C Programming – Lecture V

Instructor Özgür ZEYDAN
http://cevre.beun.edu.tr/

Modular Programming

- A function in C is a small "sub-program" that performs a particular task, and supports the concept of modular programming design techniques.
- In modular programming the various tasks that your overall program must accomplish are assigned to individual functions and the main program basically calls these functions in a certain order.
- The **`main`** body of a C program, identified by the keyword main, and enclosed by left and right braces is a function.
- It is called by the operating system when the program is loaded, and when terminated, returns to the operating system.

Reasons for Modular Programming

- Construct a program from smaller pieces or components
- Avoids code repetition. Don't have to repeat the same block of code many times in your code. Make that code block a function and call it when needed.
- Easy to debug. Each piece more manageable than the original program
- Easy to modify and expand. Just add more functions to extend program capability.
- Software reusability. Use existing functions as building blocks for new programs

Function Definitions

- Function definition format:

```
return-value-type  function-name (parameter-list)
      {
       declarations and statements
      }
```

- Function-name: any valid identifier
- Return-value-type: data type of the result (default int)
  - void - function returns nothing
- Parameter-list: comma separated list, declares parameters (default int)

Function Definitions

```
return-value-type  function-name (parameter-list)
      {
       declarations and statements
      }
```

- Declarations and statements: function body
  - Variables can be declared inside function body
  - Function can not be defined inside another function
- Returning control
  - If nothing returned: `return;`
  - If something returned: `return expression;`

Function Prototypes

- Used to validate functions
- Prototype only needed if function definition comes after use in program

- Examples:
- `int factor(int);` /* function prototype */

- `void intro(void);` /* function prototype */

Functions

- Functions that do not receive or send data.
- You can use:

```
void function-name (void)
{
Statements;
}
```

Example – 1

- Write a C code to display an introduction message (given below) on the beginning of the program.
- Use function (function will neither receive nor return any value)
- Introduction message:

```
======================
*  Programmer : Özgür ZEYDAN   *
======================
```

Example – 1

```
#include<stdio.h>
#include<conio.h>
void intro(void);      /* function prototype */
int main(void)         /* main function */
{
intro();               /* function call */
   statements;
}
void intro(void)       /* intro function */
{ printf("======================\n");
  printf("*  Programmer : Özgür ZEYDAN   *\n");
  printf("======================\n\n");   }
```

Example – 2

- Write a C program that asks user to write two integer values.
- Then computer these two integers and returns result.
- Use local variables.

Example – 2

```
#include <stdio.h>
#include <conio.h>
int sum ();              /* decleration */
main()  {
 int x,y,result;         /* local variables */
 printf("Enter two integers (space in between): ");
 scanf("%d %d",&x,&y);
 result=sum(x,y);   /* function call */
 printf("Sum of two numbers: %d",result);
 getch();
 return 0;  }
int sum(int a, int b)       /* function */
{   return a+b;  }
```

Example – 3

- Write a C program that asks user to write two integer values.
- Then computer these two integers and returns result.
- Use global variables.

Example – 3

```
#include <stdio.h>
#include <conio.h>
int x,y;                /* global variables */
int sum ();             /* decleration */
main() {
 int result;            /* local variable */
 printf("Enter two integers (space in between): ");
 scanf("%d %d",&x,&y);
 result=sum();          /* function call */
 printf("Sum of two numbers: %d",result);
 getch();
 return 0;  }
int sum()               /* function */
{   return x+y;  }
```

Example – 4

- Write a C program that asks user to write an integer values.
- Then computer calculates factorial of a given integer.
- At this time, use modular programming!
- Define a function that
  - receives an integer from main function,
  - calculates and
  - returns value of its factorial.

Example – 4 (using local variables)

```
#include<stdio.h>
#include<conio.h>
int factor(int);                /* function prototype */
int main(void)                  /* main function */
{
int x;                          /* local variable */
printf("Enter integer value to calculate its factorial: ");
scanf("%d",&x);
printf("Factorial of %d is %d",x,factor(x));  /* function call */
getch();
return(0);     }
```

Example – 4 (using local variables)

```
int factor( int a)
{
int fac=1,i; /* local variables */
for (i=1;i<=a;i++)
fac*=i;
return fac;
}
```

Example – 4 (using global variables)

```
#include<stdio.h>
#include<conio.h>
int x;                          /* global variable */
int factor(int);                /* function prototype */
int main(void)                  /* main function */
{
printf("Enter integer value to calculate its factorial: ");
scanf("%d",&x);
printf("Factorial of %d is %d",x,factor(x));  /* function call */
getch();
return(0);     }
```

Example – 4 (using global variables)

```
int factor( int x)
{
int fac=1,i; /* local variables */
for (i=1;i<=x;i++)
fac*=i;
return fac;
}
```

Example – 5

- Write a C program that calculates Combination of a given two numbers `m` and `n`.
- Use a function two make factorial calculations.

$$\binom{m}{n} = \frac{m!}{n!(m-n)!}$$

Example – 5

```
#include <stdio.h>
#include <conio.h>
int factor(int);        /* function prototype */
int main(void)          /* main function */
{
int m,n;
printf("Combination (m/n) calculator.\n");
printf("Enter m and n values: ");
scanf("%d %d",&m,&n);
printf("Combination of (%d/%d) is
   %d",m,n,factor(m)/(factor(n)*factor(m-n)));
        /* function call */
```

Example – 5

```
getch();
return(0);   }

int factor( int x)
{
int fac=1,i;          /* local variables */
for (i=1;i<=x;i++)
fac*=i;
return fac;
}
```

Example – 6

- Write a C program that calculates fifthroot of a given floating point number.
- Use function to calculate fifthroot.

Example – 6

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
float x,y;
float fifthroot ();
main()  {
 printf("Fifthroot calculator\n");
 printf("Enter a number: ");
 scanf("%f",&x);
 y=fifthroot(x);
 printf("Fifthroot of %f is %f",x,y);
```

Example – 6

```
 getch();
 return 0;  }
float fifthroot()
{
 float z;
 z=pow(x,(1.0/5.0));
 return z;
}
```

C Recursive Function

- Recursive function is a function that contains a call to itself. C supports creating recursive function with ease and efficient.
- Recursive function allows you to divide your complex problem into identical single simple cases which can handle easily.
- Recursive function must have at least one exit condition that can be satisfied. Otherwise, the recursive function will call itself repeatly until the runtime stack overflows.

Example of Using Recursive Function

- Recursive function is closely related to definitions of functions in mathematics so we can solving factorial problems using recursive function.
- All you know in mathematics the factorial of a positive integer N is defined as follows:
- N! = N*(N-1)*(N-2)…2*1;
- Or in a recursive way:
- N! = 1 if N <=1 and N*(N-1)! if N > 1

Example – 7

```
# include<stdio.h>
int factor(int a)
{
    if(a <= 1) return 1;
    return a * factor (a - 1);
}
void main()
{
…
printf("Factorial of %d is %d",x,factor(x));
…       }
```