  - Return type - data type function returns (default **`int`**)
  - Used to validate functions
  - Prototype only needed if function definition comes after use in program

    ```
    int maximum( int, int, int );
    ```

    - Takes in 3 **`int`**s
    - Returns an **`int`**
- Promotion rules and conversions
  - Converting to lower types can lead to errors

5.7 Header Files

- Header files
  - contain function prototypes for library functions
  - **<stdlib.h>** , **<math.h>** , etc
  - Load with **#include <filename>**

    #include <math.h>

- Custom header files
  - Create file with functions
  - Save as **filename.h**
  - Load in other files with **#include "filename.h"**
  - Reuse functions

5.8 Calling Functions: Call by Value and Call by Reference

- Used when invoking functions
- Call by value
  - Copy of argument passed to function
  - Changes in function do not effect original
  - Use when function does not need to modify argument
    - Avoids accidental changes
- Call by reference
  - Passes original argument
  - Changes in function effect original
  - Only used with trusted functions
- For now, we focus on call by value

5.9 Reversing an integer

```
int reverse (int numb)
{
   int r, sum = 0;
   do
   {
      r = numb%10;
      numb = numb/10;
      sum = sum*10+r;
   }while (numb > 0);
   return sum;
}

int main()
{
printf("Reversing: %d Result %d",1243,reverse(reverse(1243)));
}
```


5.10 Power Function

```
int power(int x, int y)
{
   int i, p = 1;

   for(i = 1; i <= y; i++)
      p = p*x;
   return p;

}
int main()
{
printf("Power of 2^5 is %d",power(2,5));
}
```


5.11 Determining how many digits in a number

```
int howManyDigits(int numb)
{
   int r,c = 0;
   do
   {
      r = numb%10;
      numb = numb/10;
      c++;
   }while (numb > 0);
   return c;

}

int main()
{
printf("Number of digits: %d Result %d",1345,howManyDigits(1345));
```