Advanced Programming

Zeinab Zali

References: (1) "C++ How to program" Deitel&Deitel, (2) "A Tour of C++" Bjarne Stroustrup, (3) Other useful learning pages such as geeksforgeeks and tutorialpoints

ECE Department, Isfahan University of Technology

The inventor

C++, an extension of C, was developed by Bjarne Stroustrup in 1979 at Bell Laboratories. Originally called “C with Classes,” it was renamed to C++ in the early 1980s.

The inventor

Stroustrup, when he was young and created C++

C++ versions

- **C++03 or C++98:** The C++ programming language was initially standardized in 1998 as ISO/IEC 14882:1998
- **C++11:** It makes C++ a better language for systems programming and library building and makes C++ easier to teach and learn.
- **c++14:** It was deliberately a minor release aiming at "completing C++11"
- **C++17**: It added several language features and C++ Standard Library enhancements, and fixed bugs from C++11
- **C++20**: It adds more new major features than C++14 or C++17

Configuring C++ version

- g++ source_file.cpp -o executable_name -std=c++11 for C++11
- g++ source_file.cpp -o executable_name -std=c++14 for C++14
- g++ source_file.cpp -o executable_name -std=c++17 for C++17
- g++ source_file.cpp -o executable_name -std=c++2a for C++20

The Features

- C++ provides a number of features that "spruce up" the C language, but more importantly, it provides capabilities for object-oriented programming.
- Two parts to learn the C++ "world":
  - The C++ language itself (often referred to as the "core language")
  - Classes and functions in the C++ Standard Library

Performance Tip 1.2

Using C++ Standard Library functions and classes instead of writing your own versions can improve program performance, because they're written carefully to perform efficiently. This technique also shortens program development time.

Portability Tip 1.1

Using C++ Standard Library functions and classes instead of writing your own improves program portability, because they're included in every C++ implementation.

C++ Standard Library

The C++ Standard Library can be categorized into two parts:

- The Standard Function Library: This library consists of general-purpose,stand-alone functions that are not part of any class. The function library is inherited from C.
- The Object Oriented Class Library: This is a collection of classes and associated functions.

first program

- **iostream** header in C++ is a replacement of **stdio** header in C
- header file inclusion directive does not need .h for C++ libraries
- The **std::** before **cout** is required when we use names that we've brought into the program from **iostream**

```
1  // Text-printing program.
2  #include <iostream>
3  // enables program to output data to the screen
4
5  // function main begins program execution
6  int main() {
7     std::cout << "Welcome to C++!\n"; // display message
8     return 0; // indicate that program ended successfully
9  }
10 // end function main
11
```

Standard Input/Output streams

- **cin:** The standard input stream which is normally the keyboard, but cin can be redirected to another device
- **cout:** The standard output stream which is normally the computer screen, but cout can be redirected to another device
- **cerr:** The standard error stream which is normally connected to the screen for playing error messages

```
1  // Text-printing program.
2  #include <iostream>
3  // enables program to output data to the screen
4
5  // function main begins program execution
6  int main() {
7     std::cout << "Welcome to C++!\n"; // display message
8     return 0; // indicate that program ended successfully
9  }
10 // end function main
11
```

I/O Operators

- In the context of an output statement, the « operator is referred to as the stream insertion operator.
- In the context of an Input statement, the » operator is referred to as the stream reader operator.

```
1// Printing a line of text with multiple statements.
2    #include <iostream>
3
4    int main() {
5        int num;
6        std::cout << "Welcome ";
7        std::cout << "to C++!\n";
8        std::cout << "Good luck\nEnter a random number:";
9        std::cin >> num;
10       std::cout <<"You number is "<<num<<std::endl;
11   } // end function main
```

Variable Declaration

```
1#include <iostream>
2int main() {
3    // declaring and initializing variables (new in C++11)
4    int number1{0}; // first integer to add
5    int number2{0}; // second integer to add
6    int sum{1000}; // sum with a non-zero initial value
7
8    std::cout << "Enter two numbers: "; // prompt user for
    data
9    std::cin >> number1; // read first integer into number1
10   std::cin >> number2; // read second integer into
    number2
11   sum += number1 + number2;
12   std::cout << "New sum is " << sum << std::endl;
13   // display sum; end line
14}
```

Using Declaration

- We can use "using std::cout"/"using std::cin" declaration to eliminate the need to repeat the std:: prefix before cout/cin

```
1  #include <iostream>
2  using std::cin;
3  int main() {
4      int number{0};
5      cin >> number;
6      std::cout << number<<std::endl;
7  }
```

- Also We can use "using namespace std" which enables a program to use all the names in any standard C++ header (such as <iostream>) that a program might include

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int number{0};
5      cin >> number;
6      cout << number<<endl;
7  }
```

Object Oriented

- With **C**, we can write structured programs using functions and multiple source files.
- With **C++**, we can write Object-Oriented programs using classes including functions.

Object Oriented Programming

- **Objects**, or more precisely the **classes** objects come from, are essentially **reusable software components**.
- There are date objects, time objects, audio objects, video objects, automobile objects, people objects, etc.
- Almost any noun can be reasonably represented as a software object in terms of **attributes** (e.g., name, color and size) and **behaviors** (e.g., calculating, moving and communicating).
- Software developers have discovered that using a modular, object-oriented design-and-implementation approach can make software development groups much more productive than was possible with earlier techniques
- object-oriented programs are often easier to understand, correct and modify

Object Oriented Terms

- **Classes:** user-defined data types that act as the blueprint for individual objects, attributes and methods
- **Objects:** instances of a class created with specifically defined data
- **Methods:** functions that are defined inside a class that describe the behaviors of an object.
- **Attributes:** defined in the class template and represent the state of an object.
  - Objects will have data stored in the attributes field.
  - Class attributes belong to the class itself.

Namespaces

Namespace is a mechanism for expressing that some declarations belong together and their names shouldn't clash with other names

```
1   #include <iostream>
2   #include <cmath>
3   using namespace std;
4   namespace my_namespace {
5       const int Mega = 1000000;
6       const int Kilo = 1000;
7       float sqrt(float);
8       void f();
9   }
10  float my_namespace::sqrt(float a){
11      return std::sqrt((int)a);
12  }
13  using namespace my_namespace;
14  int main(){
15      cout << "sqrt(16.8) in cmath:" << sqrt(16.8) <<
    endl;
16      cout << "sqrt(16.8) in my_namespace:" ;
17      cout << my_namespace::sqrt(16.8)<<endl;
18      cout << Kilo << endl;
19  }
```

Function Overloading

Function Overloading is having more than one function of the same name, but with arguments of different types or different numbers of arguments.

- When an overloaded function is called, the C++ compiler selects the proper function by examining the number, types and order of the arguments in the call.

Common Programming Error 6.9

Creating overloaded functions with identical parameter lists and different return types is a compilation error.

Function Overloading

```
1   #include <iostream>
2   using namespace std;
3
4   void display(int x, int y){
5       cout<< "x = " << x << ", y = " << y << endl;
6   }
7   void display(char *s1, char *s2){
8       cout<< "s1 = " << s1 << ", s2 = " << s2 << endl;
9   }
10  void display(float x, float y){
11      cout<< "x = " << x << ", y = " << y << endl;
12  }
13  void display(float x){
14      cout<< "x = " << x << endl;
15  }
```

Function Overloading

```
16      int display(char *s){
17          int len = 0;
18          while(*s!=0){
19              cout<< *s++;
20              len ++;
21          }
22          cout << endl;
23          return len;
24      }
25      int main(){
26          display(10,20);
27          display(10.7);
28          display("Hello", " C++");
29          display("Hello C++");
30      }
```

Default function parameters

In C++, it is possible to assigne default value for some parameters of a function.

- When calling function, if we don't pass those arguments, their default values are considered
- When a parameter has a default value, all the parameters after that must have a default value too.

```
1   #include<iostream>
2   using namespace std;
3
4   //compile error, c must have a default value too
5   int f1(int a, int b=10,int c){
6       return a+b+c;
7   }
8   int f2(int a, int b=20){
9       return a+b;
10  }
11  int main(){
12      cout << f2(1)<<endl;
13  }
```

Default function parameters

Common Programming Error 6.10

A function with default arguments omitted might be called identically to another overloaded function; this is a compilation error. For example, having a program that contains both *a function that explicitly takes no arguments and a function of the same name that contains all default arguments results in a compilation error when an attempt is made to use that function name in a call passing no arguments. The compiler cannot determine which version of the function to choose.*

Reference (&) vs Pointer ()

- If we want different objects to refer to the same (shared) value, we could use pointers.
- A reference is similar to a pointer, except that:
  - you don't need to use a prefix to access the value referred to by the reference.
  - a reference cannot be made to refer to a different object after its initialization.

Pointer

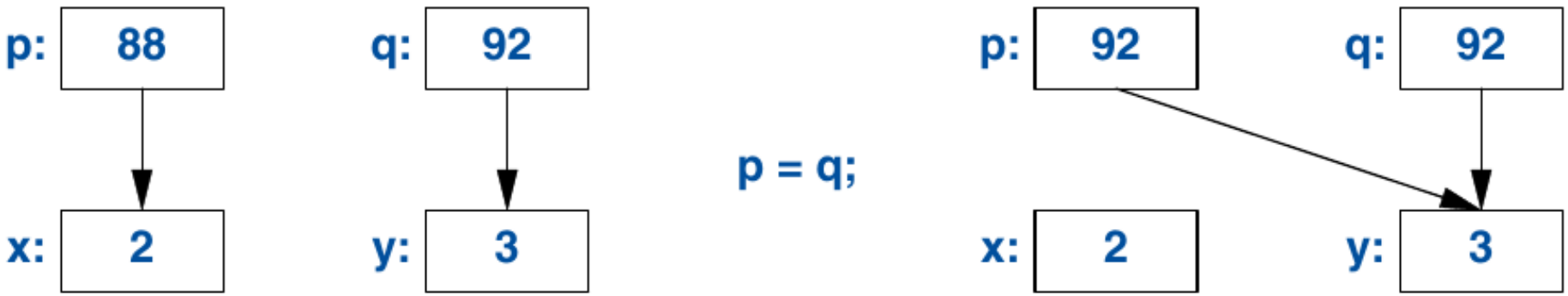
```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      int a = 2, b = 3;
5      int *p, *q;      // p and q are pointer to int
6      p = &a;           //p
7      cout << "p=" << p << ",*p=" << *p << ", a=" << a <<
       endl;
8      q = &b;
9      cout << "q=" << q << ",*q=" << *q << ", b=" << b <<
       endl;
10     p = q;
11     cout << "p=" << p << ",*p=" << *p << ", a=" << a <<
       endl;
12     cout << "q=" << q << ",*q=" << *q << ", b=" << b <<
       endl;
13 }
```

Reference

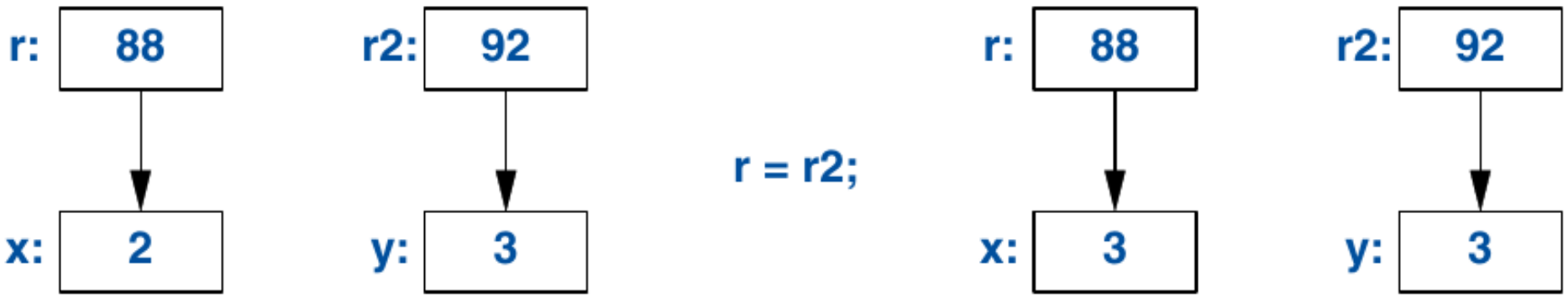
```
1   #include <iostream>
2   using namespace std;
3   int main(){
4       int x = 0;
5       int y = 3;
6       int& r1 {x}; // bind r1 to x (r1 refers to x)
7       cout << "r1=" << r1 << ", x=" << x << endl;
8       r1 = 2;     // assign to whatever r1 refers to
9       cout << "r1=" << r1 << ", x=" << x << endl;
10      //int &r2;    // error: uninitialized reference
11      int &r2 = y;    //bind r2 to y (r2 refers to y)
12      cout << "r1=" << r1 << ", x=" << x << endl;
13      cout << "r2=" << r2 << ", y=" << y << endl;
14      r1 = r2;
15      cout << "r1=" << r1 << ", x=" << x << endl;
16      cout << "r2=" << r2 << ", y=" << y << endl;
17  }
```

Reference (&) vs Pointer (*)

The assigned-to pointer gets the value from the assigned pointer, yielding two independent pointers with a same address.



Assignment to a reference does not change what the reference refers to but assigns to the referenced object



Reference in function parameter

We can use references for call by reference

```
1   #include <iostream>
2   using namespace std;
3   void swap(float &x, float &y){
4       float tmp = x;
5       x = y;
6       y = tmp;
7   }
8   int main(){
9       float x{19.9};
10      float y{20};
11      swap(x,y);
12      cout << "x=" << x << ",y=" << y << endl;
13  }
```

We can use const reference if wa use reference only for decreasing the cost of copy

- print(const struct &std)

Reference in function return

```
1   #include<iostream>
2   using namespace std;
3   int vals[]={10,20,30,40};
4
5   int & value(int i){
6       if (i>=0 && i<=4)
7           return vals[i];
8   }
9   int main(){
10      cout<<value(1)<<endl;
11      value(1) = 7;
12      cout << value(1)<<endl;
13      cin >> value(0);
14      cout << value(0)<<endl;
15  }
```

new/delete instead of malloc/free

- **new** operator is used to allocate memory dynamically
- **delete** operator is used to free allocated memory

```
1   #include <iostream>
2   int main(){
3       int *a = new int[10];
4       int *b = new int;
5       *b = 100;
6       for(int i=0;i<10;i++)
7           a[i] = i;
8       delete[] a;
9       delete b;
10  }
```

auto

C++11 **auto** keyword tells the compiler to infer (determine) a variable's data type based on the variable's initializer value.

```
1     #include <iostream>
2     #include <cmath>
3     int main(){
4         auto b = true;
5         auto ch = 'x';
6         auto i = 123;
7         auto d = 1.2;
8         auto z = sqrt(10);
9         auto bb {true};
10    }
```

auto in for syntax

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      float v[5]={2,3.4,6.4,3,99};
5      for (auto &x:v)    // add 1 to each x in v
6          x++;
7      for (auto x:v)    // add 1 to each x in v
8          cout<< x <<endl;
9  }
```

new nullptr and bool

- Pointers should be initialized to **nullptr** (added in C++11) or to a memory address either when they are declared or in an assignment.
  - a pointer with the value nullptr "points to nothing" and is known as a null pointer
- C++ has **bool** type with two values true and false
  - zero is evaluated as **false** and other numbers are **true**

```
1   #include <iostream>
2   using namespace std;
3   int main(){
4       bool pass = true;
5       int *p1 = nullptr;   //ok, it assignes 0 address
6       int *p2 = NULL;      //ok, it assignes 0 address
7       int x = NULL;        //ok, it assignes 0 value
8       // int y = nullptr;     //error
9       cout << pass << endl;
10      cout << p1 <<endl << p2 << endl << x << endl;
11  }
```

useful string type

- We can use **string** in C++ instead of char arrays in C
- A string is actually an object of the C++ Standard Library class string, which is defined in the header <string>
- There are many useful operators and function on strings in <string>

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4   int main(){
5       string s1 = "Welcom to";
6       string s2 = " C++";
7       string s = s1 + s2;
8       cout << s << endl;
9       cout <<s.substr(1,3) <<endl;//substring from index
    1 with the size of 3
10  }
```