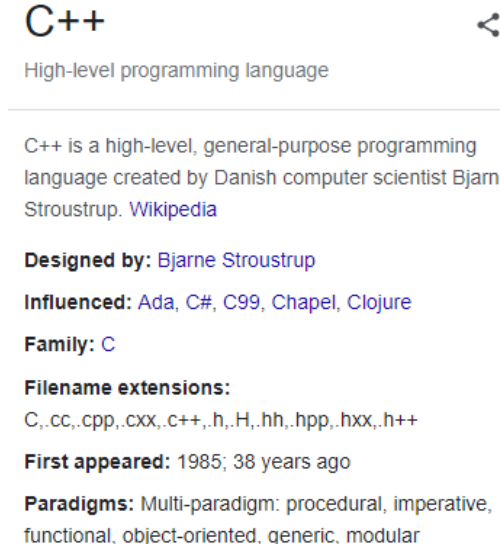
Αντικειμενοστρεφής Προγραμματισμός (Object-Oriented Programming)

(CEID_NNY106)

C++ for Java Developers



Kleanthis Thramboulidis
Prof. of Software and System Engineering
University of Patras
https://sites.google.com/site/thramboulidiskleanthis/

Moving from Java to C++

- "This appendix explains how to transfer your Java programming skills to a substantial subset of C++.
- learning to **move from one language to another** is a fact of life for today's software professionals.
- C++ has many features in common with Java, and **it is easy for a Java programmer to gain a working knowledge of C++**.
- Nevertheless, **C++ is a much more complex language than Java**.
- This appendix does not attempt to cover all features of C++. But if you master all of the constructs described in this appendix, **you will be able to use C++ effectively**.
- We only cover the differences between Java and C++.
- control flow statements are essentially identical in C++ and Java.

https://horstmann.com/ccj2/ccjapp3.html

Data Types and Variables

- The data types in C++ are similar to those in Java.
- the range of the numeric types such as int is machine-dependent.
  - On 16-bit systems such as PCs running DOS or Windows 3.x, int are 2-byte quantities with a much more limited range than the 4-byte Java int type. On those machines, you need to switch to long whenever the int range is not sufficient.
- C++ has **short** and **unsigned** types that can store numbers more efficiently. It is best to avoid these types unless the added efficiency is crucial.
- The Boolean type is called **bool** in C++.
- The C++ string type is called **string** (std::string). It is quite similar to the Java String type. However, pay attention to these differences:
  1. C++ strings store ASCII characters, **not Unicode characters**
  2. C++ **strings can be modified**, whereas Java strings are immutable.
  4. You can only **concatenate** strings with other strings, not with arbitrary objects.
  5. To **compare strings**, use the relational operators == != < <= > >=. The last four operators perform lexicographic comparison (more convenient than the use of equals and compareTo in Java).

Variables and Constants

- In C++, local variables are defined just as in Java.

```
int n = 5;
```

- There is, however, a major difference between C++ and Java.
  - The C++ compiler **does not check whether all local variables are initalized** before they are read. It is quite easy to forget initializing a variable in C++. The value of the variable is then the random bit pattern that happened to be in the memory location that the local variable occupies. This is clearly a fertile source of programming errors.
- As in Java, classes can have data fields and static variables.
- Variables can be declared outside functions and classes (global variables).
- In C++, constants can be declared anywhere. (Recall that in Java, they had to be static data of a class.)
- C++ uses the **const** keyword instead of final.

```
const int DAYS_PER_YEAR = 365;
```

Classes

```
class Point {     /* C++ */
public:

  Point();
  Point(double xval, double yval);
  ~Point(); // destructor
  void move(double dx, double dy);
  double getX() const;
  double getY() const;

private:
  double x;
  double y;
};
```

- there are public and private sections.
- Accessor methods are tagged with the keyword const
- There is a semicolon at the end of the class
- The class definition only contains the declarations of the methods. The actual **implementations <u>can be listed</u> separately.**

```
Point::Point() { x = 0; y = 0; }
void Point::move(double dx, double dy){
          x = x + dx;
          y = y + dy;
          }
double Point::getX() const {
     return x; }
```


Instances 1/2

- The major difference between Java and C++ is the behavior of object variables.
- In C++, object variables hold values, not object references.
- Instances You simply supply the construction parameters after the variable name.

```
Point* p = new Point(1,2); /* new operator returns a pointer  */
Point p(1, 2);    /* construct p */
```

- If you do not supply construction parameters, then the object is constructed with the default constructor.

```
Time now;      /* construct now with Time::Time() */
```

- When one object is assigned to another, a copy of the actual values is made. Copying a C++ object is just like calling clone in Java. Modifying the copy does not change the original.

```
Point q = p; /* copies p into q */
q.move(1, 1); /* moves q but not p */
```

Instances 2/2

In most cases, the fact that objects behave like values is very convenient. There are, however, a number of situations where this behavior is undesirable.

- 1. When modifying an object in a function, you must remember to use call by reference
- 2. Two object variables cannot jointly access one object. If you need this effect in C++, then **you need to use pointers**
- 3. An object variable can only hold values of a particular type. If you want a variable to hold objects from different subclasses, **you need to use pointers**
- 4. If you want a variable point to either null or to an actual object, then **you need to use pointers**

Functions

- Define behavior of instances or classes.
- Global functions: Functions defined outside of classes.
- Argument passing : call by value and call by reference.

```
1   #include <iostream>
2
3   void swap1(int a, int b);
4   void swap2(int& a, int& b);
5   /* run this program using the console p
6   int a=10;
7   int b = 20;
8
9   int main(int argc, char** argv) {
10      printf("swap\n");
11      printf("a=%d \t b=%d \n",a,b);
12      swap1(a,b);
13      printf("a=%d \t b=%d \n",a , b);
14      swap2(a,b);
15      printf("a=%d \t b=%d \n",a , b);
16      return 0;
17  }
```

```
18
19  void swap1(int a, int b){
20      int temp = a;
21      a = b;
22      b = temp;
23  }
24
25  void swap2(int& a, int& b){
26      int temp = a;
27      a = b;
28       b = temp;
29  }
```

```
swap
a=10    b=20
a=10    b=20
a=20    b=10
```

Input and Output 1/2

- In C++, the **standard input and output stream** are represented by the **cin** and **cout** objects. use the << operator to write output.

```
cout << "Hello, World!";
```

- To print multiple items.

```
cout << "The answer is " << x << "\n";
```

- To read a number or a word from input, use the >> operator.

```
double x;
cout << "Please enter x: ";
cin >> x;
string fname;
cout << "Please enter your first name: ";
cin >> fname;
```


Input and Output 2/2

- If the end of input has been reached, or if a number could not be read correctly, the **stream is set to a failed state**. You can test for that with the **fail method**.

```
int n;
cin >> n;
if (cin.fail()) cout << "Bad input";
```

- Once the stream state has failed, you cannot easily reset it. If your program needs to handle bad input, you should use the getline method and then manually process the input.
- The **getline** method reads an entire line of input.

```
string inputLine;
getline(cin, inputLine);
```

Pointers 1/2

- In C++, a variable that can refer to an object is called a **pointer**. If T is any type, then T* is a pointer to an object of type T.
- a pointer variable can be initialized with a) NULL, b) a call to new, c) another pointer variable.

```
Producer* p1 = NULL;
Producer * p2 = new Producer(buf);
Producer * p3 = p2;
```

- a pointer variable can also be initialized with the address of another object, by using the & operator.

```
Producer producer1(buf);
Producer * producer2 = & producer1;
```

- This is usually not a good idea. As a **rule of thumb,** C++ pointers should only refer to objects allocated with new.

Pointers 2/2

- You must **apply the * operator** to access the object to which a pointer points. If p is a pointer to an Employee object, then *p refers to that object.

```
Employee* p = . . .;
Employee boss = *p;
(*p).setSalary(91000); // invokes the setSalary method on the object *p
p->setSalary(91000);   // invokes the setSalary method on the object *p
```

- In C++, it is the responsibility of the programmer to manage memory.
- Object variables are automatically reclaimed when they go out of scope. However, objects created with new must be reclaimed manually with the delete operator.

```
Employee* p = new Employee("Hacker, Harry", 38000);
. . .
delete p; /* no longer need this object */
```

Inheritance 1/2

- In C++, you use : public instead of extends to denote inheritance.
- By default, functions are not dynamically bound in C++. If you want which dynamic binding for a particular function, you must declare it as virtual.

```
class Manager : public Employee {
public:
    Manager(string name, double salary, string dept);
    virtualvoid print() const;
private:
    string department;
};
```


Inheritance 2/2

- call the **superclass constructor** outside the body of the subclass constructor.

```
Manager::Manager(string name, double salary, string
dept)
: Employee(name, salary) /* call superclass constructor */
{  department = dept; }
```

- To call the superclass method use the name of the superclass and the :: operator.

```
Employee::print(); /* call superclass method */
```

- A C++ object variable holds objects of a specific type. **To exploit polymorphism in C++, you need pointers.** A T* pointer can point to objects of type T or any subclass of T.

java to c++ transition tutorial

cs123: Java to C++ Transition Tutorial

https://cs.brown.edu/courses/cs149/handouts/javatoc.shtml

Table of Contents