C Basics, I/O, Parameter Passing, Functions

Prepared by: RNC Recario
rncrecario@gmail.com
Institute of Computer Science UPLB
Nov 2011

OBJECTIVES

At the end of the laboratory session, the student is expected to

- Recall the fundamentals of C Programming from CMSC 11
- Identify the basic C routines for input and output
- Identify and recall the different types of parameter passing
- Identify and recall functions
- Develop a modular programming approach of solving a problem
- Solve an exercise which involves modularization and parameter passing

C BASICS and I/O

- According to types of statements:
  - Sequential –statements that are executed sequentially.
  - Conditional – statements that are executed depending on the truth value of a condition imposed
  - Iterative – statements that are repeated done over a number of times or when a condition is already FALSE.

- Basic input / output
  - `printf("[text|formatspec]", [var|const]);`
    – prints the value unto the screen
    – `text` represents any text or string of characters
    –`formatspec` represents the format specifiers in C (e.g., `%d`, `%i`, `%f`, `%lf`)
    – `var` represents any variable corresponding with the `formatspec`
    –`const` represents any constant corresponding to the `formatspec`
  - `scanf("formatspec[,formatspec]", (&var|var)[,&var|var]);`
    – gets input from the keyboard
    –formatspec represents the format specifier
    –var represents the variable corresponding with the formatspec. This is used if the variable is non-atomic data type.
    –&var represents the memory address of a variable. This is used if the variable is atomic data type.

FUNCTIONS

Functions are common constructs in C. By definition, a function is a block of codes that does some specific tasks. A function is composed of a function name, argument(s) or parameter(s), function body and the return value. Almost all programming language has an equivalent function construct.

```
int computeSum(int a, int b){
     return(a + b);
}
```

A function can be "classified" into four types according to the arguments and return type.

A function can be any of the following

- No argument(s), no return value
- No argument(s), has return value
- Has argument(s), no return value
- Has argument(s), has return value

A function can have multiple arguments but can only return at most 1 value. However, if there is a need to return two or more values, this can be accomplished through pointers (an example of this is passing an array in and out of a function).

A common mistake done by students (and that might include you in the near future) is that they usually create or define a function within another function. Here is an example of this erroneous approach.

```
int getInput(){
     int a=3, b=5;
int computeSum(int a, int b){
                return(a + b);
}
}
```

Note that the above example is WRONG. A correct way to do that would like something like the one shown below although it will still not work. (The next question is why?)

```
int getInput(){
     int a=3, b=5;
}
int computeSum(int a, int b){
return(a + b);
}
```

A function can be invoked or called from the `main()` or in any function. An example of a function call would look something like this:

```
int k;
k = computeSum(5, 4);
```

unlike a function definition or creation, function calls can be nested. A valid function call would look something like this:

```
k = computeSum(computeSum(1,2),3);
```

while an invalid function call based on the above example would look something like this:

```
k = computeSum(computeSum(1,2,3),3);
```

C follows a very rigid rule when calling a function. In C, a function call to another function is allowed if the function to be called has already been defined before the function calling it. To simplify my point, let’s create a simple example.

```
void func_a(){
    func_b();
}
void func_b(){
   func_c();
}
void func_c(){
  //some codes
}
```

The above function calls will not work because function `func_a` has been defined first before `func_b` which it needs to call. In order for this set of functions to work properly, the ordering should be:

```
void func_c(){
  //some codes
}
void func_b(){
   func_c();
}
void func_a(){
    func_b();
}
```

In some cases, a function needs to be called more than once by some functions. This implies a case that function calls will violate the said rule. In order to override this rule, we have what we call function prototype. By function prototyping, we are declaring the functions before they are defined. This gives us the opportunity to validly declare and define functions in any order.

An example of function prototype is shown below:

```
void func_a();
void func_b();
void func_c();
void func_c(){
  //some codes
}
void func_b(){
   func_c();
}
void func_a(){
    func_b();
}
```

PARAMETER PASSING

There are types of parameter passing: pass by value and pass by reference. A pass by value allows you to create another copy of the value within a variable that you passed. In contrast, a pass by reference allows you to pass the actual variable that you need by passing its memory address. In practice or application, the difference of the two types is not observable. But if you are handling a huge amount of data (e.g, 2d array, let’s say an image), pass by reference should be the one used.

An example of a pass by value is this one:

```
int computeSum(int a, int b){
      return(a +b);
}
main(){
int a=5, b=4;
printf("%d", computeSum(a, b));
}
```

An example of a pass by reference is this one:

```
int * computeSum(int *a, int *b){
      return(*a +*b);
}

main(){
int a=5, b=4, c;
c=computeSum(&a, &b));
printf("%d", c);
}
```

Exercise 1: C Basics

Prepared by: RNC Recario
rncrecario@gmail.com
Institute of Computer Science UPLB
Nov 2011

filename: `<surname>_<section>_exer<exerno>.c`
Example: `recario_u1l_exer1.c`
Ensure that you have a backup copy of your work. Always save your work! You will show your work to me after you have finished working.

The Problem

You need to create a basic calculator application subject to the following conditions:
There are four operations valid: addition, subtraction, multiplication and division. For simplicity, the operation works for integer-based values only.
Your application must be able to catch some mathematical conditions. Example, division by zero is not allowed.
Everything should be modular. Everything should be pass by reference.
A menu should be available that allows you to choose what operation you need to do.
A skeleton c code is provided for you to start. You are allowed to modify it if you are not comfortable with my coding style provided that you will still meet the conditions outlined in this exercise.