Programming in C

Chapter - 6

Functions

by, Santa Basnet

Functions

Introduction

Every **‘C’** program consists of one or more functions.

Functions

Introduction

- Functions let you to divide a larger computing tasks into many smaller units.
- C language is designed to write functions for smaller jobs.
- It allows you to use already existing work to avoid writing program from scratch.

Functions

Function Prototype

- A Function in C has following components:
  1. a prototype
  2. a definition
  3. arguments
  4. return expression
- Before defining a function, we declare a function prototype (A function prototype is not must in C but remember the good practice).
- The prototype assist the compiler to verify whether the function calling is made correctly or not.

Functions

Function Definition

- A function prototype should look like this:

```
return-type function-name(arguments-type);
```

- A function definition contains all the source codes of your computation task.

- A function definition should look like this:

```
return-type function-name(argument
declarations)
{
        declarations and statements
}
```

Functions

Function Definition

- A minimal function: no arguments and no return-type, does nothing for you.

```
empty(){
}
```

- A typical function definition: adds two integers.

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

Functions

Function Call/Access

- A typical C program should have at least main function.
- Functions communicate each other by passing arguments and value returned, and through external variables.
- A typical function call look like this:

```
int s = sum(500, 945);
```

Functions

Function Return-type

- The return statement is the mechanism for returning a value from the called function to its caller.
- Any expression can follow return:

```
return expression;
```

- The expression will be converted to the return type of the function. Parenthesis in the expression is optional.
- Returns always a single value.

Functions

Function Return-type

- After return statement, the control return to the caller function.

```
#include <stdio.h>
int add(int, int); /* Prototype */
int main()
{
    int result = add(100, 500); /* Call and assign return
                                   to the result.*/
    printf("Sum (x, y) = %d", result); /* Print result */
    return(1); /* main function return a constant(int) 1
                                   */

}
int add(int x, int y){ /* A function definition */
    return(a+b); /* A return expression results the
                    evaluation of the expression */
}
```

Functions

Class Work

- Functions: lowercase to UPPERCASE example.

Functions: Recursion

Recursive Functions

- Recursion is a process by which a function calls itself repeatedly, until some specified/base condition has been satisfied.
- With recursive process, the repetitive computations can be defined in in terms of a previous result.
- Many iterative (repetitive) problems can be solved with recursion.

Functions: Recursion

Recursive Functions

- A Recursive Factorial Function:

```c
/** A Recursive factorial function. **/
#include<stdio.h>
long int factorial(int n){
    if(n <= 1) return 1;
    else return (n * factorial(n -1 ));
}

int main() {
    printf("Factorial of 5 = %ld", factorial(5));
    return 0;
}
```

Functions: Recursion

Recursive Functions

- A Recursive Factorial Function:

$$1! = 1$$

$$2! = 2 \times 1! = 2 \times 1 = 2$$

$$3! = 3 \times 2! = 3 \times 2 = 6$$

$$4! = 4 \times 3! = 4 \times 6 = 24$$

. . . . . .

$$n! = n \times (n - 1)! = \cdots$$

Functions: Recursion

Recursive Functions

- A Recursive Reverse :

```c
#include<stdio.h>
void reverse(void);
int main() {
    reverse();
    return 0;
}
/** Read a single line and print in reverse.**/
void reverse() {
    char c;
    c = getchar();
    if (c != '\n')
        reverse();
    putchar(c);
    return;
}
```

Functions: Recursion

Recursive Functions

- A Recursive Reverse Stack:

```
HELLO

to

OLLEH

How ?
```

Functions: Recursion

Recursive Functions

- Problem of Recursive Function:

Computation/Memory overflow ???

Functions: Recursion

Recursive Functions

- Solution:

Tail Recursion

Functions: Recursion

Recursive Functions

- Solution:

```
#include<stdio.h>
/** Factorial with Tail Call Recursion. **/
long factorialGo(long number, long result) {
    if (number <= 1) return result;
    else return factorialGo(number - 1, result * number);
}

/** A Factorial function that calls from Main. **/
long factorial(long number) {
    return factorialGo(number, 1);
}

int main() {
    factorial(10);
    return 0;
}
```

Thank you.

Questions ?