Computer Programming (a)

E1123

Fall 2022-2023

Lecture 7

Functions

INSTRUCTOR

DR / AYMAN SOLIMAN

➢ Contents

➢Introduction

<u>In this Lecture, you will:</u>

- ➢ Learn about standard (predefined) functions and discover how to use them in a program
- ➢ Learn about user-defined functions
- ➢ Examine value-returning functions, including actual and formal parameters
- ➢ Explore how to construct and use a value-returning, user-defined function in a program

➢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

- predefined
- user-defined

➢Ex

- In algebra, a function is defined as a rule or correspondence between values, called the function's arguments, and the unique value of the function associated with the arguments

  - If $f(x) = 2x + 5$, then $f(1) = 7$, $f(2) = 9$, and $f(3) = 11$

    - 1, 2, and 3 are arguments

    - 7, 9, and 11 are the corresponding values

➢Predefined Functions

Some of the predefined mathematical functions are:

sqrt(x)

pow(x, y)

`pow(x,y)` calculates x^y

`pow(2, 3) = 8.0`

Returns a value of type `double`

`x` and `y` are the parameters (or arguments)

The function has two parameters

`sqrt(x)` calculates the nonnegative square root of `x`, for `x >= 0.0`

`sqrt(2.25)` is `1.5`

Type `double`

Function	Header File	Purpose	Parameter(s) Type	Result
abs(x)	<cstdlib>	Returns the absolute value of its argument: abs(-7) = 7	int	int
ceil(x)	<cmath>	Returns the smallest whole number that is not less than x: ceil(56.34) = 57.0	double	double
cos(x)	<cmath>	Returns the cosine of angle x: cos(0.0) = 1.0	double (radians)	double
exp(x)	<cmath>	Returns e^x, where e = 2.718: exp(1.0) = 2.71828	double	double
fabs(x)	<cmath>	Returns the absolute value of its argument: fabs(-5.67) = 5.67	double	double

➢Predefined Functions

```
#include <iostream>
#include <cmath> // for sqrt and pow
using namespace std;

int main()
{
    double number, squareRoot;
    cout << "Enter a number: ";
    cin >> number;
    // sqrt() is a library function to calculate square root
    squareRoot = sqrt(number);
    double power=pow(number,3);
    cout << "Square root of " << number << " = " << squareRoot;
    cout << endl;
    cout <<  number << " ^ 3 = " << power;
    return 0;
}
```

```
Enter a number: 2.5
Square root of 2.5 = 1.58114
2.5 ^ 3 = 15.625
```

➢Predefined Functions

```
#include <iostream>
#include <cmath>
#include <cctype>
#include <cstdlib>

using namespace std;

int main()
{
    int   x;
    double u, v;

    cout << "Line 1: Uppercase a is "
         << static_cast<char>(toupper('a'))
         << endl;                                    //Line 1

    u = 4.2;                                         //Line 2
    v = 3.0;                                         //Line 3
    cout << "Line 4: " << u << " to the power of "
         << v << " = " << pow(u, v) << endl;         //Line 4

    cout << "Line 5: 5.0 to the power of 4 = "
         << pow(5.0, 4) << endl;                     //Line 5

    u = u + pow(3.0, 3);                             //Line 6
    cout << "Line 7: u = " << u << endl;             //Line 7

    x = -15;                                         //Line 8
    cout << "Line 9: Absolute value of " << x
         << " = " << abs(x) << endl;                 //Line 9

    return 0;
}
```

```
Line 1: Uppercase a is A
Line 4: 4.2 to the power of 3 = 74.088
Line 5: 5.0 to the power of 4 = 625
Line 7: u = 31.2
Line 9: Absolute value of -15 = 15
```

➢user-defined Functions

Syntax:

```
functionType functionName(formal parameter list)
{
    statements
}
```

`functionType` is also called the data type or return type

➢Function Call

```
functionName(actual parameter list)
```

➢Return Statement

Once a value-returning function computes the value, the function returns this value via the return statement

It passes this value outside the function via the return statement

The return statement has the following syntax:

In C++, return is a reserved word

When a return statement executes

Function immediately terminates

Control goes back to the caller

When a return statement executes in the function main, the program terminates

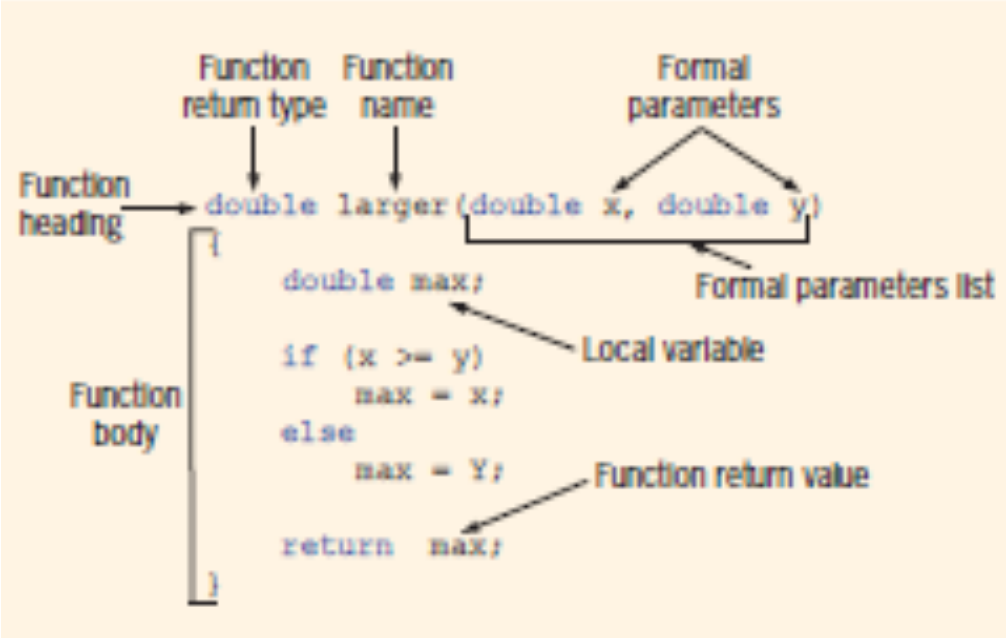
➢user-defined Functions

```
double larger(double x, double y)
{
    double max;

    if (x >= y)
        max = x;
    else
        max = y;

    return max;
}
```

You can also write this function as follows:

```
double larger(double x, double y)
{
    if (x >= y)
        return x;
    else
        return y;
}
```

```
double larger(double x, double y)
{
    if (x >= y)
        return x;

    return y;
}
```



➢user-defined Functions

```
//Program: Largest of three numbers

#include <iostream>

using namespace std;

double larger(double x, double y);


int main()
{
   cout << "Line 2: The larger of 5 and 10 is "
        << larger(5, 10) << endl;                    //Line 2

    return 0;
}

double larger(double x, double y)
{
    if (x >= y)
        return x;
    else
        return y;
}
```

```
double larger(double x, double y)
{
    if (x >= y)
        return x;
    else
        return y;
}

double compareThree (double x, double y, double z)
{
    return larger(x, larger(y, z));
}
```

Sample Run: In this sample run, the user input is shaded.

```
Line 2: The larger of 5 and 10 is 10
Line 3: Enter two numbers: 25 73

Line 6: The larger of 25 and 73 is 73
Line 7: The largest of 23, 34, and 12 is 34
```

➢user-defined Functions

```
#include <iostream>
using namespace std;
// Function definition
void welcome()
{
 cout <<"Enter your first name: ";
 string name;
 cin >> name;
  cout<<"Hey "<< name << "!";
}

int main()
{
    welcome();
    return 0;
}
```

```
Enter your first name: Sayed
Hey Sayed!
```

➢user-defined Functions

The **return type** is the type declared before the function name. Note that the return type does not indicate what specific value will be returned. It only indicates what type of value will be returned.

```
#include<iostream>
double return_value()
{
    return 3.2;
}

int main()
{
    std::cout << return_value() << std::endl;
    return 0;
}
```

3.2

```
#include<iostream>
int return_value()
{
    return 3.2;
}

int main()
{
    std::cout << return_value() << std::endl;
    return 0;
}
```

3

➢user-defined Functions

```
#include<iostream>
char return_value()
{
    return 'C';
}

int main()
{
    std::cout << return_value() << std::endl;
    return 0;
}
```

C

```
#include<iostream>
int return_value()
{
    return 'C';
}

int main()
{
    std::cout << return_value() << std::endl;
    return 0;
}
```

67


Thank
you