Fundamentals of Programming

Lecture 1
Hamed Rasifard
Outline

- C’s Memory Map
- Global & Local Variables
- Calling Functions by Value & Reference
- Recursion
Before concluding this chapter, a few words about C++ are in order. Newcomers are sometimes confused about what C++ is and how it differs from C. In short, C++ is an object-oriented programming language that was built upon the foundation of C. In general terms, C is a subset of C++, or conversely, C++ is a superset of C.

In general, you can use a C++ compiler to compile a C program. In fact, today most compilers handle both C and C++ programs. Thus, most programmers will use a C++ compiler to compile their C code! However, since C++ was built upon the 1989 C standard, you must restrict your C code to the features defined by that standard (which are the features described in Part One of this book).

There is one thing that you must be careful about when using a C++ compiler to compile a C program: the file extension. By convention, C programs use the .C extension. C++ programs use .CPP. Don't accidentally give your C program a .CPP extension. Differences between the two languages might prevent a valid C program from being compiled as if it were a C++ program. By specifying the .C extension, you are telling the C++ compiler to perform a "C compile."

Global Variables

- Global variables are known throughout the program and may be used by any piece of code.
- They will hold their value throughout the program's execution.
- Global variables are created by declaring them outside of any function.
Local Variables

- *Local variables* are declared inside a function.

- Local variables can be used only by statements that are inside the block in which the variables are declared.

- Local variables exist only while the block of code in which they are declared is executing.
Local Vs. Global

```
#include <stdio.h>

int count;
void function1();

int main()
{
    int x = 10;
    count = 100;
    function1();
    printf("x in main function is equal to \%d\n", x);
    printf("count is equal to \%d\n", count);
    return 0;
}

void function1()
{
    int x = 20;
    printf("x in function1 is equal to \%d\n", x);
    count = 200;
}
```
Calling Functions by Value

• In call-by-value a copy of the argument’s value is made and passed to the called function

• Changes to the copy do not affect an original variable’s value in the caller

• Call-by-value should be used whenever the called function does not need to modify the value of the caller’s original variable
Calling Functions by Reference

• In call-by-reference the caller allows the called function to modify the original variable’s value

• Call-by-reference should be used only with trusted called functions that need to modify the original variable
Value Vs. Reference

- In C, all calls are by value
- It’s possible to simulate call-by-reference by using address operators and indirection operators
- Arrays are automatically passed by reference
Recursion

- A recursive function is a function that calls itself either directly or indirectly through another function.
- The function actually knows how to solve only the simplest case(s), or so-called base case(s).
- If the function is called with a base case, the function simply returns a result.
Recursion Form I

```c
return_type function_1(parameters) {
    if (base_case)
        return result;
    else
        return function_1(parameters);
}
```
Recursion Form 2

return_type function_1(parameter(s))
{
    if(base_case_1)
        return result_1;
    else
        return function_2(parameter(s));
}

return_type function_2(parameter(s))
{
    if(base_case_2)
        return result_2;
    else
        return function_1(parameter(s));
}
Recursion Example 1

- A function that computes $x^n$

```c
long double power(double x, double n)
{
    if (n==0)
        return 1;
    else
        return x * power(x, n-1)
}
```
Recursion Example 2

- A function that computes factorial

```c
long double factorial(double number)
{
    if (number <= 1)
        return 1;
    else
        return number * factorial(number - 1);
}
```
Towers of Hanoi

- The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:
  - Only one disk may be moved at a time.
  - Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
  - No disk may be placed on top of a smaller disk.
Example of Towers of Hanoi

- Initial Setup
- Step 1
- Step 2
- Step 3
- Step 4
- Step 5
- Step 6
- Step 7
void hanoi(int num_disk, char from_rod, char to_rod, char aux_rod) 
{
    if( num_disk == 1) 
    { 
        printf("Move Disk From %c to %c\n",from_rod,to_rod); 
        return; 
    } 
    else 
    { 
        hanoi(x-1,from_rod,aux_rod,to_rod); 
        printf("Move Disk From %c to %c\n",from_rod,to_rod); 
        hanoi(x-1,aux_rod,to_rod,from_rod); 
        return; 
    } 
}