C++ Programming I

Functions

C++ Programming
March 1, 2018

Dr. P. Arnold
Bern University of Applied Sciences

Agenda

- **Functions**
  - Need of Functions
  - Function Syntax
  - Overloading Functions
  - Passing Data to Functions
  - Default Parameters
  - Lambda Function

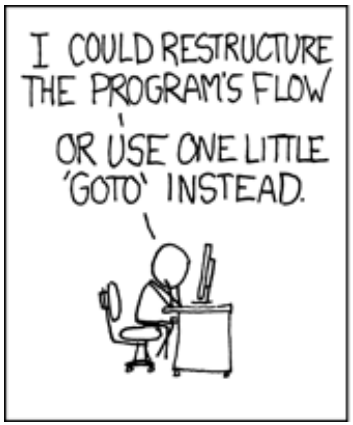
Agenda

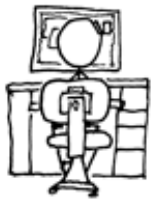
- **Functions**
  - Need of Functions
  - Function Syntax
  - Overloading Functions
  - Passing Data to Functions
  - Default Parameters
  - Lambda Function
- **Outlook and Homework**

Functions

Need of Functions

- Functions are used to provide modularity to a program, to create logical blocks
- Creating an application using functions makes it easier to understand, edit, check errors and maintain
- Functions enable reusing code! So less work for us





- Think before you code!
- Choose meaningful names for variables and functions

Functions

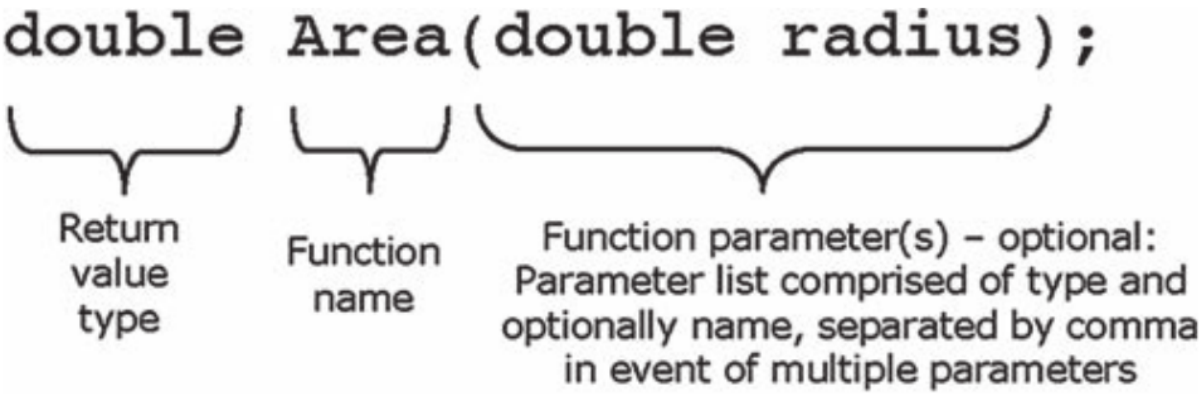
An Example

```
1  #include <iostream>
2  const double PI = 3.14159265;
3  using namespace std;
4  // Function Declarations (Prototypes)
5  double area(double radius);
6  double circumference(double radius);
7
8  int main()
9  {
10     double radius = 2.5;
11     // Call function "Area"
12    cout << "Area is: " << area(radius) << endl;
13    cout << "Area is: " << area(3.5)    << endl;
14
15    // Call function "Circumference"
16    cout << "Circumference is: " << circumference(radius) << endl;
17    return 0;
18 }
19
20 // Function definitions (implementations)
21 double area(double radius)
22 {
23    return PI * radius * radius;
24 }
25
26 double circumference(double radius)
27 {
28    return 2 * PI * radius;
29 }
```


Bern University
of Applied Sciences

Syntax of Function Declaration

Example and General



Syntax of Function Declaration

Example and General

```
// function prototype / declaration
returnType funcName(paramterType parameter);

int myFunctionA(int valA, int valB, unsigned int valC);

int myFunctionC(int, int, unsigned int);

void myFunctionD(void);
```

Syntax of Function Declaration

Example and General

```
// function prototype / declaration
returnType funcName(paramterType parameter);

int myFunctionA(int valA, int valB, unsigned int valC);

int myFunctionC(int, int, unsigned int);

void myFunctionD(void);
```

- The prototype is the interface of a function.
- Before calling a function its interface must be defined. Therefore, declare a function before calling it.
- Parameter names are optional for the prototype – *it is good practice to write them*.
- The function declaration is a statement ↪ ends by a semicolon "**;**"

Syntax of Function Declaration

Example and General

```
// function prototype / declaration
returnType funcName(paramterType parameter);

int myFunctionA(int valA, int valB, unsigned int valC);

int myFunctionC(int, int, unsigned int);

void myFunctionD(void);
```

- The prototype is the interface of a function.
- Before calling a function its interface must be defined. Therefore, declare a function before calling it.
- Parameter names are optional for the prototype – *it is good practice to write them*.
- The function declaration is a statement ↪ ends by a semicolon "**;**"
- The declaration can be either in the **source file** or in a **header file**. *Putting it in a **header file** makes function available for other source files when including the header file.*

Syntax of Function Definition

Example and General

```
1  double area(double radius)
2  {
3      return PI * radius * radius;
4  }
```

- This is the definition
- No semicolon!

Syntax of Function Definition

Example and General

```
1  // function head
2  returnType functionName(parameterName)
3  {
4      /* function body */
5  }
6
7  // function definition (head + body)
8  int myFunctionA(int valA, int valB, unsigned int valC)
9  {
10     /* Implementation */
11     return valA + valB +valC;
12 }
```

- The function head has never a semicolon at the end. If you copy it from the prototype remove semicolon.
- In the function header are all parameters listed by their unique names.
- The function body contains the implementation. The block starts and ends by curly braces (compound statement).
- Function definition = function header + function body

Overloading Functions

```
1  // Prototypes
2  double area(double radius); // for circle
3  double area(double radius, double height); // overloaded cylinder
4
5
6  // Definition for circle
7  double area(double radius)
8  {
9     return Pi * radius * radius;
10 }
11
12 // Definition Overloaded for cylinder
13 double area(double radius, double height)
14 {
15    // reuse the area of circle
16    return 2 * area (radius) + 2 * Pi * radius * height;
17 }
```

- The the compiler determines the most appropriate definition to use by comparing the argument types you have used to call the function
- The process of selecting the most appropriate overloaded function is called **overload resolution or signature matching**

Passing Data to Functions

In C++ there are three different ways to pass data to a function.
Passing:

1. by value:
   ```
   void passByValue(int value);
   ```
2. by reference:
   ```
   void passByReference(int& valueRef);
   ```
3. by pointer:
   ```
   void passByPointer(int* valuePtr);
   ```

- All have different characteristics when it comes to efficiency, storage and behaviour
- We'll focus on 1 & 2
- Passing by pointer is a legacy method used by C-style programs (or function pointers)

Passing by Value

Passing a Copy

```
1  #include <iostream>
2  using namespace std;
3
4  int square(int x);
5
6  int main()
7  {
8      int x = 2;
9
10     cout << "The square of " << x << " is "
11          << square(x) << endl;
12
13     return 0;
14 }
15
16 int square(int x)
17 {
18     return x * x;
19 }
```

- The underlying object is copied using its copy constructor
- Additional memory allocated
- Function works on the copy only!
- For large objects there will be a performance impact

Passing by Reference

Reference

```
1  #include <iostream>
2  using namespace std;
3
4  int square(int& x);
5
6  int main()
7  {
8      int x = 2;
9
10     cout << x << "^2 is " << square(x) << endl;
11
12     cout << x << "^2 is " << square(x) << endl;
13
14     return 0;
15 }
16
17 int square(int& x)
18 {
19    return  x *= x;
20 }
```

- Underlying object not copied
- The function is given the memory address of the object itself
- Original object can be modified! **Possibility of bugs!**

Passing by Reference to Const

Const Reference

```
1  #include <iostream>
2  using namespace std;
3
4  int square(const int& x);
5
6  int main()
7  {
8      int x = 2;
9
10     cout << "The square of " << x << " is "
11          << square(x) << endl;
12
13     return 0;
14 }
15
16 int square(const int& x)
17 {
18     //x *= x; // compilation error! x-cant be changed
19     return x * x;
20 }
```

- No copy AND no modification
- Interface is precise about its intent
- Efficient and safe

Use Reference

Fetching the Result of a function as Reference Parameter

Result as Reference Parameter

```
1  #include <iostream>
2  using namespace std;
3
4  void square(const int& x, int& result);
5
6  int main()
7  {
8      int x = 2;
9      int result = 0;
10
11     square(x, result);
12     cout << "The square of " << x << " is "
13          << result << endl;
14
15     return 0;
16 }
17
18 void square(const int& x, int& result)
19 {
20     result =  x * x;
21 }
```

Find the Bug

```
1  #include <iostream>
2
3  using namespace std;
4  const double Pi = 3.1416;
5
6  void Area(double radius, double result)
7  {
8      result = Pi * radius * radius;
9  }
10
11 int main()
12 {
13     cout << "Enter radius: ";
14     double radius = 0;
15     cin >> radius;
16
17     double areaFetched = 0;
18     Area(radius, areaFetched);
19
20     cout << "The area is: " << areaFetched << endl;
21     return 0;
22 }
```

- What is wrong with the code above
- In the function header are all parameters listed by their unique names.

Function Parameters with Default Values

```cpp
#include <iostream>
using namespace std;

// Function Declarations (Prototypes) with default Pi
double Area(double radius, double pi = 3.14);

int main()
{
    double radius = 2.5;
    double circleArea = 0;

    circleArea = Area(radius); // Ignore 2nd param, use default value

    double accuratePi = 3.14159265359;
    circleArea = Area (radius, accuratePi);

    // Call function "Area"
    cout << "Area is: " << circleArea << endl;

    return 0;
}

// Function definitions (implementations)
double Area(double radius, double pi)
{
    return pi * radius * radius;
}
```

Lambda Function

```
1  #include <iostream>
2  #include <algorithm>
3  #include <vector>
4
5  void DisplayNums(vector<int>& dynArray)
6  {
7     for_each (dynArray.begin(), dynArray.end(), \
8             [](int Element) {cout << Element << " ";} );
9
10    std::cout << endl;
11 }
12
13 int main()
14 {
15    std::vector<int> myNums;
16    myNums.push_back(501);
17    myNums.push_back(-1);
18    myNums.push_back(25);
19    myNums.push_back(-35);
20
21    DisplayNums(myNums);
22
23    std::cout << "Sorting them in descending order" << std::endl;
24
25    sort (myNums.begin(), myNums.end(), \
26          [](int Num1, int Num2) {return (Num2 < Num1); } );
27
28    DisplayNums(myNums);
29    return 0;
30 }
```

Outlook and Homework

Outlook and Homework

- Next time we'll look at chapter 8 of the book: **pointers and references**
- I recommend to read the book **until chapter 7** as homework!
- **Solve Exercise-02**

Thank You

Questions

???