ECE 462
Object-Oriented Programming using C++ and Java

Lecture 23

Yung-Hsiang Lu

yunglu@purdue.edu

C++ History

- why to study history?
  - Knowing the past often helps us plan for the future.
  - The design decisions of one programming language help us design better languages.
- Since C++ (1982), many new programming languages have been developed:
  - 1991 Python
  - 1995 Java 1
  - 1995 PHP
  - 1997 OO COBOL
  - ...

History of C++: 1979-1991
by Bjarne Stroustrup

- background
  - 1977 Apple 1 & 2 (1MHz processor, 4-48KB memory, $1300-$2600)
  - 1979 Intel 8088
  - 1980 Seagate (then called Shugart) 5.25-in 5MB disk
  - 1981 IBM PC (4.77MHz, 16-640KB memory)
  - 1983 TCP/IP

- Computers were slow and expensive.

C++

- design goals:
  - Simula's facilities for program organization
  - C's efficiency and flexibility
  - for system programming
- 1979-1983 C with Classes
- 1982-1985 C++
- 1985-1988 C++ 2.0
- 1988- standardization (ISO / ANSI)
- ISO = International Organization for Standardization
- ANSI = American National Standards Institute

Simula

- simulator for a distributed system
- class hierarchy
- capturing type errors by compiler
  - type: int, string, Student, Computer ...
  - type error, for example, a Student object + 3, a Computer object + "hello" ...
- problem of Simula: link time too long
  - run-time type checking
  - variable initialization
  - garbage collection, even for a program without garbage

  $\Rightarrow$ performance too low

Programming Language Design

- Never attack a problem with wrong tools.

1. support for program organization: class, hierarchy, concurrency, static type checking
2. good tools to compile files separately, to link files written in different languages, and to produce fast programs
3. portable across different machines

- His background in OS and communication affects many design decisions, such as model of protection and exception handling
- **A good language requires a good implementation. Performance matters.**

C with Classes

- new language developed to analyze UNIX kernel: analyze network traffic and modularize kernel

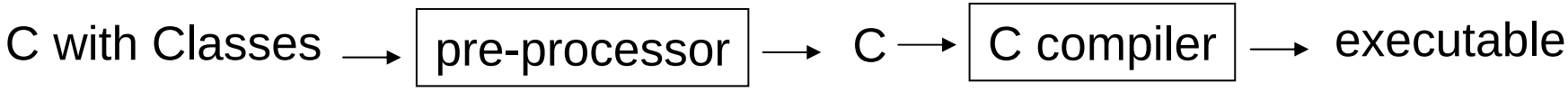
$\Rightarrow$ develop an extension of C by adding tools

$\Rightarrow$ Some programming languages are developed for specific purposes and then are generalized.

- no primitives to express concurrency, use libraries instead (different from Java with built-in thread supports)
  - built-in support: consistent with language, but may cause unnecessary overhead to the users that do not need this feature
  - libraries: more flexibility but increase the overhead in system administration to ensure version compatibility
- C with Classes to be used anywhere C is used $\Rightarrow$ **efficiency** requirements eliminate built-in runtime safety checking

Features in C with Classes

- "philosophy": language designers should not force programmers to use a particular style; instead, designers should provide styles, practices, and tools to help programmers avoid well known traps ⇒ C allows low-level operations and type conversions, so does C with Classes
- features (1980): class, derived class, public / private access, constructor / destructor, call and return, friend class, type checking and conversion of function arguments
- features (1981): inline, default arguments, overloading of assignment operator
- C with Classes was implemented as pre-processor of C ⇒ portable across machines ⇒ **common approach** for language design today



Design Decisions in C with Classes

- A class is a type.
- Local variables are allocated at stack, not heap $\Rightarrow$ no need to call destructor or garbage collection
- Default access control is private.
- Static type checking for function arguments and return values.
- Class declarations and function definitions can be in different files (different from Java). Hence, class declaration can be the "interface" (Java distinguishes interface from class)
- "new" calls constructor (not all valid C programs are valid C++ programs)
- Use-defined types (classes) are treated in the same way as the built-in types.
- Function inlining is used to reduce the overhead of calls $\Rightarrow$ discourage programmers from declaring data members as public.

```
// classX.h
class X {
public:
    void foo(int, float);
};
// both a class declaration and
// interface
```

```
// classX.cpp
#include "classX.h”
void X::foo(int a, float b) {
    ...
}
// define the implementation of a
// member function
```

Garbage Collection in C++

- considered until 1985
- inappropriate for a language (C) already had run-time memory management
- GC would degrade performance unacceptably

- Stroustrup stressed that there was no "grand plan" to develop C++. Hence, the usefulness of the language resided on the ability to attract users in Bell Lab by solving their problems, efficiently.

1982 C++

- C with Classes was a "medium success"
- major features:
  - virtual function
  - function and operator overloading
  - reference
  - constant
- virtual function
  - to adapt to similar but different (common base class) types
  - a large if-then-else or switch-case block is undesirable
  - dilemma: allow adaptability by users without allowing the change of base classes (possibly from the library)

```
void shape::draw()
{
    switch (type) {
    case circle:
        // draw a circle
        break;
    case square:
        // draw a square
        break;
    case triangle:
        // draw a triangle
        break;
    }
}
```

1986 C++ 2.0

- multiple inheritance, "the fundamental flaw in these arguments is that they take multiple inheritance far too seriously... it is quite cheap... you don't need it very often but when you do it is essential."
- type-safe linkage
- abstract class
- static member functions
- protected members
- overloading ->

- Exception handling was added later.

Summary

- C++ was developed to solve a specific problem: simulating distributed systems
- It is important to choose a good language as the base and build on top of the base; this can obtain immediate tool support.
- Features do not have to be added at once. Most features are added out of necessity, as the basic functionalities are available.
- Separate compilation and linking is critical for developing large-scale programs.
- Performance is essential. Many design decisions are based on the impact of performance.

Brief History of Java 1995-

- started in 1991 and announced in 1995
- Java started as a technology for entertainment "set-top box" to create a language that can run on small portable systems, not intended for system programming (as C++) ... but cable companies were unwilling to support
- The focus then switched to support Internet for processor (hardware) independent and operating-system independent (to be further discussed later)
- need: execute programs from remote machines through the Internet

$\Rightarrow$ A new language is more likely to succeed to solve a new problem. Solving an old problem is harder because of the existing programs and the infrastructures.

- 1994, a "better browser"

Then ... and ... Now

- interactive browser:
  - With Java, users can interact with the browser, beyond browse, scroll, and click.
  - Sun Microsystem, as a primarily hardware company, developed Java to create the demand for high-performance networking equipment and computers.
  - 1995/03/23 San Jose Mercury News headline
  - Security is crucial since malicious code can easily propagate through the Internet (different goals from C++)
- widely used on
  - 4.5B devices
  - 1.5B phones
  - printer, webcam, game, car ...

Java: Sun vs. Microsoft

- 2002, Sun filed a lawsuit against Microsoft for violating the license agreement about Java
- Java was considered a threat to Microsoft's control of the operating system market.
- Sun accused that Microsoft modified Java in Windows and thus made it incompatible with other platform running Java.
- 2004, the two companies settled

- (background)
  - 1998 US antitrust against Microsoft, settled on 2001/11/02
  - 2003 European Union issued penalty to Microsoft

	C++	Java
organization	AT&T Bell Lab	Sun Microsystem
target environment	system programming	embedded system Internet
base language	C	N/A
priority	efficiency	security
growth force	personal computer (to a lesser extent)	Internet
object-oriented	optional	mandatory
run-time array index checking	N/A	exception
memory management	destructor	garbage collection
global base class	N/A	Object
multiple inheritance	yes	interface

	C++	Java
concurrency	external library	built-in, thread
friend function / class	yes	N/A
parameter passing	value (primitive types), pointer, reference	value (primitive types), reference
virtual function	explicit	implicit
separate interface and implementation	yes (.h and .cpp)	N/A
exception handling	yes	yes
function overloading	yes	yes
default value of function parameters	yes	N/A
operator overloading	yes	N/A
graphics library	external	built-in AWT and SWING

Lessons Learned

- A successful language needs a clearly defined target. Creating a new language to replace an existing one is unlikely to succeed.
- Prioritize the requirements: efficiency for C++ and platform neutral for Java
- Tools (compiler, linker, debugger, runtime environment ...) and libraries (graphics, thread ...) are crucial, probably more important than the "elegance" of a language.
- Performance cannot be ignored. Any new language will be compared with C in terms of performance.
- Keep the non-essential portions of the new language the same as a popular existing language. Do not confuse users.
- Be aware of non-technical forces (such as legal issues)