CS 201
Introduction to c++ (2)

Debzani Deb

Classes (1)

- A class definition – in a header file : .h file
- A class implementation – in a .cc, .cpp file

```
// header file point.h
#include <iostream>
using namespace std;
class point {
    public:
        point (void);
        ~point (void);
        int getX (void) const;
        int getY (void) const;
        int getR (void) const;
    private:
        int x_;
        int y_;
};
```

Indicates that it does not change its caller

Classes (2)

```
// implementation for point.cpp
#include <iostream>
#include "point.h"
using namespace std;
point :: point (void) { x_ = 0; y_ = 0;}
point :: ~point (void) { /* empty */}
int point :: getX (void) const { return x_;}
int point :: getY (void) const { return y_;}
int point :: getR (void) const
        { return sqrt(x_*x_ + y_*y_); }
```


Constructors & Destructors

- Object is an instance of a class.
- A constructor initializes members of an object.

  point :: point (void) { x_ = 0; y_ = 0;}
  - A class can have multiple constructors
  - A default constructor is a constructor that can be called with no arguments.
- Destructors are used to release any resources allocated by an object.
  - No parameters and returns no value

  point :: ~point (void)

Instantiation

```
{
    point a;        /* create an object a of type point.
                       Constructor is called */
}                   // The object is destroyed
point * b = new point();
```

- Creates an object which will not be destroyed unless delete is called.

  delete b;
- Like our old friend: malloc/free pair

Accessing Class Members

```
point a;
int x = a.getX(); //valid
int xx = a.x_; // not valid

point * b = new point ();
int y = b -> getY(); // valid
int yy = b ->y_  // not valid
```

Polymorphism

- Poly means “many” and morph means “forms or shapes”
- Polymorphism - “capable of assuming various forms”
- Here is an example of real-life polymorphism
- Suppose you want to mail something
  - Walk into a post office (PostOffice Object)
    Mail_Item(....)
  - Give them a letter to be mailed and the fee to mail it, and expect it to be mailed properly.
    Void Mail_Item (Letter letter, double posage_fee);
  - To mail a box, we expect the same thing.
    Void Mail_Item (Box box, double posage_fee);

Why is this polymorphism?

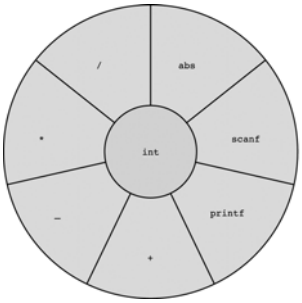
- No matter what type of item you wish to mail, you use the same post office.
  (imagine if there were different post offices for every type of item.)
- You do not have to worry about “how” the item is going to arrive at its destination.
- You simply leave the item at the post office and expect that the item will be sent.

Function Overloading

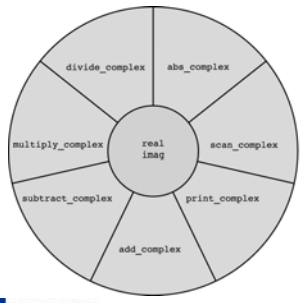
- Two or more functions can share the same name provided that:
  - Type of their arguments differs
    OR
  - Number of their arguments differs
- To overload a function, simply declare and define all required versions.

```
class point {
public:
   //distance to b
   double distance (const point & b);
   // distance to (0,0)
   double distance (void);
};
point a;
double d = a.distance (b);
double d0 = a.distance ();
```

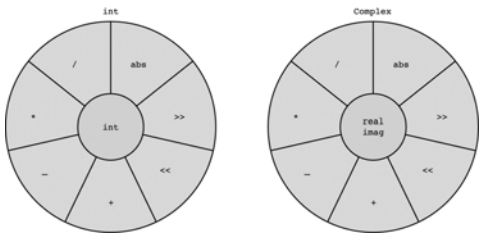

“Donut” Model of Standard Type int



“Donut” Model of an Abstract Data Type



Wouldn’t be more easier to have something like this?



Operator Overloading

- One of C++'s most powerful feature.
- Operator overloading is just a type of function overloading
- An operator is always overloaded relative to a user-defined type, such as a class.
- To overload an operator, you create an operator function.
- An operator function is often a member of the class for which it is defined.
- Syntax:

```
returnType className :: operator# (arg)
{
    // operations to be performed
}
```


Binary Operator

- When a member operator function overloads a binary operator, the function will have only one parameter.
- Consider : X + Y
- X is the object that generates the call to the operator function.
- Y is only a parameter of the operator function called by X.

Example : coord class

```
class coord {
    int x, y; // coordinate values
public:
   // constructor function is overloaded
   coord () { x = ; y = 0; }
   coord (int i, int j) { x = i; y = j;}
    // our new operator function
    coord operator+ (coord ob2) const;
};
coord coord:: operator+ (coord ob2) const {
    coord temp;
    temp.x = x + ob2.x;
    temp.y = y + ob2.y;
    return temp;
}
```

```
int main() {
  coord o1(10, 10), o2(5, 3), o3, o4;
  o3 = o1 + o2;
  // string of addition is also allowed
  o4 = o1 + o2 + o3;
  return 0;
}
```


Example coord class

```
class coord {
    int x, y; // coordinate values
public:
   coord () { x = ; y = 0; }
   coord (int i, int j) { x = i; y = j;}
    // operator function
    coord operator+ (coord ob2) const;
    coord operator- (coord ob2) const;
    void operator= (coord ob2) const;
};
```

- The operator –() can be implemented similarly to operator +().

Example: overloading operator =()

- We focus on the assignment operator function, operator =().

```
void coord :: operator = (coord ob2) {
   x = ob2.x;
   y = ob2.y;
}
int main () {
   coord o1(4,6), o2;
   o2 = o1;
   return 0;
}
```


Why is operator overloading polymorphism?

- Polymorphism is using a common interface to accomplish multiple "expected tasks".
- Because C++ does not automatically define "+" for user-defined classes, we can use operator overloading to add functionality to "+".
- With an overloaded "+' operator, we can now use the "+" interface for adding multiple types of objects as well. We can also keep its original capabilities.

Rules of operator overloading

- The operator must obey the built-in definition on its precedence, associativity, and number of operands.
- When overloaded as a member function, the operator has its associated object as the left-most operand.
- As member functions, unary operators have no arguments, and binary operators have only one.
- Only three operators : ::, ., .* can not be overloaded.

Reasons for using/not using C++

- Advantages
  - C++ is a superset of C.
  - Efficient implementation
  - Low-level and high-level features
  - Portable
  - No need for fancy OOP resources
- Disadvantages
  - A hybrid
  - Little confusing syntax and semantics
  - Programmers must decide between efficiency and elegance.
  - No automatic garbage collection.

Important !!!

- Final: 4 -5.50 pm, Tuesday, 8th May.
- Take Home exercise will be uploaded at 5.00 pm today. You will have 48 hours to submit it.
- Sort out all lab related problem before this Friday. Make appointment with Fuad if you need to see him.
- Course survey: http://www.cs.montana.edu/survey/
- Please go to the lab now and fill out the survey. It is very important for me and the department to have your feed back about this course.

Thanks

- Thank you very much for your attention.
- Thank you for all your efforts in this course.
- It was a pleasure to have you all as my students.