FUNCTIONS

CHAPTER 5
(PART 1)

prepared by Senem Kumova Metin
modified by İlker Korkmaz

Function concept

- Top-down design
  - Decomposing a problem into small, manageable sub-procedures is critical to write large programs.
- Abstraction
  - The terms of "what" and "how-to" concepts for the procedures can be separated through a black-box notion. This abstraction view provides the programmers with an ease of use on complex functions.

C programs contain function(s)

- Every C program consists of at least one function called “main”.
- The execution of any C program begins at main() function.

function declaration and definition

TO DECLARE A FUNCTION: Declare the *signature* (name, and types of the parameters), and return type.

```
int factorial(int n);   // to be defined later
```

TO DEFINE A FUNCTION: Implement the body.

```
return_type  function_name(input_parameter_list)  /*header of the funciton*/
{ /*body of the function*/
   declarations
   statements
}
```

function definitition examples:

```
double twice(double x)  /* header */
{    double result;  /* body starts here */
     result = x*2;
     return result; // OR return x*2;
}

int add( int x, int y) /* header */
{    int result; /* body starts here */
     result=x+y;
     return result;      // OR  return (x+y);  // OR return x+y;
}
```

global versus local variables (1)

- **Any variable declared within any function is "local" to that function**
- **Any variable declared out of any function is "global" for the related program**
- **EXAMPLE :**

```
#include<stdio.h>

int a =3;  // global variable

void main(void)
{     int b= 7; // local variable to main()
      printf("%d", a);
      printf("%d", b);
      printf("%d", a+b);
      printf("%d", a++);
}
```

global v.s. local variables (2)

```
#include<stdio.h>
int a =3;  // global variable can be accessed from all functions

int new_func(int y);

void main(void)
{       int b=7, r =0;        // local variables to main()

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

        r=new_func(b); // 7, the value of variable "b",  is sent to the function

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

int new_func( int y)
{       int b =1;  // this "b" variable is local to new_func()
        a++; y++; b++;
        return a+y+b;
}
```

return statement

- "return" is used in **callee** routines to return any value of any variable to the **caller** routine.
- "return" statement may or may not include an expression.
- "return" is the last statement of any function.
- "return" statement terminates the execution of the function

- examples:

```
float f(int a, char b)
    { int i;
      …..
      return i;          /* i will be converted to float and then the
                            value of i will be returned*/
     // OR return (i);
    }
```

function prototypes

- Every function needs to be declared before its usage.
- ***ANSI C standard*** provides for a new function declaration syntax called the ***function prototype***.

example:

```
#include<stdio.h>
/* prototype of "twice" function*/
double twice (double x); // OR double twice (double );

void main(void)
{    double y, r;
     r= twice(y); // twice()is used in this line
     printf("result is %f\n", r);
}

/* definition of "twice" function*/
double twice (double x)
{  return x*2;}
```

function declarations from the compiler's viewpoint

- int f(double x) // ANSI C style
  {…..}
  /* The compiler knows about the parameter list. f(1) works properly. When an int gets passed as an argument, it will be converted to a double. */

function invocation (function call)

- **İf 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 function my_sum()
   printf("%d\n" ,n); }

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

developing large programs (1)

```
/* myHeader.h */

#define myPI 3.14

int mySum(int, int);
int mySubtract(int, int);
```

```
/* myProgram.c */
#include<stdio.h>
#include"my_header.h"

void main(void)
{
    int a=3, b=4, result=0;
    result=mySum(a,b);
    printf("%d\n",result);
    result=mySubtract(a,b);
    printf("%d\n",result);
    printf("%f\n",myPI);
}

int mySum(int x,int y)
{
    x++;
    y++;
    return x+y;
}
int mySubtract(int x, int y)
{
    return x-y-1;
}
```

developing large programs (2)

- 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

An example to implement a meaningful function definition

Good implementation: simple and proper definition
Good comments: what (input,output), how(body)

Example: iterative factorial function

```
/*factorial function gets an integer and returns the factorial of it*/
int factorial (int n)
{
    int product=1;

    // n! = n*(n-1)*(n-2)*...*2
    for( ; n>1 ; --n)
        product=product*n;

    return product;  // product is n!
}
```

TO DO (at home)

- Implement an "isPrime" function, which takes an int as an argument and returns 1 if the number is prime, 0 otherwise.
- Example prototype:
  - int isPrime(int);