FUNCTIONS

CHAPTER 5
(PART 2)

prepared by Senem Kumova Metin
modified by İlker Korkmaz

A brief review on function invocation and call-by-value concepts

- **If a function is invoked from somewhere, the body of that function will be executed at that moment.**
- **If an argument is passed to a function through the "call-by-value" approach, the stored value in the *caller* environment will not be changed.**

example:

```
#include<stdio.h>
int my_sum(int n);

void main(void)
{   int n=9;
    printf("%d\n" ,n);
    printf("%d\n" ,my_sum(n)); // call (or invoke) "my_sum" function
    printf("%d\n" ,n); // stored value of "n" of "main()" is not changed
}

int my_sum(int n)
{   n=n+2; // stored value of "n" of "my_sum()" is changed
    return n;
}
```

A brief review on developing large programs

- If more than one source file is used within an application, all source files can be compiled seperately and their object files can be linked into one executable file.
- Contents of an example program:

  source files: source1.c + source2.c

  header files: header1.h

  utiliy files: READ_ME

  →to produce the executable file, ***program***:

  gcc –o program source1.c source2.c

What about “TO DO” works?

- ??
- Have you done the previous “TO DO” work, which was given within the last lecture presentations?
  - Implement an “isPrime” function.

Scope Rules (1)

- **Each identifier is accessible only within the block that it has been declared in.**

- **EXAMPLE:**

```
#include<stdio.h>

void func_1(int a)
{    int b, c; .... }

void func_2(int a, double b, float d)
{ char c; .... }

void main ()
{    int a,b,d;
     char c; .... }
```

Scope Rules (2)

- **An outer block name is valid unless an inner block redefines it. If redefined, the outer block name is hidden from the inner block.**

- **EXAMPLE:**

```
{
        int a=2; float b=3.2;  // the outer block's "a" and "b"
        printf("%d",a);
        {
                int a =5;  // the inner block's "a"
                printf("%d",a);
                printf("%f",a+b);
                {
                        int b=4;  // the most inner block's "b"
                        printf("%d",a+b);
                }
        }
        printf("%d",++a); // the outer block's "a"
}
```

Storage Classes (1)

- Every variable and function in C has two attributes:
  - ***type*** and ***storage class***.
- The four storage classes are ***automatic***, ***external***, ***register***, and ***static***.
  - auto
    - Variables declared within function bodies are automatic by default.
  - extern
    - It is used to tell the compiler to look for it elsewhere either in this file or in some other file.
  - register
    - It tells the compiler that the associated variables should be stored in high-speed registers.
  - static
    - It allows a local variable to retain its previous value when the block is reentered.

EXAMPLES:

```
auto float f;  // same with:   float f;
extern int a;
register int i;
static int cnt=0;
```

Storage Classes (2)

extern

```
/* file1.c */

/* file1.c and file2.c are available
   within the same project */

int f(void)
{
   extern int a;/* look for it elsewhere */
   int b, c;

   a = b = c = 4;
   return (a + b + c);
}
```

```
/* file2.c*/

/* file1.c and file2.c are available within the
   same project */

#include <stdio.h>

int   a = 1, b = 2, c = 3;
/* external variables */

int   f(void);

int main(void)
{
   printf("%3d%3d%3d\n", a, b, c);
   printf("%3d\n", f());
   printf("%3d%3d%3d\n", a, b, c);
   return 0;
}
```

Storage Classes (3)

register : an attempt to improve execution time

EXAMPLES:

```
register char c;


register int a=1;
/* IS EQUIVALENT TO */
register a=1; // default type is int
```

Storage Classes (4)

static

- **EXAMPLES:**

```
#include<stdio.h>
int ver();

main()
{
   printf("1. value=%d", ver());
   printf("2. value=%d", ver());
   printf("3. value=%d", ver());
}

int ver()
{
   static int k;
   k=k+5;
   return k;
}
```

Recursion

- A function is said to be recursive if it calls itself, either directly or indirectly.
- EXAMPLES:

```
int main(void)
{
    printf("I am calling myself!\n");
    main();
    return 0;
}
-----------------------------
int sum(int n) // What does this function compute ??
{
    if(n<=1)        return n;
    else            return (n + sum(n-1) );
}
```

RECURSION : FACTORIAL

```
/*iterative version*/
int factorial (int n)
{
    int product=1;

    for(;n>1;--n)
          product=product*n;

    return product;
}
```

```
/* recursive version */
int factorial(int n)
{
    if(n<=1)
       return 1;
    else
       return (n * factorial(n-1));
}
```

TO DO

- Three things to do **in class**:
  - Define “fibonacci” function, and test it by depicting the “function call stack”.
  - Discuss the pros and cons of “recursion” against to “iteration”.
  - Illustrate how to create pseudorandom numbers by calling “srand()” and “rand()”.
- A recursive problem to do **at home**:
  - “Towers of Hanoi” example.