

Chapter 06

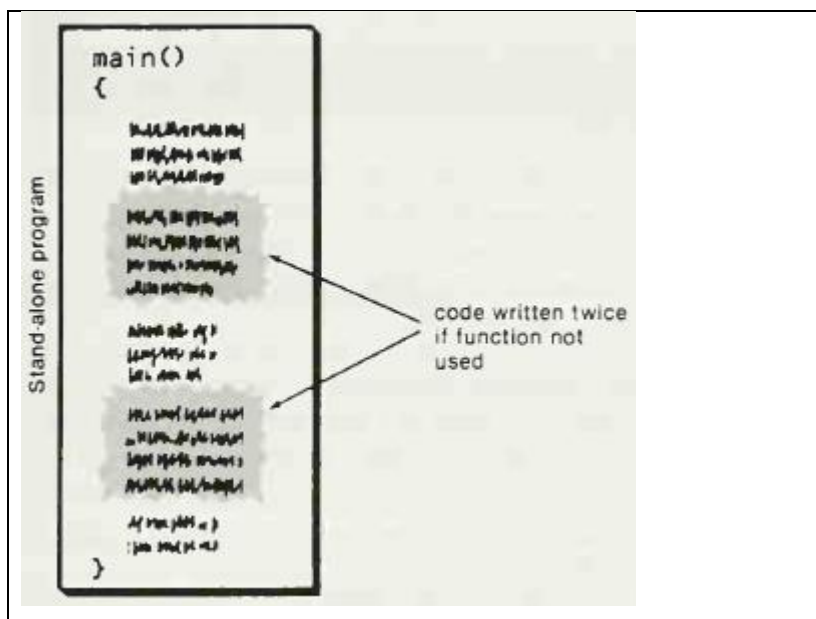
Functions

■ function

- No one can perform all of life's tasks personally. You may ask a repair person to fix your TV set, hire someone to mow your lawn, or rely on a store to provide fresh vegetables rather than growing your own.
- A computer program (except for a very simple one) is in much the same situation; it can't handle every task alone. Instead, it calls on other program-like entities — called **functions** in C—to carry out specific tasks.

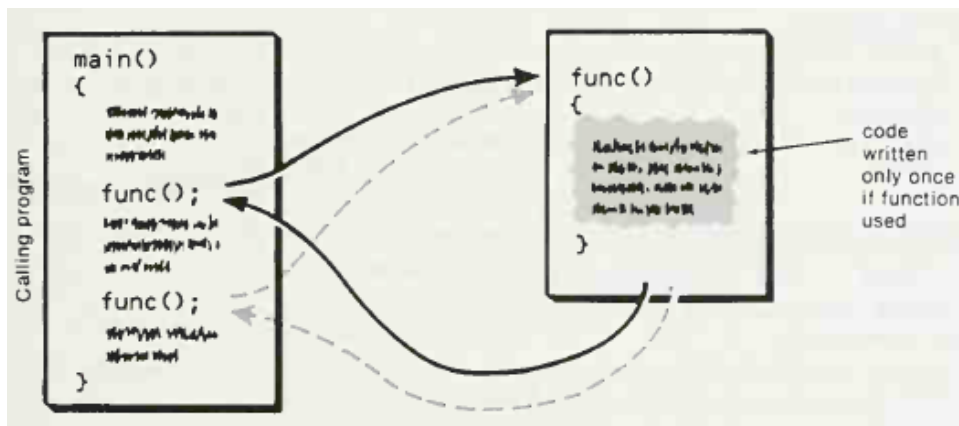
■ Why functions?

Avoid having to write the same code over and over



Chapter 06

Functions



- Easier to organize programs and keep track of what they were doing
 - Easier to debug
 - Programmer do not need to worry about using accidentally using the same variables names in different functions
- ✂ Modules in C are called functions
- ✂ Pre-packaged function → C standard library

▣ Mathematical ▣ String manipulation ▣ Character ▣ Input / Output	Avoid reinventing the wheel
▣ Example: Printf — scanf — pow	

■ Programmer defined function

- Your own function
- A boss asks a worker to perform a task and report back when the task is done
 - Boss → calling function — caller
 - Report back → Return
- Using a function is in some ways like hiring someone to perform a specific job for you
- There are three program elements involved in using function
 - **Function definition**
 - **The call to function**
 - **Function prototype**

```
#include <stdio.h>
#include <stdlib.h>
void line (void);

main()
{
    line();
    printf("\xdb Mostafa A. Elhosseini \xdb\n");
    line();
}

void line (void)
{
    int j;
    for (j = 1; j<26; j++)
        printf("\xdb");
    printf("\n");
}
```

function prototype - declarator

function calling

function definition

- **Function definition** is the function itself
 - It starts with a line called declarator
 - **void line (void)**
 - the first void means that line () does not return any thing
 - the second means that it takes no arguments
- **Calling the function**
 - as the same as any function, you call it using the name and parentheses
 - line ()
 - this calling causes control to be transferred to the code in the definition of line () and then return back control to main () to the statement immediately following the function call
- **function prototype (Declaration)**
 - This is the line before the beginning of main ()
 - This looks very much like the declarator line at the start of the function definition, except that it ends with a semicolon.
 - The function declaration tells the compiler the name of the function, the data type the function returns (if any), and the number and data types of the function's arguments (if any)

- **void line (void);**

■ local variable

- variable used in function is known only to that function — it is invisible to main ()
 - add this line to main ()
`printf("%d",j)`
and you will get unhappy message → compiler error
 - variables defined in a function are unknown outside the function
- **visibility** → which function know about a variable and which do not
- local variable is called an automatic variable — automatically created when a function is called and destroyed when the function returns

■ Example

Two beep function

```
#include <stdlib.h>

void twobeep (void);

main()
```

```
{
    twobeep();
    printf("\nType any character: ");
    getche();
    twobeep();
}

void twobeep (void)
{
    long j;
    printf("\x7");
    for(j=1;j<100000;j++) ;
    printf("\x7");
}
```

■ Function that return a value

Get lower character

```
#include <stdio.h>
#include <stdlib.h>
char getlc (void);

main()
{
    char ch;
    printf("\nType 'a' for first selection, 'b'
for second: ");
```

```
    ch = getlc();
    switch (ch)
    {
        case 'a':
            printf("\nYou typed a");
            break;
        case 'b':
            printf("\nYou typed b");
            break;
        default:
            printf("\nYou      chose      a      non-existent
selection");
    }
}

char getlc (void)
{
    char ch;
    ch = getchar();
    if(ch >64 && ch < 91)
        ch += 32;
    return (ch);
}
```

■ Function that take and return a value**Get lower character**

```
#include <stdio.h>
#include <stdlib.h>
char getlc (char);

main()
{
    char chg,ch;
    printf("\nType 'a' for first selection, 'b'
for second: ");
    chg = getchar();
    ch = getlc(chg);
    switch (ch)
    {
    case 'a':
        printf("\nYou typed a");
        break;
    case 'b':
        printf("\nYou typed b");
        break;
    default:
        printf("\nYou      chose      a      non-existent
selection");
    }
}
```

```
}  
char getlc (char c)  
{  
    if(c >64 && c < 91)  
        c += 32;  
    return (c);  
}
```

- ✍ It is preferred to put function definition, and standard header files in your own header file "Mostafa.h" → not <Mostafa.h>

Get lower character using header file

1. Main ()

```
#include "CIEH.h"  
main()  
{  
    char ch;  
    printf("\nType 'a' for first selection, 'b' for second: ");  
    ch = getlc();  
    switch (ch)  
    {
```

```
case 'a':
    printf("\nYou typed a");
    break;
case 'b':
    printf("\nYou typed b");
    break;
default:
    printf("\nYou      chose      a      non-existent
selection");
    }
}
```

**Get lower character using header
file**

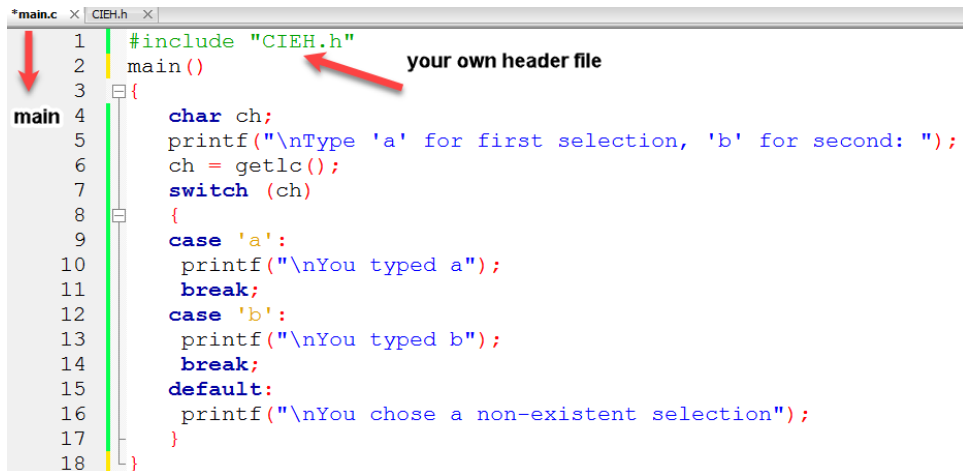
2. CIEH.h

```
#ifndef CIEH_H_INCLUDED
#define CIEH_H_INCLUDED
#include <stdio.h>
#include <stdlib.h>
char getlc (void)
{
    char ch;
    ch = getchar();
    if(ch >64 && ch < 91)
        ch += 32;
    return (ch);
}
```

Chapter 06

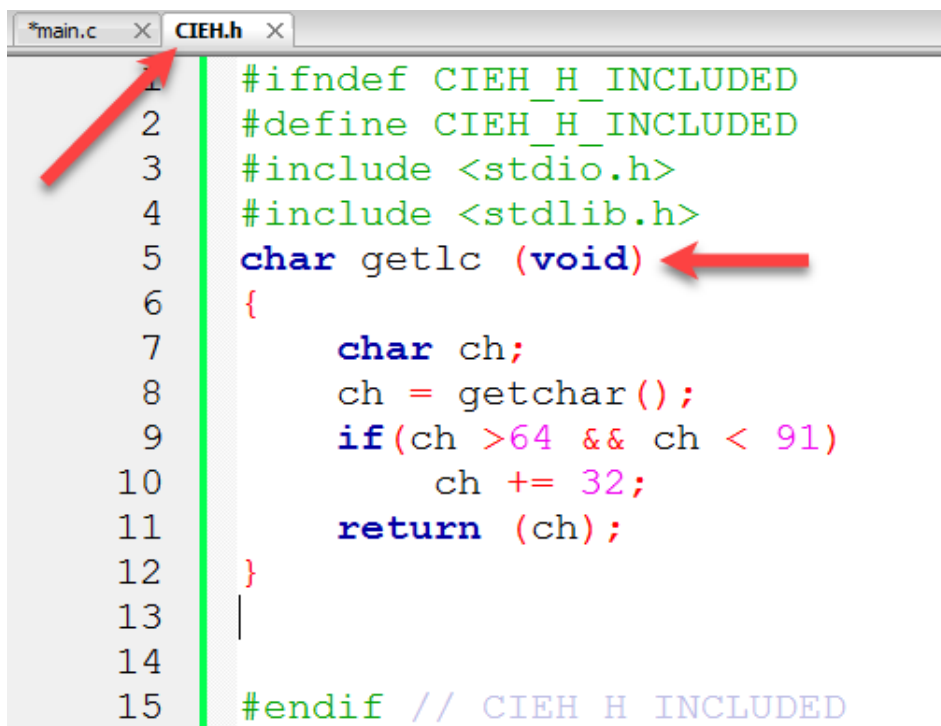
Functions

```
}  
#endif // CIEH_H_INCLUDED
```



```
*main.c x | CIEH.h x  
1  #include "CIEH.h"  
2  main()  
3  {  
4      char ch;  
5      printf("\nType 'a' for first selection, 'b' for second: ");  
6      ch = getlc();  
7      switch (ch)  
8      {  
9          case 'a':  
10         printf("\nYou typed a");  
11         break;  
12         case 'b':  
13         printf("\nYou typed b");  
14         break;  
15         default:  
16         printf("\nYou chose a non-existent selection");  
17     }  
18 }
```

your own header file



```
*main.c x | CIEH.h x  
1  #ifndef CIEH_H_INCLUDED  
2  #define CIEH_H_INCLUDED  
3  #include <stdio.h>  
4  #include <stdlib.h>  
5  char getlc (void)  
6  {  
7      char ch;  
8      ch = getchar();  
9      if(ch > 64 && ch < 91)  
10         ch += 32;  
11     return (ch);  
12 }  
13 |  
14  
15 #endif // CIEH_H_INCLUDED
```

Difference between times —

main ()

```
#include <stdio.h>
#include <stdlib.h>
#include "CIEgetMins.h"

main()
{
    int mins1,mins2;
    printf("\nEnter first time (form 3:22): ");
    mins1 = getMins();
    printf("\nEnter second time (form 3:22): ");
    mins2 = getMins();
    printf("\nDifference = %d",mins2-mins1);
}
```

Header file — CIEgetMins.h

```
#ifndef CIEGETMINS_H_INCLUDED
#define CIEGETMINS_H_INCLUDED
int getMins (void)
{
    int h,m;
    scanf("%d:%d",&h,&m);
    return (h*60+m);
}

#endif // CIEGETMINS_H_INCLUDED
```

 D:\NileProg\function04\bin\Debug\function04.exe

Enter first time (form 3:22): 3:50

Enter second time (form 3:22): 4:55

Difference = 65

Process returned 16 (0x10) execution time : 21.941 s

Press any key to continue.

Sphere Area — main ()

```
#include <stdio.h>
#include <stdlib.h>
#include "circleH.h"
main()
{
    float answer;
    printf("\nEnter sphere radius: ");
    answer = area();
    printf("\nSphere area = %f",answer);
}
```

Header file — circle.h

```
#ifndef CIRCLEH_H_INCLUDED
#define CIRCLEH_H_INCLUDED
const float PI = 3.14159;
float area (void)
{
    float r;
    scanf("%f",&r);
```

```
        return(4*PI*r*r);  
    }  
  
#endif // CIRCLEH_H_INCLUDED
```

```
Enter sphere radius: 3  
Sphere area = 113.097244  
Process returned 25 (0x19)   execution time : 1.590 s  
Press any key to continue.
```

■ Limitation to return

- Before we move on to other things, you should be aware of a key limitation in the use of the **return** statement — you can only use it to return one value. If you want your function to return two or more values to the calling program, you need another mechanism named POINTER.
- Using a **return** statement, only one value can be returned by a function.
- Using **arguments**, it's possible to pass more than one piece of information to a function.

■ Using arguments to pass data to a function

- The mechanism used to convey information to a function is the **argument**.
- You've already used arguments in `printf ( )` and `scanf ( )` functions
 - the format strings and the values used inside the parentheses in these functions are **arguments**.

Bar graph

```
#include <stdio.h>
#include <stdlib.h>
void bar(int);
main()
{
    printf("SpongeBob\t");
    bar(42);
    printf("PatrickStar\t");
    bar(34);
    printf("SandyCheeks\t");
    bar(10);
    printf("GaryTheSnail\t");
    bar(8);
    printf("Sheldon..\t");
    bar(6);
```

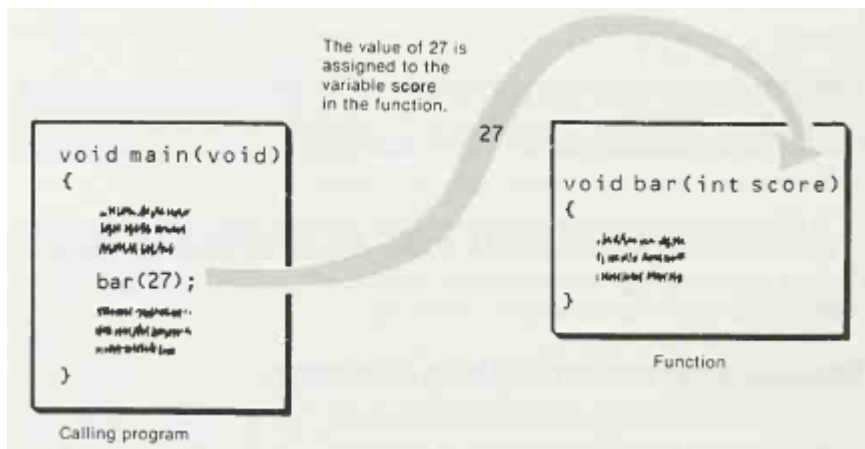
```
}  
void bar(int d)  
{  
    int j;  
    for(j=1;j<=d;j++)  
        printf("\xcd");  
    printf("\n");  
}
```

SpongeBob	=====
PatrickStar	=====
SandyCheeks	=====
GaryTheSnail	=====
Sheldon..	=====

- the number we want to pass to the function bar () is included in the parentheses
 - we may use a variable name instead of the number
- In the function definition, a variable name score is placed in the parentheses following bar.
 - void bar(int score)
- This ensures that the value included between parentheses in the main program is assigned to the variable between parentheses in the function definition. This is shown below

Chapter 06

Functions



- Notice that the data type of `score` is also specified in the function definition.
- This constitutes a definition of the argument variable, setting aside memory space for it in the function.

Bar graph — variable

```
#include <stdio.h>
#include <stdlib.h>

void bar(int);

main()
{
    int inscore;
    inscore = 1;
    printf("\n");
    while(inscore != 0)
    {
        printf("score = ");
```

```
        scanf("%d",&inscore);
        bar(inscore);
    }
}

void bar(int d)
{
    int j;
    for(j=1;j<=d;j++)
        printf("\xcd");
    printf("\n");
}
```

```
score = 20
```

```
score = 15
```

```
score = 7
```

```
score = 50
```

```
score = 17
```

```
score = 0
```

- Different names have been assigned to the arguments in the calling and called functions: **inscore** in the calling program and **d** in the function.
 - Actually, we could have used the same name for both variables; because they're in different functions, the

compiler would still consider them to be separate variables.

- Note that the argument in the calling program is referred to as the **actual argument**,
- while the argument in the called function is the **formal argument**. In this case, the variable **inscore** in the calling program is the actual argument, and the variable **d** in the function is the formal argument.

■ Passing multiple arguments

Rectangle

```
#include <stdio.h>
#include <stdlib.h>
void rectang(int, int);
main()
{
    int length, width;
    printf("\nLiving Room\n");
    rectang(22,12);
    printf("\nCloset\n");
    rectang(4,4);
    printf("\nKitchen\n");
    rectang(16,16);
}
```

```
printf("\nBathroom\n");
rectang(6,8);
printf("\nBedroom\n");
rectang(12,12);

}

void rectang(int l,int w)
{
    int j,k;
    l /= 2;
    w /= 2;
    for(j =1;j<=l;j++)
    {
        printf("\t\t");
        for(k=1;k<=w;k++)
            printf("\xdb");
        printf("\n");
    }
}
```

Chapter 06

Functions

Closet



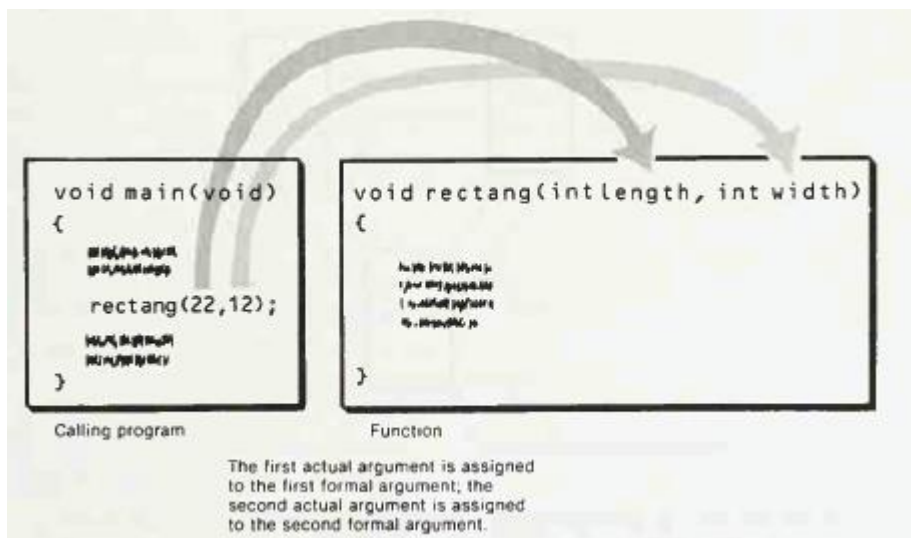
Kitchen



Bathroom



Bedroom



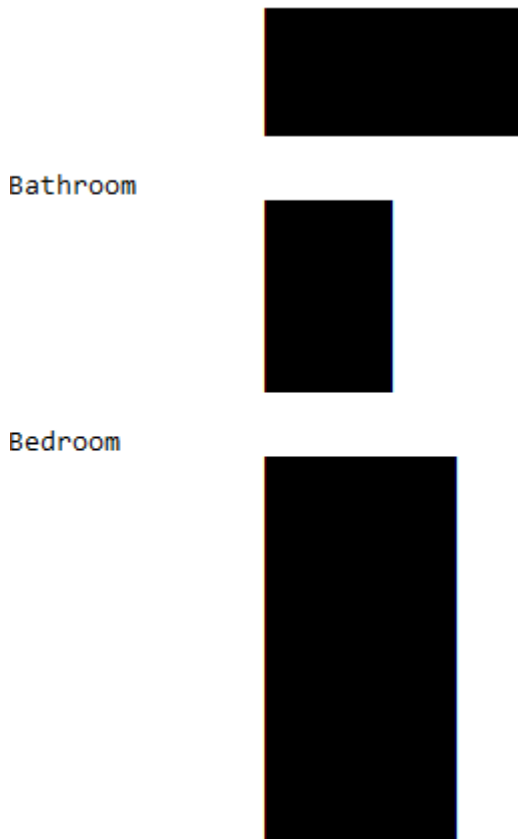
- **Scale factors** → These are necessary so that the room dimensions can be expressed in feet. The function divides the length (the horizontal dimension on the screen) by 2 and the width (the vertical dimension) by 4.
- This makes it possible for a series of typical rooms to fit on the screen and also compensates for the fact that a **character on the screen is twice as high as it is long**. To make a square room look square, we must divide its vertical dimension by twice as much as its horizontal dimension.

Rectangle – without scaling

```
#include <stdio.h>
#include <stdlib.h>
void rectang(int, int);
main()
{
    int length, width;
    printf("\nLiving Room\n");
    rectang(22,12);
    printf("\nCloset\n");
    rectang(4,4);
    printf("\nKitchen\n");
    rectang(16,16);
    printf("\nBathroom\n");
    rectang(6,8);
```

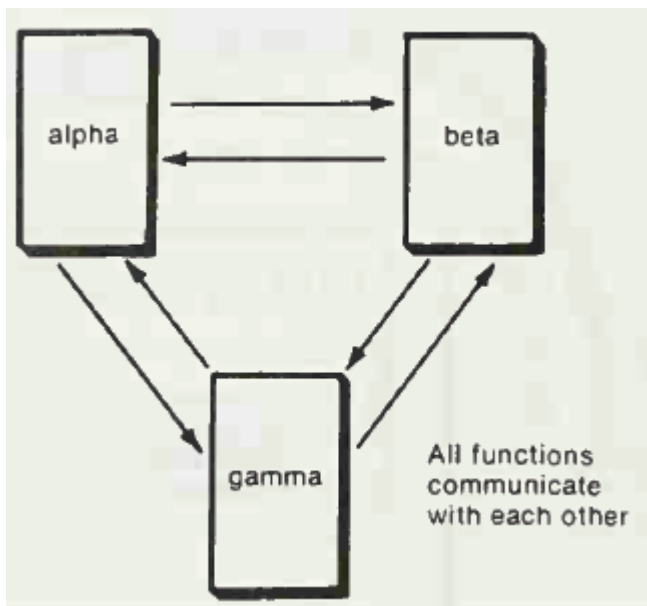
```
printf("\nBedroom\n");
rectang(12,12);

}
void rectang(int l,int w)
{
    int j,k;
    for(j =1;j<=l;j++)
    {
        printf("\t\t");
        for(k=1;k<=w;k++)
            printf("\xdb");
        printf("\n");
    }
}
```



■ Using more than one function

- You can use as many functions as you like in a program, and any of the functions can call any of the other functions.
- In C, however, all functions are visible to all other functions.



- In C, all functions, including `main()`, have equal status and are visible to all other functions.
- Example:

Let's look at a program involving several functions. This program calculates the sum of the squares of two integers typed by the user. The program actually uses three functions as well as `main ( )`. The first function does the actual calculation, while `main ( )` simply gets the numbers from the user and prints the result. The second function returns the square of a number (the number multiplied by itself), and the third function returns the sum of two numbers.

Sum of square

```
#include <stdio.h>
#include <stdlib.h>
int sumsqr(int,int);
int sqr(int);
int sum(int, int);
main()
{
    int num1,num2;
    printf("Type two numbers: ");
    scanf("%d %d",&num1,&num2);
    printf("sum      of      the      squares      is
%d",sumsqr(num1,num2));
}
int sumsqr(int j,int k)
{
    return(sum(sqr(j),sqr(k)));
}
int sum(int x,int y)
{
    return (x+y);
}
int sqr(int z)
{
    return(z*z);
}
```

```
Type two numbers: 3 4
sum of the squares is 25
```

Another version — sum of squares using my own header file

Sum of square --- main

```
#include <stdio.h>
#include <stdlib.h>
#include "zeyad.h"

main()
{
    int num1,num2;
    printf("Type two numbers: ");
    scanf("%d %d",&num1,&num2);
    printf("sum      of      the      squares      is
%d",sumsqr(num1,num2));
}
```

Sum of square --- Header file zeyad.h

```
#ifndef ZEYAD_H_INCLUDED
#define ZEYAD_H_INCLUDED
int sumsqr(int j,int k)
{
    return(sum(sqr(j),sqr(k)));
}
int sqr(int z)
{
    return(z*z);
}
```

Chapter 06

Functions

```
}  
int sum(int x, int y)  
{  
    return (x+y);  
}  
  
#endif // ZEYAD_H_INCLUDED
```

