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
- **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

I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

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

Functions

An Example

```cpp
#include <iostream>
const double PI = 3.14159265;
using namespace std;
// Function Declarations (Prototypes)
double area(double radius);
double circumference(double radius);

int main()
{
     double radius = 2.5;
     // Call function "Area"
    cout << "Area is: " << area(radius) << endl;
    cout << "Area is: " << area(3.5)     << endl;

    // Call function "Circumference"
    cout << "Circumference is: " << circumference(radius)
         << endl;
    return 0;
}

// Function definitions (implementations)
double area(double radius)
{
    return PI * radius * radius;
}

double circumference(double radius)
{
    return 2 * PI * radius;
}
```

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 $\hookrightarrow$ 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

```
double area(double radius)
{
   return PI * radius * radius;
}
```

- This is the definition
- No semicolon!

```
// function head
returnType functionName(parameterName)
{
    /* function body */
}

// function definition (head + body)
int myFunctionA(int valA, int valB, unsigned int
    valC)
{
    /* Implementation */
    return valA + valB +valC;
}
```

- 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

```
// Prototypes
double area(double radius); // for circle
double area(double radius, double height); // overloaded
    cylinder


// Definition for circle
double area(double radius)
{
    return Pi * radius * radius;
}

// Definition Overloaded for cylinder
double area(double radius, double height)
{
    // reuse the area of circle
    return 2 * area (radius) + 2 * Pi * radius * height;
}
```

- 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

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

int square(int x);

int main()
{
    int x = 2;

    cout << "The square of " << x << " is "
         << square(x) << endl;

    return 0;
}

int square(int x)
{
    return x * x;
}
```

- 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

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

int square(int& x);

int main()
{
    int x = 2;

    cout << x << "^2 is " << square(x) << endl;

    cout << x << "^2 is " << square(x) << endl;

    return 0;
}

int square(int& x)
{
   return  x *= x;
}
```

- 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

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

int square(const int& x);

int main()
{
    int x = 2;

    cout << "The square of " << x << " is "
         << square(x) << endl;

    return 0;
}

int square(const int& x)
{
    //x *= x; // compilation error! x-cant be
        changed
    return x * x;
}
```

- 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

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

void square(const int& x, int& result);

int main()
{
    int x = 2;
    int result = 0;

    square(x, result);
    cout << "The square of " << x << " is "
         << result << endl;

    return 0;
}

void square(const int& x, int& result)
{
    result =  x * x;
}
```

Find the Bug

```cpp
#include <iostream>

using namespace std;
const double Pi = 3.1416;

void Area(double radius, double result)
{
    result = Pi * radius * radius;
}

int main()
{
    cout << "Enter radius: ";
    double radius = 0;
    cin >> radius;

    double areaFetched = 0;
    Area(radius, areaFetched);

    cout << "The area is: " << areaFetched << endl;
    return 0;
}
```

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

Function Parameters with Default Values

```
#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

```cpp
#include <iostream>
#include <algorithm>
#include <vector>

void DisplayNums(vector<int>& dynArray)
{
    for_each (dynArray.begin(), dynArray.end(), \
            [](int Element) {cout << Element << " ";} );

    std::cout << endl;
}

int main()
{
    std::vector<int> myNums;
    myNums.push_back(501);
    myNums.push_back(-1);
    myNums.push_back(25);
    myNums.push_back(-35);

    DisplayNums(myNums);

    std::cout << "Sorting them in descending order" <<
        std::endl;

    sort (myNums.begin(), myNums.end(), \
          [](int Num1, int Num2) {return (Num2 < Num1); }
              );

    DisplayNums(myNums);
    return 0;
}
```

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

???