C under Linux

Dr. Naeem Odat

Department of Computer and Communications Engineering

C - Functions

C - Functions

C - Functions

Functions

- A function is a group of statements executed when it is called.
- Once the statements in the function are executed, program flow resumes from the place where we called the function from.
- This helps us reduce redundancy in our code.
- Some C functions are *main*, *printf*, *scanf*, etc.

C - Functions

Using functions

```
#include<stdio.h>
int sum(int, int); /*Declaration*/
int main(){
    int a = 10, b = 20, c = 0;
    c = sum(a, b); /*Call */
    printf("Sum is %d\n", c);
    return(0);
}
/* Definition */
int sum(int a, int b){
    return(a + b);
}
```

C - Functions

Function definition

- Arguments are optional.
- Functions can take arguments (extra information for it to work).
- The function body contains the statements of the function.
- The return value is the value that is passed back to the calling function.
- Functions terminates whenever a return statement is encountered.

C - Functions

Function arguments

- Arguments helps us pass information to the function body when the function is called.
- There can be zero or more arguments to the function.
- The function arguments acts as local variable to the function.
- Arguments are **passed by value**.
- Passing argument value when calling the function, acts as the initial value for the argument variable.
- Arguments can be **passed by reference**.

C - Functions

Function arguments

- When declaring the function, the argument type is mandatory, where as the argument name can be left.
- Example:

```
int sum(int, int); /*Function Declaration*/
int sum(int a , int b) /* Function Definition*/{
    <Set of Statements>
}
```

- If a function has no arguments, specify **void** within the parenthesis of the function declaration and definition.

C - Functions

return statement

- **return** statement is used to return a value from the called function to the calling function.
- The data type of the value returned by the function is denoted by the return type specified in the function declaration and function definition.
- As soon as the return statement is encountered within the function, the control is transferred to the calling function.
- If the return type is omitted, the function is presumed to return value of type int.
- Specify the return type of the function as void, if the function is not returning any value.
- **return** statements may appear in functions whose return type is void, provided that no expression is given:

  ```
  return; /* return in a void function*/
  ```

C - Functions

Definition before call - example

```
#include <stdio.h>
int sum(int a, int b){
    return(a + b);
}

int main(){
    printf("%d\n", sum(10,20));
    return 0;
}
```

C - Functions

Structure as a function argument

```
typedef struct{
    char name[20];
    float total;
    char grade;
}student_record;

void print_record(student_record s){
    printf("Name: %s\n", s.name);
    printf("Numerical Grade: %f\n", s.total);
    printf("Letter Grade: %c\n", s.grade);
}
```

C - Functions

Structure as a return value

```
#include<stdio.h>
typedef struct {
    int hh, mm, ss;
} Time;
Time GetSystemTime(){
    Time t;
    /* Do some processing and get time in t */
    return(t);
}
```

C - Functions

Reusability of functions *(gcc -o Ourmain OurMain.c MyLib.c)*

- MyLib.h:

  ```
  /* Declare Functions and Global variables*/
  int Display();
  ```

- MyLib.c:

  ```
  /*Define Func*/
  #include<stdio.h>
  #include "MyLib.h"
  int Display(){ ...}
  /*Other function definitions*/
  ```

- OurMain.c:

  ```
  /*Main Function*/
  #include<stdio.h>
  #include "MyLib.h"
  int main(void) {
      Display();
  }
  ```

C - Functions

Advantages of Separate Files

- The functions declared in a separate file can be used by different programs.
- We can maintain our own library which will have similar set of functions with a single file.
- Debugging of the code is easier.

C - Functions

Storage Classes

- Every variable in C programming has two properties: type and storage class.
- Storage class determines the **scope** (**visibility**) and **lifetime** of a variable.
- There are 4 types of storage class:
  - automatic.
  - register.
  - static.
  - external.

C - Functions

The auto Storage Class

- It is the default for all local variables.
- It is local and its life time is the life of the function.

```
{
    int mount;
    auto int month;
}
```

C - Functions

The register Storage Class

- It is used to define a local variable.
- The & unary operator cannot be used with a variable defined by register.

```
{
    register int  miles;
}
```

C - Functions

The static Storage Class

- Keeps a local variable in existence during the life-time of the program.
- Causes a global scope to be restricted to the file in which it is declared.

```
#include <stdio.h>
void func(void);
static int count = 5; /* global variable */
main() {
    while(count--) {
        func();
    }
    return 0;
}
void func( void ) {
    static int i = 5; /* local static variable */
    i++;
    printf("i is %d and count is %d\n", i, count);
}
```

C - Functions

The extern Storage Class

- Causes a global variable to be visible to ALL the program files.
- The extern modifier is most commonly used when there are two or more files sharing the same global variables or functions as explained below.

```
#include <stdio.h>//main.c
int count ;
extern void write_extern();
main() {
    count = 5;
    write_extern();
}
#include <stdio.h>//support.c
extern int count;
void write_extern(void) {
    printf("count is %d\n", count);
}
```