



Computer Programming (a) - E1123

(Fall 2021-2022)

Lecture 7

Functions-Part2

INSTRUCTOR

DR / AYMAN SOLIMAN



➤ Contents

- 1) Examples
- 2) Introduction
- 3) Predefined Functions
- 4) user-defined Functions
- 5) Function Call



Functions

```
graph TD; A[Functions] --> B[predefined]; A --> C[user-defined]
```

predefined

user-defined

➤ Why functions are useful

- **Organization** → As programs grow in complexity, having all the code live inside the main() function becomes increasingly complicated. A function is almost like a mini-program that we can write separately from the main program, without having to think about the rest of the program while we write it. This allows us to reduce a complicated program into smaller, more manageable chunks, which reduces the overall complexity of our program.
- **Reusability** → Once a function is written, it can be called multiple times from within the program. This avoids duplicated code (“**Don’t Repeat Yourself**”) and minimizes the probability of copy/paste errors. Functions can also be shared with other programs, reducing the amount of code that must be written from scratch (and retested) each time.

➤ Why functions are useful

- **Testing** → Because functions reduce code redundancy, there's less code to test in the first place. Also, because functions are self-contained, once we've tested a function to ensure it works, we don't need to test it again unless we change it.
- **Extensibility** → When we need to extend our program to handle a case it didn't handle before; functions allow us to make the change in one place and have that change take effect every time the function is called.
- **Abstraction** → In order to use a function, you only need to know its name, inputs, outputs, and where it lives. You don't need to know how it works, or what other code it's dependent upon to use it. This lowers the amount of knowledge required to use other people's code (including everything in the standard library).

➤ Functions Scope Variables

- Function parameters, as well as variables defined inside the function body, are called **local variables**

```
#include <iostream.h>
int add(int x, int y) // function parameters x and y are local variables
{
    return x+y;
}
int main()
{
    cout<<add(3,4)<<endl;
    return 0;
}
```

 "C:\Users\Dr Ayman\
?

Press any key to

In this lesson, we'll look at some properties of local variables in more detail

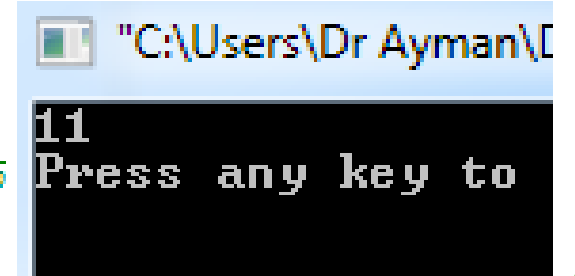
➤ Local variable lifetime

```
int add(int x, int y)           // function parameters x and y are initialized
{
    return x+y;
}                               // x and y destroyed here
```

- Much like a person's lifetime is defined to be the time between their birth and death, an object's lifetime is defined to be the time between its **creation** and **destruction**.

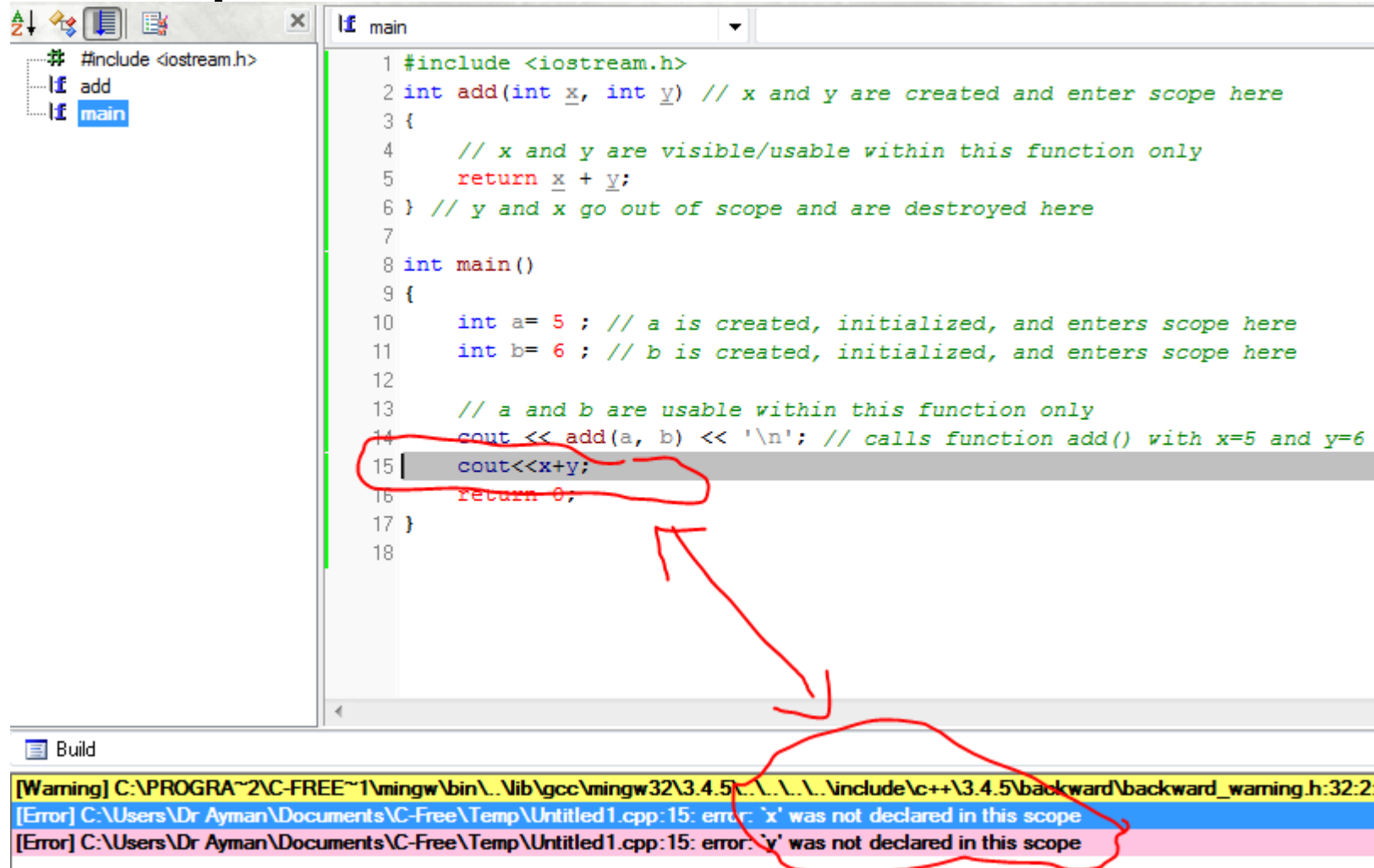
➤ Local scope

```
1 #include <iostream.h>
2 int add(int x, int y) // x and y are created and enter scope here
3 {
4     // x and y are visible/usable within this function only
5     return x + y;
6 } // y and x go out of scope and are destroyed here
7
8 int main()
9 {
10     int a= 5 ; // a is created, initialized, and enters scope here
11     int b= 6 ; // b is created, initialized, and enters scope here
12
13     // a and b are usable within this function only
14     cout << add(a, b) << '\n'; // calls function add() with x=5 and y=6
15     return 0;
16 }
```



```
"C:\Users\Dr Ayman\...
11
Press any key to
```


➤ Local scope



The screenshot shows a C++ IDE with a file named 'main'. The code defines a function 'add' and a 'main' function. The 'main' function calls 'add(a, b)' and then attempts to print 'x+y' and return 0. A red circle highlights the 'cout<<x+y;' line, and a red arrow points from it to the error messages in the build window. The error messages indicate that 'x' and 'y' are not declared in the scope of the 'main' function.

```
1 #include <iostream.h>
2 int add(int x, int y) // x and y are created and enter scope here
3 {
4     // x and y are visible/usable within this function only
5     return x + y;
6 } // y and x go out of scope and are destroyed here
7
8 int main()
9 {
10    int a= 5 ; // a is created, initialized, and enters scope here
11    int b= 6 ; // b is created, initialized, and enters scope here
12
13    // a and b are usable within this function only
14    cout << add(a, b) << '\n'; // calls function add() with x=5 and y=6
15    cout<<x+y;
16    return 0;
17 }
18
```

Build

[Warning] C:\PROGRA~2\C-FREE~1\mingw\bin\..lib\gcc\mingw32\3.4.5\..\.include\c++\3.4.5\backward\backward_warning.h:32:2:
[Error] C:\Users\Dr Ayman\Documents\C-Free\Temp\Untitled1.cpp:15: error: 'x' was not declared in this scope
[Error] C:\Users\Dr Ayman\Documents\C-Free\Temp\Untitled1.cpp:15: error: 'y' was not declared in this scope

➤ Example

```
#include <iostream.h>
```

```
int add(int x, int y)
{
    return x + y;
}
```

```
int multiply(int z, int w)
{
    return z * w;
}
```

```
int main()
{
    cout << add(4, 5) << endl;
    cout << multiply(2, 3) << endl;
    cout << add(1 + 2, 3 * 4) << endl;

    int a = 5;
    cout << add(a, a) << endl;
    cout << add(1, multiply(2, 3)) << endl;
    cout << add(1, add(2, 3)) << endl;
    return 0;
}
```

9
6
15
10
7
6

➤ Forward declarations and definitions

```
1 #include <iostream.h>
2
3 int main()
4 {
5     cout << "The sum of 3 and 4 is: " << add(3, 4) << '\n';
6     return 0;
7 }
8
9 int add(int x, int y)
10 {
11     return x + y;
12 }
```

Build

Compiling C:\Users\Dr Ayman\Documents\C-Free\Temp\Untitled2.cpp...

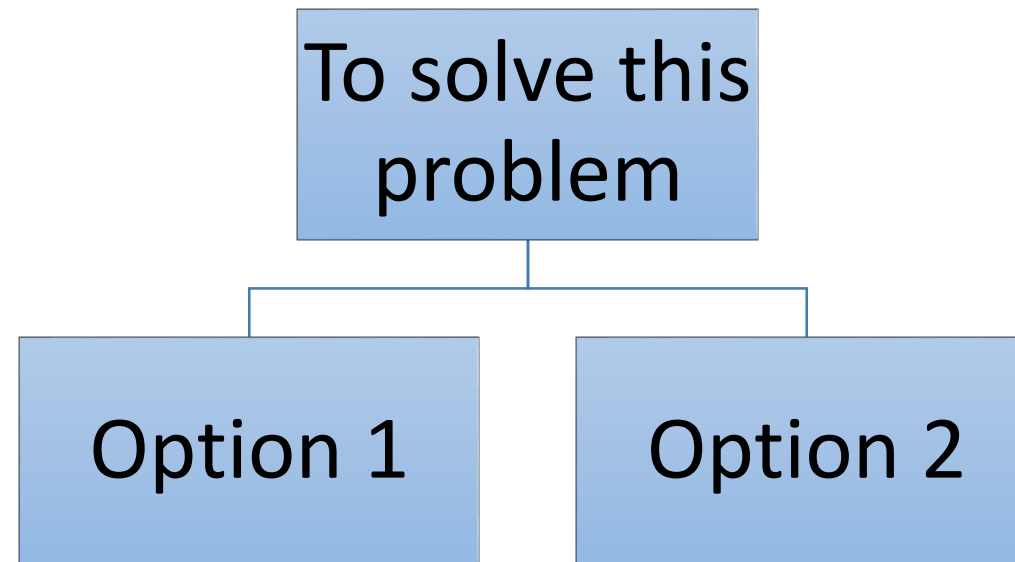
[Warning] C:\PROGRA~2\C-FREE~1\mingw\bin\..\lib\gcc\mingw32\3.4.5\..\..\include\c++\3.4.5\backward\backward_warning.h:32:2:

[Error] C:\Users\Dr Ayman\Documents\C-Free\Temp\Untitled2.cpp:5: error: 'add' was not declared in this scope

➤ Forward declarations and definitions (cont.)

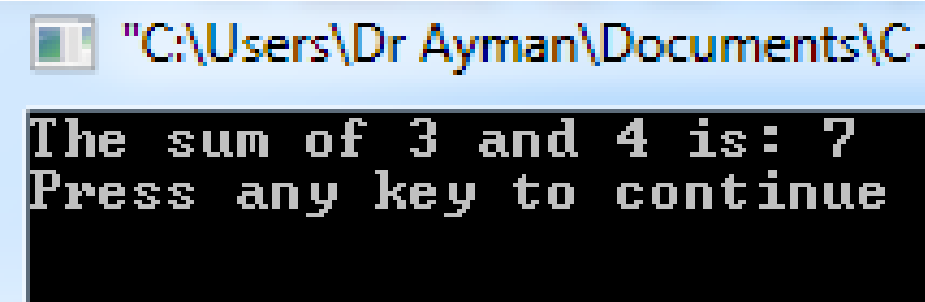
In fact, it doesn't compile at all! Visual Studio produces an error:

- The reason this program doesn't compile is because the compiler compiles the contents of code files sequentially. When the compiler reaches the function call to add on line 5 of main, it doesn't know what add is, because we haven't defined add until line 9! That produces the error, identifier not found.



➤ Option 1: Reorder the function definitions

```
1 #include <iostream.h>
2 int add(int x, int y)
3 {
4     return x + y;
5 }
6 int main()
7 {
8     cout << "The sum of 3 and 4 is: " << add(3, 4) << '\n';
9     return 0;
10 }
```



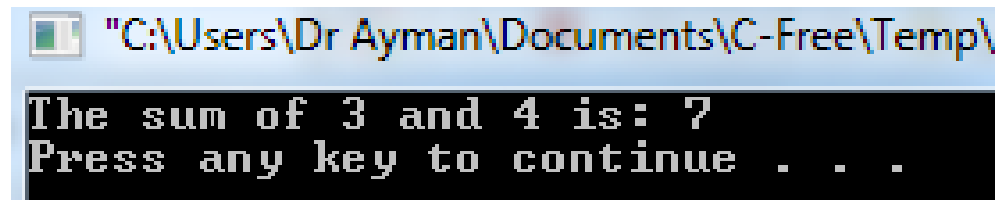
"C:\Users\Dr Ayman\Documents\C-

```
The sum of 3 and 4 is: 7
Press any key to continue
```

➤ Option 2: Use a forward declaration

❑ A **forward declaration** allows us to tell the compiler about the existence of an identifier before defining the identifier.

```
1 #include <iostream.h>
2
3 int add(int x, int y); // forward declaration of add() (using a function prototype)
4
5 int main()
6 {
7     cout << "The sum of 3 and 4 is: " << add(3, 4) << '\n'; // this works because we forward declared add() above
8     return 0;
9 }
10 int add(int x, int y) // even though the body of add() isn't defined until here
11 {
12     return x + y;
13 }
```

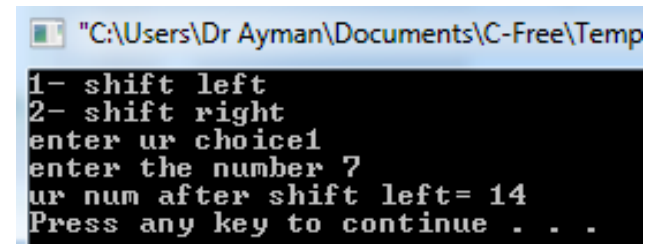


```
"C:\Users\Dr Ayman\Documents\C-Free\Temp\  
The sum of 3 and 4 is: 7  
Press any key to continue . . .
```

- Example Write the program that ask the user to print shift left or shift right to any number depending on your choice.

```
1 #include <iostream.h>
2 void sh_lf(int x);
3 void sh_rh(int x);
4
5 int main()
6     {
7     int num;
8     cout <<"1- shift left"<<"\n";
9     cout <<"2- shift right"<<"\n";
10    cout << "enter ur choice";
11    cin >> num;
12    switch (num) {
13    case 1:sh_lf(num);|
14    break;
15    case 2:sh_rh(num);
16    break;
17    }
18    cout<<endl;
19    }
```

```
20
21 void sh_lf(int x)
22 {
23 int y;
24 cout <<"enter the number ";
25 cin >> x;
26 y= (x << 1);
27 cout << "ur num after shift left= "<<y;
28 }
29
30 void sh_rh(int x)
31 {
32 int y;
33 cout <<"enter the number ";
34 cin >> x;
35 y= x >> 1;
36 cout << "ur num after shift right= "<<y;
37 }
```



```
"C:\Users\Dr Ayman\Documents\C-Free\Temp
1- shift left
2- shift right
enter ur choice1
enter the number 7
ur num after shift left= 14
Press any key to continue . . .
```

Thank
you

