Today’s Goals

• Object-oriented Programming
• Intro to C++

Object-oriented Programming

• C is not designed to write applications
• C is also not designed to write large programs
  ◦ Not just a linear multiplication of code size and programming time
• OOP is a programming paradigm
  ◦ A program is composed of a collection of units (objects)
  ◦ What is the traditional paradigm, i.e. C’s view?

Fundamental Concepts

• Modularity
  ◦ Units are self-contained, easily identifiable and reusable
• Abstraction
  ◦ Implementation of specific functionality can be unspecified
• Encapsulation
  ◦ Internal state of the object cannot be changed in unexpected ways

• Inheritance
  ◦ Objects maybe defined and created from already existing ones
• Polymorphism
  ◦ Allowing the same definition to be applied to different types of data

C++

• An extension of C
• Developed by Bjarne Stroustrup of AT&T Bell labs in the 1980s
• Mostly backwards compatible to C
• Name your C++ programs with extension .cpp or .C
• Use g++ instead of gcc

Minor Conveniences

• Comments – //</t> • Variable declarations anywhere in a function
• Tag names are automatically type names
  ◦ typedef struct _Complex {
    double re, im;
  } Complex;
  ◦ struct Complex (double re, im);
• Keyword void can be omitted
• Default function arguments
  ◦ void new_line (int n=1) {
    while (n-- > 0) putchar(‘\n’);
  }
#include

- New #include style
  - Drop the .h
  - Prepend c to standard C libraries

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

## Namespaces

- A way to group variables and functions under a name.

```cpp
class first {  
  int var = 5;
}
class second {  
  double var = 3.1416;
}

int main() {  
  cout << first::var << endl;
  cout << second::var << endl;
  return 0;
}
```

## Pass by Reference

- Pascal style declaration
- Classic C:
  ```c
  void swap (int *a, int *b) {  
    int temp;  
    temp = *a;  
    *a = *b;  
    *b = temp;
  }
  swap (&x, &y);
  ```

- C++:
  ```cpp
  void swap (int &a, int &b) {  
    int temp;  
    temp = a;  
    a = b;  
    b = temp;
  }
  swap(x, y);
  ```

## Other Use of References

- A function may return a reference

```cpp
doubles bigger (doubles x, doubles s) {  
  if (r>s) return r;
  else return s;
}
```

- Use reference to make a variable be another

```cpp
double a = 1.2;
double b = a; // b is a
```

- b's coupling with a can not be changed

## References in C++

- Reference were invented for people who really did not want to use pointers
- References are far less flexible than pointers
- Trying to avoid pointers by replacing them with references can lead to bad problems
- In general, avoid references all together and learn to use pointers properly
- Or use Java ☝

## Dynamic Allocation

- Instead of `malloc` and `free`, C++ provide `new` and `delete` and `delete[]`

```cpp
int *int_ptr, *array_ptr;
int_ptr = new int;
array_ptr = new int[10];
delete int_ptr;
delete[] array_ptr;
```
Classes

- A class is a declaration of a new data type
- More powerful than `struct` and `typedef` as it includes functions

```cpp
class Fraction {
public:
    void print();
private:
    int num;
    int denom;
    void reduce();
};
void Fraction::print() {
    printf("%d/%d", num, denom);
}
void Fraction::reduce() {
    int d = gcd(denom, num);
    num /= d;
    denom /= d;
}
```

Constructors

- Same as Java constructors – a function with the same name as the class itself with no specified return type

```cpp
class Fraction {
public:
    void print();
private:
    int num;
    int denom;
    void reduce();
};
Fraction f(2,3);
Fraction f2(2); // Fraction f(2,1);
Fraction f3;    // Fraction f(0,1);
```

Constructors

- Every class comes with a default constructor with takes no arguments and does no initialization
- A class may have multiple constructors
- Constructors should be public!
- A constructor may not take an object of its own class as argument, but may take a reference to its own class
- A copy constructor is automatically provided if not specified

```cpp
Fraction::Fraction(Fraction& f) {
    num = f.num; denom = f.denom;
}
```

Destructors

- Destructors are typically not called by a programmer but left to the compiler
- Called whenever an object is destroyed, i.e. by going out of scope or using `delete`
- Need to write destructors if you dynamically allocate memory for your class objects, either in a constructor or in a member function

```cpp
Fraction::~Fraction();
```

Example

```cpp
class BigStr{
    char* str;  // private
    long size;  // private
public:
    BigStr();
    ~BigStr();
};
BigStr::BigStr() {
    str = new char[sizeof(size_t)+1];
    str[0] = '\0';
    size = sizeof(size_t);
}
BigStr::~BigStr() {delete[] str;}
```

Operator Overloading

- Function overloading
  - Multiple functions taking different types are defined with the same name
  - Compiler calls the right one by examining the arguments
- C++ allows the same for built-in operators
Operator Overloading

```cpp
class Fraction {
public:
    ...  
    Fraction operator*(Fraction f);
private:
    ...  
};
Fraction Fraction::operator*(Fraction f) {
    Fraction res;
    res.num = num * f.num;
    res.denom = denom * f.denom;
    res.reduce();
    return res;
}
```

Strings in C++

• C-style strings
  - `#include <string.h>`
• `string` class provided by the standard template library
  - `using namespace std;`  
  - `#include <string>`
  - `string fname = "Dianna", lname = "Xu";`  
  - `string name = fname + " " + lname;`

Inheritance

• C++ inheritance works very much the same way as in Java
• Constructor inheritance rules are similar to those in Java
  - no `super()`, but can invoke explicitly by name
• Method overriding is called virtual functions
  - Late-binding works the same
• C++ supports multiple inheritance

Example

```cpp
class Figure {
public:
    void move(int xinc, int yinc);
    virtual double area();
private:
    int x, y;
};
class Circle : public Figure {
public:
    double area() {return 3.14*radius*radius;}
private:
    int radius;
};
```

Exceptions

• Exceptions are thrown with keyword `throw`
• Exceptions are less structured in C++, and can be practically any type
• Exceptions are caught with `try()` and `catch()`

Access Modifiers

• `public`
• `private`
  - In C++ default is `private` if undeclared
• `protected`
• `friend` – adhoc access to private variables
  - By declaring a function or a class friend, a class allows access to its private data members
I/O in C++

- Standard C I/O still works via stdio.h
- C++ style I/O through iostream.h
  - cin and cout streams
  - overloaded << and >> operators
  ```cpp
  cout << "Enter a number: ";
  cin >> n;
  cout << "The square is: " << n*n << endl;
  ```
- Easier than printf/scanf, but not as flexible and versatile

Mixing C/C++

- Generally not a good idea
  - Use both C and C++ strings
  - Use both references and pointers
  - etc
- Okay to mix in an entire functionality and staying consistent
  - Use only pointers but not references
  - All C++ but with I/O entirely through stdio.h

Summary

- C++ is really a combination of C and Java
- Use C++ in your project whenever appropriate, especially if inheritance is called for.