Introduction to Programming

Lecture 6: Functions & Program Structure

Mahmoud El-Gayyar

elgayyar@ci.suez.edu.eg

Review of Chapter 5

- *Arrays*
  - Initialization
  - Multi-dimensional arrays
- *More Operators*
  - Assignment operators
  - Increment and decrement operators
  - Order of evaluation

Outline

- *Functions Basics*
  - Function Prototypes
- *Void (Non Value-Returning) Functions*
- *Variables Visibility and Lifetime*
  - Default Initialization
  - Examples

Outline

- ***Functions Basics***
  - Function Prototypes
- *Void (Non Value-Returning) Functions*
- *Variables Visibility and Lifetime*
  - Default Initialization
  - Examples

What is a Function?

What is a Function?

- *Code within a function should has these properties:*
  - It performs some ***well-defined task (Useful to the program)***
  - It might be useful to other programs as well
  - The rest of the program doesn't have to know the details of how the function is implemented
  - Avoid to repeat code in the program (***simpler code***)
  - Can be re-written (***improved***) while the rest of the program is not modified.

Function Basics

- *A function consists of:*
  - Name
  - Parameters (inputs)
  - Body (set of instructions: sequential, loop, conditional)
  - Return type (the type of its output)
- *Example: Multiply an int by two function,*

```
int multbytwo(int x){
            int retval;
            retval = x * 2;
            return retval;
}
```

Function Basics

- *The previous function can be written in a simplified format: (return can be used to return an expression)*

```
int multbytwo(int x){
            return x * 2;
}
```

- *But how can we call our defined function?!*
  - For this we will see a full program using the mltbytwo function.

Functions: Full Example

```
#include <stdio.h>

int multbytwo(int);              //function prototype

void main(){
      int i, j;
      i = 3;
      j = multbytwo(i);          //function call
      printf("%d\n", j);
}

/*----------Function multbytwo-----------*/
int multbytwo(int x){
      return x * 2;
}
```

Function Prototypes

- *prototypes help to*
  - ensure that the compiler can generate correct code for calling the functions
  - allowing the compiler to catch certain mistakes you might make
  - however, prototypes are optional. (Define functions before main)
- *Actually header **(.h)** files contains only functions prototypes while code is available in dynamic libraries **(.dll files)***

Outline

- *Functions Basics*
  - Function Prototypes
- ***Void (Non Value-Returning) Functions***
- *Variables Visibility and Lifetime*
  - Default Initialization
  - Examples

Void Functions

- *Void functions are created and used just like value-returning functions except they do not return a value after the function executes.*
- *Example: Write a function to print "Hello" for n times on the screen.*

```
void printHello(int num){
     for(int i=0;i<num;i++)
          printf("Hello\n");
}
```

- *How to call?!*

Parameters Passing: arrays

- *When passing an array to a function, we only need to specify the array name*
- *The following example is invalid*

```
void f(int x[20]){
  …
  }

void main(){
  int y[20];
  f(y[0]); //invalid, type mismatch
}
```

Parameters Passing: arrays

The size of array is optional.

```
void f(int a[])
```

if the content of a[i] is modified in the function, the modification will persist even after the function returns (Call by reference)

```
void f(int a[3]){
   cout << a[0] <<endl; //1 is printed
   a[0]=10;
}

void main (void) {
   int a[3]={1,2,5}; //an array with 3 elements
   f(a); //calling f() with array a
   cout << a[0] <<endl; //10 is printed
}
```

Only need to input the array name!

Parameters Passing: arrays

Write a C function to count n numbers from an array?

Outline

- *Functions Basics*
  - Function Prototypes
- *Void (Non Value-Returning) Functions*
- ***Variables Visibility and Lifetime***
  - Default Initialization
  - Examples

Variable Visibility

- *The **visibility** of a variable determines how much of the rest of the program can access that variable.*
- *A variable declared within a block (braces { }) are called **local variables** → **Visible only within the block***
  - Function blocks
  - for/if/switch blocks
- *a variable declared outside of any function is a **global variable**, and it is potentially **visible anywhere** within the program.*

Variable Life Time

- ***Automatic duration:*** *start at the beginning of the block and they (and their values) disappear at the end of the block (e.g. local variables).*
- ***Static duration:*** *they last, and the values stored in them persist (for sure can be changed), for as long as the program does. (e.g. global variables)*
- *static keyword can be used to switch the local variable duration into a static one.*

Example: Variable Life Time

```
void staticExample( );
int z=2;
void main(){
      cout<<z++<<endl;
      staticExample();
      staticExample();
      staticExample();
}

void staticExample( ){
      int x=0;
      static int y=0;
      cout<< x++ << y++<<endl;
      cout<<z++<<endl;
}
```

```
2
0 0
3
0 1
4
0 2
5
```

Variable Initialization

- *If you do not explicitly initialize them, **automatic-duration** variables (that is, local, non-static ones) are not guaranteed to have any particular initial value (**garbage**)*
- *Static-duration variables (**global and static local**), on the other hand, are guaranteed to be initialized to 0 (**zero**) if you do not use an explicit initializer in the definition.*

Example 1

```
int globalvar = 1;

int anotherglobalvar;


f(){
            int localvar;

            int localvar2 = 2;

            static int persistentvar;
}
```

Example 2

Write a function to compute the factorial of a number, and use it to print the factorials of the numbers 1-7.

```
int fact (int n );

int main(){
        for(int i=1; i<=7 ; i++)
                printf("factorial of %d equals %d \n", i, fact(i));

        return 0;
}

int fact (int n ){
        int factorial=1;

        for(int i=n; i>1 ; i--)
                factorial=factorial*i;

        return factorial;
}
```

Summary

- *How to write functions*
- *Don’t forget your prototype*
- *Difference between local and global variables*