EEM103
Computer Programming

Week7

- Functions
  - Definition & Call
  - Function prototypes
  - Returning a value
  - Scope – Global, Local, Static variables
- Recursive Functions

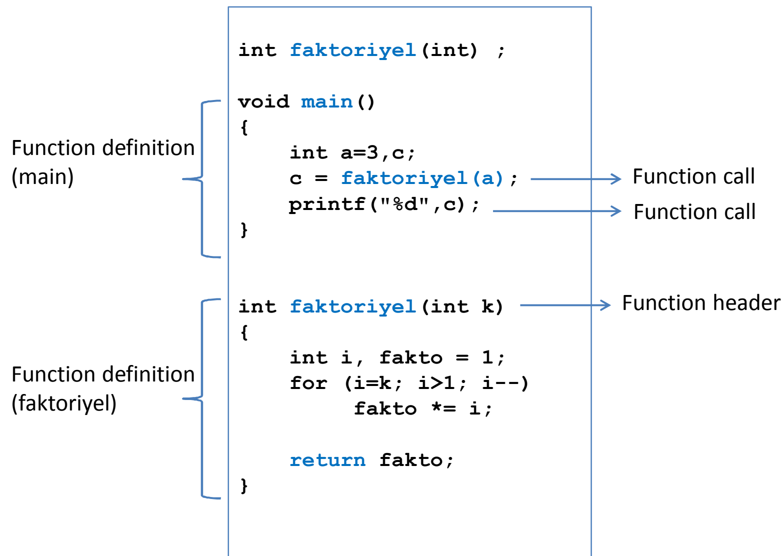
1

Functions

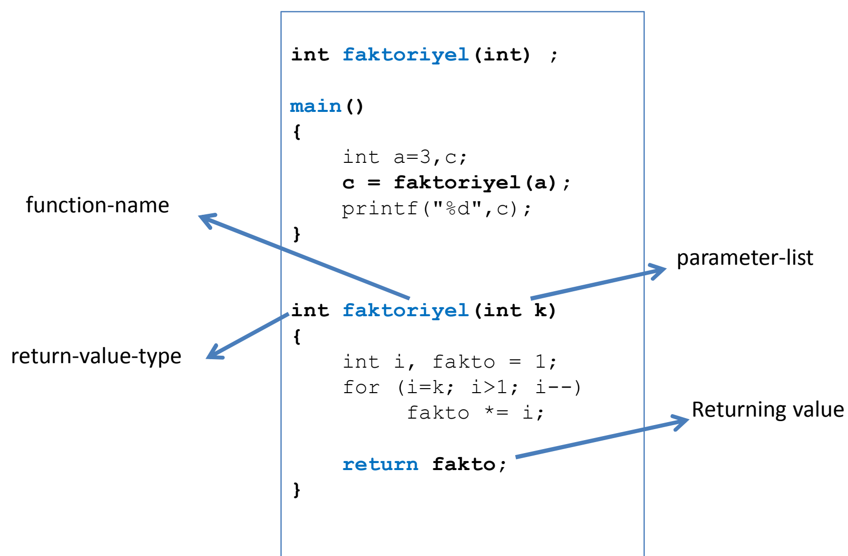
- Function is a code block that performs a specific task
- There is at least 1 function in every C program (main)
- Programmer defined function are *defined* once and are executed every time they are *called*.
- It is easier to write small modules, instead of large main programs.
  - Functions are a way of creating modules
- Motivations;
  - dividing into modules makes program development more manageable.
  - software reusability—using existing functions as building blocks to create new programs.
  - to avoid repeating code in a program.

2

Function definition & Function call



3



4

Prototype

- If the function definition is written after main function block (or if it is defined in another file) a FUNCTION PROTOTYPE should be written before main().
- Prototype is nothing but the function header terminated with a statement terminator (;)
- Prototype informs compiler;
  - The name of the function,
  - its parameter list, and
  - return type.

5

Prototype

```
#include<stdio.h>

int faktoriyel(int k)
{
    int i,fakto=1;
    for (i=k; i>1; i--)
         fakto *= i;

    return fakto;
}


main()
{
    int a=3, b=2,c;
    c=faktoriyel(a)+faktoriyel(b);
    printf("%d ",c);
}
```

```
#include<stdio.h>

int faktoriyel(int);   /* prototype */


main()
{
    int a=3, b=2,c;
    c=faktoriyel(a)+faktoriyel(b);
    printf("%d ",c);
}

int faktoriyel(int k)
{
    int i,fakto=1;
    for (i=k; i>1; i--)
         fakto *= i;

    return fakto;
}
```

6

parameter-list

- The parameter-list is a comma-separated list that specifies the parameters received by the function when it's called.
- If a function does not receive any values, parameter-list is empty (can be written *void*).
- A type must be listed explicitly for each parameter.
  - Parameters are local variables that are belong to the function.
  - In contrast to a normal local variable, they can get value from where the function is called.

7

```
float average(int, int) ;

void main()
{
    int a=3, b=2;
    float avg;
    avg = average(a,b);

    printf("%f",avg);
}


float average(int k, int m)
{
    float result;
    result= ((float)k+m)/2;
    return result;
}
```

a, b, avg: *local* variables of main()

k, m, result: *local* variables of average()

k and m are also *parameters*. i.e: they get value from main().

8

```
float average(int, int) ;

void main()
{
    int a=3, b=2;
    float avg;
    avg = average(a,b);

    printf("%f",avg);
}

float average(int k, int m)
{
    float result;
    result= ((float)k+m)/2;
    return result;
}
```

int k = a

int m = b

a, b, avg: *local* variables of main()

k, m, result: *local* variables of average()

k and m are also *parameters*. i.e: they get value from main().

At function call:
k gets the value of a,
m gets the value of b,

9

return

- A function **can return a single value** to the point it is called.
- The type of the return value is written to the left of the function name at the function definition.
- If a function does not return a value, then its return type is **void**.
- Functions **can not return more than one value** (until pointers chapter)

10

return

```
void function1()
{
    int a=1;
    float b=2.0;
    ...
}
```

```
int function2()
{
    int a=1;
    float b=2.0;
    ...
    return a;
}
```

```
float function3()
{
    int a=1;
    float b=2.0;
    ...
    return b;
}
```

```
int max(int a, int b)
{
    if (a>b)
       return a;
    else
       return b;
}
```

```
int function()
{
    int a=1, b=2;
    return a;
    return b;     !!! No effect
}
```

11

return

- If a function returns a value, it should be used as a right-hand value in function call statement.

```
int  factorial(int);       /* prototype */
void print_max(int, int);  /* prototype */
...
a = factorial (k);
b = factorial (k) + 10;
printf("%d\n", factorial (k));
factorial (k) ;                  !!!! warning
...
print_max(d,e);
p = print_max (d,e) ;            !!!! error
printf("%d\n", print_max(d,e));  !!!! error
...
```

12

Scope of a variable

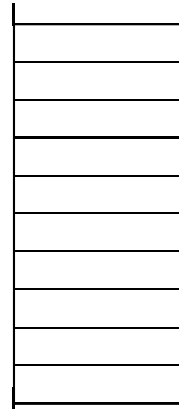
- A variable which is defined in a block (main or function block) is a LOCAL variable belongs to that block
  - The scope of a local variable is its block. It is undefined outside of the block
- A variable which is defined outside of all blocks is called GLOBAL variable.
  - The scope of a global variable is whole program.
- There may be variables which have same name but different scopes.

13

```
#include<stdio.h>

void fonk1();

int a=5;

void main()
{
        int b=6;
        printf("main:%d\n",a);
        fonk1();
}

void fonk1()
{
        int c=7;
        printf("fonk1:%d\n",c);
}
```

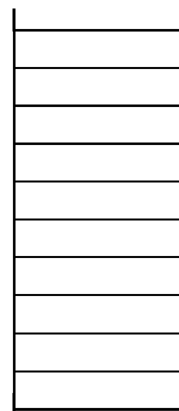

14

```
#include<stdio.h>

void fonk1();
void fonk2();

int a=5;

void main()
{
       int a=6;
       printf("main:%d\n",a);
       fonk1();
       printf("main:%d\n",a);
       fonk2();
       printf("main:%d\n",a);
}

void fonk1()
{
        int a=7;
        printf("fonk1:%d\n",a);
}

void fonk2()
{
        printf("fonk2:%d\n",a);
}
```



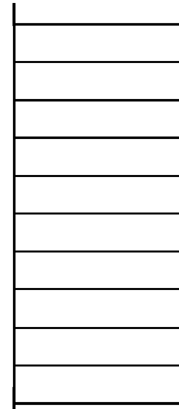
15

```
#include<stdio.h>

void main()
{
       int a=6;

       {
           int a=7, b=0;
           printf("%d %d\n", a, b);
       }

       printf("%d %d\n", a, b);

}
```



16

```
#include<stdio.h>

float average(int, int);

float global_var ;

void main()
{
    int a=2 , b=3 ;
    global_var = 1.0 ;
    float result = average(a,b);

    printf("average: %f\n",result);
    printf("global_var: %f\n",global_var);
}

float average(int s, int t)
{
   float avg;
   avg=(float)(s+t)/2;

   global_var = 100.0 ;

   return avg;
}
```



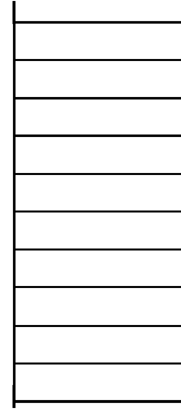
17

static variables

- Local variables declared with the keyword ***static*** are still known only in the function in which they're defined, but static local variables retain their value when the function is exited.
- Static variables are initialized only once, before the program begins execution.
- The next time the function is called, the static local variable contains the value it had when the function last exited.
  - this does not mean that these variables can be accessed throughout the program.

18

```
#include<stdio.h>

int square(int);

void main()
{
    int a=2 , i ;
    for(i=0; i<4; i++)
      a = square(a) ;
    printf("a = %d\n", a );
}

int square(int b)
{
    static int counter = 0 ;
    counter++;
    printf("%d. call, b = %d\n", counter,b);
    return b*b;
}
```



19

Recursive Functions

- For some types of problems, it's useful to have functions call themselves.
- A recursive function is a function that calls itself either directly or indirectly through another function.

  e.g: 5! = 5 x 4!

  e.g: fibonacci(10) = fibonacci(9) + fibonacci(8)

20

Recursive Functions

- Recursive functions generally have two cases;
  - Base case
  - Recursive case
- If the function is called with a base case, the function simply returns a result.

  Base case of factorial function: 1! = 1

  Base case of fibonacci function: fibo(1)=fibo(2)=1

- If the function is called with recursive case, it calls itself.