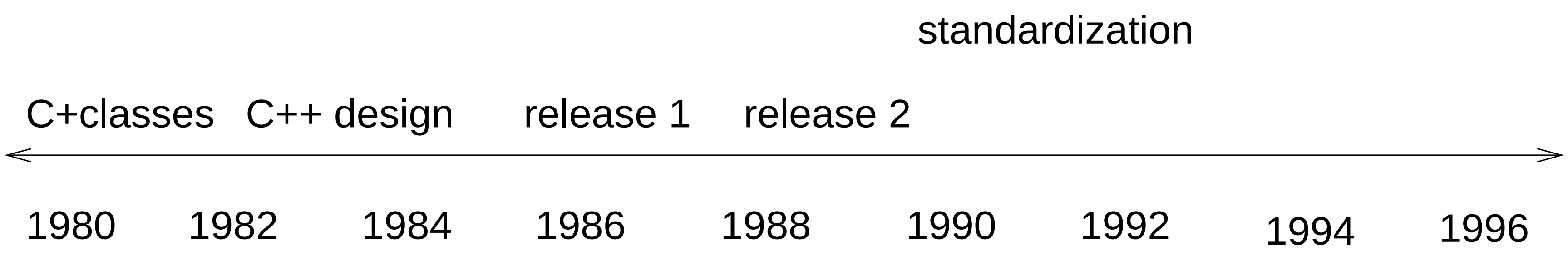
C++

C++ is a class-based, object-oriented language.

Design goals:

- provide Simula's facility for program organization
- with C's efficiency for systems programming
- retain flexibility, portablity and availability of C



C++ Classes

```
class employee {
  public:
    void print ();
  private:
    int salary; ...};
```

Class contain *public*, *protected* and *private* data and functions.

- Private things are only visible by member functions in the class.
- Public things are visible anywhere.
- Protected things are visible only by member function in the class and by the member functions of classes that inherit from this class.

Exception to the rules: a class may declare other classes or functions as *friends*. Friend may see anything that the class can see.

C++ classes are idential to C++ structs, which are like C structs. The only difference is default protection: public for structs; private for classes.

C++ Classes (continued)

The rules on the previous slide are about naming/scoping. An orthogonal question is how many copies of the variable there are (one per object or one per class) and how long the variable retains its value.

Classes (like C functions) may contain *static* data and functions, which are shared by all instances of the class.

```
static employee* all_emp;
static void print_all();
```

The lifetime of a static variable is the lifetime of the program; all other variables in the class have the lifetime of the containing object.

A static function may only access static data, not object-specific data.

Constructors and Destructors

With each class, we may associate a set of constructors and one destructor (defaults provided).

- Constructors are called automatically whenever a variable of that type is declared or a new object created.
- Copy constructor is called when passed by value.
- Destructor called when the variable disappears from scope or object deleted.

```
class Account {
   public:
     Account();                  // a constructor
     Account(int);               // another constructor
     Account(& Account);         // copy constructor
     ~Account();                 // the destructor
     ...
};

void foo () { Account a; ... }
```

Inheritance in C++

A *derived* class inherits from a *base* class in C++.

```
class manager : public employee {
  public:
    employee* group;
    int level; ...
  };
```

Within the derived class, the base class names can be explicitly accessed by appending the base class name:

```
employee::print();
```

A class may be inherited as either *public* or *private*, determining external visibility of the names.

```
class secretary : public employee;
class immutable_set : private set;
```

Different Styles of Inheritance

```
      Is-a                     As-a            Has-a
      -----                    ----            -----
     Geometric                 List            Board
      /     \                 /     \          /   \
   OneD      TwoD        Stack      Queue  Player  Referee
   /         /   \
Line   Square   Circle

A Square is a TwoD          A Stack can be     A Player has a
object, etc.                implemented as     Board inside.
                            a list.
```

The `Is-a` style inheritance is inheriting the interfaces, whereas *As-a* and *Has-a* are inheriting code (implementation inheritance, probably private).

Virtual Functions

It is often the case that you want function definitions in the derived class to override those of the base class. In C++, this is not the default behavior. Instead, you need to use *virtual functions*.

```
class employee { ...
    virtual void print();
};

employee::print() { ... }
```

The base class function give the "default" behavior for all derived objects, but may be redefined.

Pure Virtual Functions

Sometimes, there may be no reasonable default behavior, but we want a standard interface to be used by all derived classes. *Pure virtual functions* allow this.

```
class shape {
   ...
   void move (int, int);
   virtual void erase (int, int);
   virtual void rotate (int) = 0;
};
```

No definition of `rotate` is needed. Therefore, one may not define an object of type `shape`, but only pointers to them, which will always point to some objects of a derived class that defines `rotate`.

A base class with pure virtual functions is also called an *abstract base class*.

Multiple Inheritance

```
class teacher {                       class researcher {
   ...                                    ...
   void print(); };                       void print(); };
```

```
class professor public teacher, public research { ... };
```

Must explicitly disambiguate and may redefine:

```
   void professor::print() { researcher :: print(); }
```

If teacher and researcher both inherit from employeed, we get:

```
employee        employee
    |               |
teacher         researcher
     \             /
       professor
```

Unless employee is an abstract base class of teacher and researcher.

Multiple inheritance is very controversial, and was not included in early C++ definitions.

Polymorphism in C++

Operator and function name overloading in C++ provide *ad hoc* polymorphism.

- separate the semantics of desugared names from disambugation problems
- helps solve the binary operator wierdness of OO languages.

  ```
  Mynum operator+ (Mynum second) { ... }
  ```

- Can be used to overload nearly every operator in the language. Exceptions?
- Note that +, +=, and ++ are all defined independently.

Templates provides something closer to *parametric polymorphism* (passing types as parameters), but is really more like a macro facility.

```
template <class ElemType> List { ... };

template <class ElemType>
  sort (List <ElemType> l1, List <ElemType> l2);
```

Safety and C++

What are some of the common bugs in C++ programs? These are programming errors that do not necessarily generate a compiler or runtime error.

- Accessing an array out-of-bounds
- Dereferencing a null pointer.
- Accessing a previous deallocated object.
- Using a pointer to access a stack variable no longer exists.
- Memory leaks.

Many of the well-known bugs in Unix security rely on these techniques. They are also a big sourch of bugs in released software.

Object Models

A major difference between C++, C, and ML, Scheme, and CLU is the memory model.
In Scheme, CLU, and ML, everything variable (semantically) is a pointer to an object. Operationally, everything on a stack is a pointer, although the compiler may optimize some of these away.
In C and C++, objects may themselves named, not just pointers to them. This causes several complications:

- meaning of assignment, parameter passing.
- implementing polymorphism.
- tied to lack of GC and need for destructors.
- inheritance and use of virtual functions.

Many of the complications in C++ come from this, which is part of the general goal of being a C superset.

Safe C++

Is is possible to define a *safe* subset of C++?

What types of things need to be eliminated from C++?

- Pointer casts, except widening casts or checked narrowing casts (inheritance).
- Casting an integer to a pointer.
- No access to unitialized varables, members, or array elements.
- No access to a union member of the wrong type.
- No access to dangling pointers or references.
- Illegal pointer arithmetic or subscript expression (outside of array).
- Applying delete to a pointer not obtaine from new.
- Exiting from a value-returning function without a explicit return or throw.
- Incorrect use of variadic functions.

Observation: safety requires garbage collection or other automatic memory management.

Garbage Collection and C++

Definition of GC-safety [Owicki 81]: If all the program's actions are "legitimate" then the collector will remain "invisible" to the program.

Who enforces the notion of "legitimate?"

- programmer: Fortran aliases
- compiler: Strong Typing
- runtime system: Bounds checking and dynamic typing

Note complication of GC in an environment like C++ with destructors – when does the cleanup code execute to close files, etc.?

Three properties that a "legitimate" program must maintain:

1. Separation of memory: The program cannot modify the gc's internal memory.
2. Visibility of collected pointers: All pointers should be visible at all times.
3. Visibility of assignment: All asignments to pointers must be visible.

Java

According to Per van der Linden (a member of the Java team) The Java design was motivated by:

- popularization of the Internet
- consumer electronics merging into PC technologies
- computer industry drive to interactive multimedia everything
- rejection of the complexity of C++
- desire for independence from Microsofter/Intel desktop monopolies

Although safety is a concern for any programming language, it is especially important if you're going to run programs that other people place on the net.

Java Types

Java treats variable in two different ways, depending on their type:

- variable of built-in types: boolean, char, int, etc.
- objects

```
int i;  // defines an integer
i = 0;
i++;    // all OK

MyClass m;  // defines a pointer to MyClass
m = bar;    // m now shares the bar object
```

Similar difference in parameter passing.

Most variables are created by a call to a constructor:

```
Bicycle schwinn = new Bicycle();
Mammal dalmation = new Dog ("spotty");
```

Note that althought there are constructors, there are no destructors.

Java Portability

Polymorphism in Java

Polymorphism is Java is supported through overloading and inheritance. There is no parametric polymophism.

```
class Fruit { ...
   void peel(); };

class Citrus extends Fruit {
     void peel();
     void sqeeze();
}
```

You may now make a container (e.g., list) of Fruit, which may contain Citrus or other types of Fruit.

There is an "object" class from which other objects inherit.

"List of that object" works for everything except the built-in types.

Miscellaneous aside: There are no standalone functions in Java, just member functions.

The Byte Code Model

New ideas in Java:

- All platforms that support Java suppor the same “object code” called the Java byte code.
- Therefore, cross compilation is trivial.
- You can embed Java code in web pages.
  (“Hello world” of Java is a moving banner.)

The Java “compilers” generate byte code, for which an interpeter (will eventually) exist on all/most platforms.

The byte-code itself is safe (although slow, 100x for Linpack).