REVE1

Programming in C++

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

int main(void)
{
  cout << "Hello, world!" << endl;
  return 0;
}
```

Module Outline

- Environment
- Program Structure
- Language Elements
- Classes and Inheritance
- The C++ Standard Template Library
- Modern C++
- Module Summary

Environment

Environment

- Default extension of C++ programs is .cpp

- Compile with the GNU C++ compiler

  $ g++ hello.cpp

- In a Makefile, the variable CXX refers to the C++ compiler
  - should default to g++
  - set CXXFLAGS for general flags

  ```
  CXX=g++
  CXXFLAGS=-Wall -O2 -std=c++0x
  ```

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

int main(void)
{
  cout << "Hello, world!" << endl;
  return 0;
}
```

Program Structure

Hello, world!

- **Our first C++ program**

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

int main(void)
{
  cout << "Hello, world!" << endl;
  return 0;
}
```

```
$ g++ -o hello hello.cpp
$ ./hello
Hello, world!
$
```

General Structure

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

#define LIMIT 50

int A = 0;

int fib(int n) {
  if (n > 1) return fib(n-1)+fib(n-2);
  else return 1;
}

int main(void) {
  int n;

  cout << "Enter n: ";
  cin >> n;

  cout << "Fib(" << n << ") = " << fib(n) << endl;

  return 0;
}
```

include files
namespace declaration
preprocessor defines
global variables
function definitions
local variables
main function definition

Excellent Compiler Errors

- **C++ compilers have excellent error reporting functionality**
  - precise, concise, easy to understand
  - try it out

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

int fib(int n) {
  if (n > 1) return fib(n-1)+fib(n-2);
  else return 1;
}

int main(void) {
  int n;

  cout << "Enter n: ";
  cin  << n;

  cout << "Fib(" << n << ") = " << fib(n) << endl;

  return 0;
}
```

Language Elements

Language Elements

- **The syntax of C++ is similar to that of C**
  - C constructs work as expected
  - with extended features
  - used to be called “C with Classes”

- **Main differences**
  - object orientation
  - exception handling
  - templates
  - name spaces
  - standard template library (STL)

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

int main(void)
{
  cout << "Hello, world!“
       << endl;
  return 0;
}
```

Language Elements

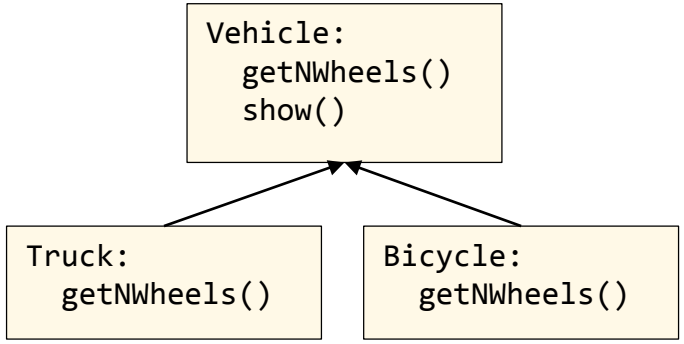
- **Comments, Identifiers, Whitespace**
  - identical to C
- **Additional keywords**
- **Control flow constructs identical**
- **Same data types plus**
  - bool
  - wchar_t (16bit wide char)
- **Same composite data structures but also supports STL arrays**

```
/*
 * language elements
 */
#include <stdio.h>

// this is the main function
int main    (     void
){
  // some variables
  int _i,j123__5;
  unsigned char x;
printf("Hello, world!\n");
  return 0;
}
```

Language Elements

- **Keywords**
  - those of C plus

asm	mutable	try
bool	namespace	typeid
catch	new	typename
class	operator	using
const_cast	private	virtual
default	protected	wchar_t
delete	public	
dynamic_cast	reinterpret_cast	
explicit	static_cast	
true	template	
false	this	
friend	throw	



Classes and Inheritance

Class Declaration and Definition

- **Class declaration**

  class <name> [: <relation> <superclass>]
  {
  [
  [<access attribute>:]
  [<method declaration> ; |
  <field declaration> ;]*
  ]*
  };

```
class Vehicle {
  public:
    Vehicle(int nwheels);

    void show(void) const;
    int getNWheels(void) const;

  private:
    virtual string getName(void);
    int _nwheels;
};
```

Class Declaration and Definition

- **Class Definition**

```
Vehicle::Vehicle(int nwheels)
  : _nwheels(nwheels)
{
}

void Vehicle::show(void) const
{
  cout << getName() << " has "
       << getNWheels()
       << " wheels." << endl;
}

string Vehicle::getName(void)
{
  return "Vehicle";
}
```

```
class Vehicle {
  public:
    Vehicle(int nwheels);

    void show(void) const;
    int getNWheels(void) const;

  private:
    virtual string getName(void);
    int _nwheels;
};
```

Object Declaration

- **Object declaration**
  <classname> <name>;

```
int main(void) {
  Vehicle v(4);
  v.show();
  return 0;
}
```

```
$ g++ -o v vehicle.cpp
$ ./v
Vehicle has 4 wheels.
$
```

Inheritance

- **Class inheritance**

```
class Vehicle {
  public:
    Vehicle(int nwheels);

    void show(void) const;
    int getNWheels(void) const;

  private:
    virtual string getName(void);
    int _nwheels;
};
```

```
class Truck : public Vehicle {
  public:
    Truck(void);

  private:
    virtual string getName(void);
};

Truck::Truck(void)
  : Vehicle(6)
{
}

string Truck::getName(void)
{
  return "Truck";
}
```

```
class Bicycle : public Vehicle {
  public:
    Bicycle(void);

  private:
    virtual string getName(void);
};

Bicycle::Bicycle(void)
  : Vehicle(2)
{
}

string Bicycle::getName(void)
{
  return “Bicycle";
}
```

Object Declaration

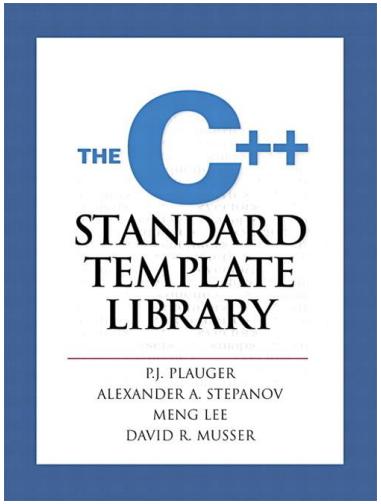
- **Dynamic object declaration**
  <classname> *<name>;

  <name> = new <classname>;

  delete <name>;

```
int main(void) {
  Vehicle *v;

  v = new Vehicle(4);
  v->show();
  delete v;

  v = new Truck();
  v->show();
  delete v;

  v = new Bicycle();
  v->show();
  delete v;

  return 0;
}
```

```
$ g++ -o v vehicle.cpp
$ ./v
Vehicle has 4 wheels.
Truck has 6 wheels.
Bicycle has 2 wheels.
$
```



The C++ Standard Template Library

The C++ Standard Template Library (STL)

- The C++ equivalent to the C standard library

- Defines
  - C library
  - I/O stream library
  - containers (complex data types)
    - vector, map, list, heap, …
    - iterators to operate on the data types
  - algorithms
    - sort(), …

- Reference

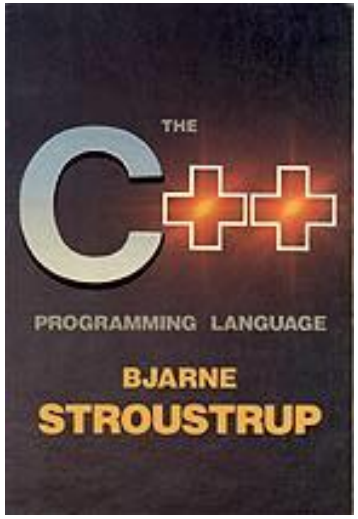
Modern C++

Lambda expressions

- **Unnamed function objects**

  [captures] (params) [-> ret] { body }

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

int main()
{
   vector<int> a;

   for(int i=0;i<20;i++) a.push_back(i+1);

   sort(a.begin(), a.end(), [](const int& x, const int& y) -> bool
    {
        return x > y;
    });

    cout << "Sorted List in decreasing order: \n";
    for(int i=0;i<20;i++) cout << a[i] << " ";
    cout << endl;

    return 0;
}
```



Module Summary

Programming in C++

- **Properties of C++**
  - Extension of the C language
  - Most important features
    - Classes and inheritance
    - Exception handling
    - Templating
    - Namespaces
    - STL with lots of data structures & algorithms

> “C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off.”
>
> — Bjarne Stroustrup