Biomedical Engineering Institute, Bogazici University Instructor: Ahmet Ademoglu, PhD

BM 593 Numerical Methods & C Programming

11th week Object Oriented Programming with C++ Language

Scope resolution operator ::

```
int AllDone;
void AnyFunction(void)
{
  int AllDone;
  AllDone=1; // refers to local variable
  if (::AllDone) // Refers to global variable
    DoSomething();
}
```

const : The const keyword prefixing the name of a variable indicates that the variable is a constant and must not be modified by the program. If a function's argument is a pointer and if that pointer is declared as constant the function cannot modify the contents of the location referenced by that pointer.

```
class shape {
  public:
     virtual void draw(void) const{}
};
class circle_shape : public shape{
  public:
  virtual void draw(void) const;
}
// Create instances of circle and rectangle shapes
circle_shape c1(100.,100.,50.);
rectangle_shape(10.,20.,30.,40);
c1_draw();
r1.draw();
```

Dynamic Binding

```
// shape.h
#include <stdio.h>
#include <math.h>
class shape{
    public:
    virtual double compute_area(void) const{
      printf("Not implemented\n");
      return 0.;
    }
    virtual void draw(void) const{};
}
class circle_shape : public shape{
   private:
     double x,y;
     double radius;
   public:
     circle_shape(double x, double y, double radius);
     virtual double compute_area(void) const;
     virtual void draw(void) const;
}
class rectangle_shape : public shape{
   private:
     double x1,y1;
     double x2,y2;
   public:
     rectangle_shape(double x1, double y1, double x2, double y2);
     virtual double compute_area(void) const;
     virtual void draw(void) const;
}

int i;
shape *shapes[2];
shapes[0]= new circle_shape(100.,100.,50.);
shapes[1]= new rectangle_shape(10.,20.,30.,40);
for (i=0;i<2;i++)  shapes[i]->draw();
```

const member Functions

Use the const keyword after the arguments in the declaration of a member function if that member function does not modify any member variable. This tells the compiler that it can safely apply this member function to a const instance of this class. For example the following is permissible because

```
compute_area
```

is a const member function:

```
const circle_shape c1(100.,100.,50.);
double area = c1.compute_area();
```

Virtual Destructors

```
#include <iostream.h>
class Base{
  public:
     Base() {cout << "Base: constructor" << endl;}
     // destructor should be virtual
     ~Base(){cout << "Base: destructor" << endl;}
};
class Derived : public Base {
  public:
     Derived() {cout << "Derived: constructor" << endl;}
     ~Derived() {cout << "Derived: destructor" << endl;}
};


void main(){
  Base* p_base = new Derived;
  // use the object...
  // Now delete the object
  delete p_base;
}
```

The output is

Base: constructor

Derived: constructor

Base: destructor

If the base class destructor is defined as:

```
virtual ~Base(){cout << "Base: destructor" << endl;}
```

The output will be

Base: constructor

Derived: constructor

Derived: destructor

Base: destructor

friend functions

```
#include <stdio.h>
class complex{
  float real, imag;
  public:
    friend complex add(complex a, complex b);
    friend void print(complex a);
    complex(){real=imag=0.};
    complex(float a, float b){real=a;imag=b;}
};
complex add(complex a, complex b){
  complex z;
  z.real=a.real+b.real;
  z.imag=a.imag+b.imag;
  return z;
}
void print (complex a){
 cout << a.real << "+i" << a.imag <<endl;
}
main(){
  complex a,b,c;
  a=complex(1.5,2.1);
  b=complex(1.1,1.4);
// print and add functions can be accessed from outside the class
  cout << "Sum of ";
  print(a);
  cout << "and";
  print(b);
  c=add(a,b);
  printf(" = ");
  print(c);
}
```

Referencing

```
int i=5;
int *p=&i;
int &r=i;
r+=10;  // adds 10 to i because r is another name for i.

void twice (int &a){
  a*=2;
}
int x=5;
twice(x);
cout << "x=" << x;  // x prints 10
```

Overloaded Operators

```
class complex{
  float real, imag;
  public:
    friend complex operator+(const complex &a, const complex &b);
    complex(){real=imag=0.};
    complex(float a, float b){real=a;imag=b;}
};

complex operator+(const complex &a, const complex &b){
   complex z;
   z.real=a.real+b.real;
   z.imag=a.imag+b.imag;
   return z;
}
complex a,b,c;
a=complex(1.5,2.1);
b=complex(1.1,1.4);
c=a+b;
complex z(1.1,1.2);
cout << z;
ostream& operator<<(ostream& s, const complex& x);
```

```
#include <iostream.h>
class complex {
   float real,imag;
   complex(float a, float b){real=a; imag=b;}
   void print(ostream& s) const;
};
void complex::print(ostream& s) const {
  s<<real << "+" << imag;
}
ostream& operator << (ostream &s, const complex& z){
 z.print(s);
 return s;
}

void main(){
  complex a(1.5,2.1);
  cout << "a= "<< a << endl;
}
```

const member functions

If a member function should not alter any data in the class you sshould declare that member function as const function.

```
size_t length(void) const;
```

This informs the compiler that the length function should not alter any variable in the class. Using Pointer to Class Members

```
class Sample{
 public :
    short step;
    void set_step(short s)
    //...
}
short Sample::*p_s;
p_s=&Sample::step;

Sample s1;
s1.*p_s=5;

Sample s1;
Sample *p_sample1 = &s1;
p_sample->*p_s=5;
```

Pointers to Member functions

```
#include <iostream.h>

class CommandSet{
  public:
     void help(){cout << "Help"' << endl;}
     void nohelp(){cout << "No Help"' << endl;}
  //...
}
void (CommandSet::*f_help)()=CommandSet::help;
main(){
  CommandSet set1;
  (set1.*f_help)();
  f_help=CommandSet::nohelp;
  (set1.*f_help)();
}
```

Pointers as references

```
void swap_int(int &a, int &b){  // instead of void swap_int(int *p_a, int *p_b)
  int temp;
  temp=a;
  a=b;
  b=temp;
}


int x=2,y=3;
swap_int(x,y);
```

Copy Constructor `X(const X&)` (Provide it for any class that allocates memory)

```
class String{
  public:
    String();
    String(size_t len);
    String(const char *str);
    String(const String &s);
    ~String();
    private:
      char *p_c;
      size_t _length;
      size_t _maxlen;
}
String ::String(const String &s)
{
  _length=s.length;
  _maxlen=s.maxlen;
  p_c=new char [_maxlen];
  strcpy(p_c,s.p_c);
}


String s1="Hello";
String s2=s1;
String (const String&);
```

Member Initializer List

```
class Point{
  public:
    Point (double _x=0.0, double _y=0.)
    x=_x;
    y=_y;
}
Point (const Point& p) {x=p.x,y=p.y;}
private:
   double x,y;
};
class Line{
  public:
   Line (const Point& b, Point& e) : p1(b), p2(e) {}


   private :
     Point p1,p2;
};
Line:Line(const Point& b, Point& e){
  p1=b;
  p2=e;
}
```

Operators as functions

```
&x  // x.operator&()
x+y //x.operator+(y)
```

Arguments to operator functions

When declared as a friend the operator function requires all arguments explicitly. This means to declare operator+ as a friend function of class x, you write

```
friend x operator+(x&, x&) // assume x is a class
```

Operator + for string class

```
String s1("This"), s2("and that"),s3;
s3=s1+s2;


String::String::operator+(const String& s){
   size_t len=_length + s._length;
   char *t =new char [len+1];
   strcpy(t,p_c);
   strcat(t,s.p_c);
   String r(t);
   delete [] t;
   return r;
}
String s1="World!";
String s2="Hello,"+s1; // "Hello".operator+(s1) this is an error
```

Solution is to define a friend function which takes two arguments :

```
friend String operator+(const String& s1, const String& s2)
```

the compiler converts Ḧello,+̈s1 to the function call:

```
operator+(String("Hello"),s1)
```

```
String operator+(const String& s1, const String& s2)
{
   size_t len = s1.length + s2.length;
   char *t = new char [len+1];
   strcpy(t,s1.p_c);
   strcat(t,s2.p_c);
   Sring S3(t);
   delete [] t;
   return (S3);
}
```

Assignment Operator for the String class

```
String& String::operator=(const String& s){
  if (this != &s ){
    _length=s._length;
    _maxlen=s._maxlen;
    delete [] p_c;
    p_c= new char[_maxlen];
    strcpy(p_c,s.p_c);
  }
  return *this;
}
```

Overloading the Input/Output Operators

```
#include <iostream.h>
class String {
.
.
}
void String::print (ostream& os) const{
   os << p_c;
}
ostream& operator<<(ostream& os, String& s){
   s.print(os);
   return os;
}

String user_input;
cin >> user_input;
String greetings = "Hello world!";
cout << greetings << endl;
istream& operator>>(istream& is, String& s){
  const bufsize = 256;
  char buf[bufsize];
  if (is.get(buf,bufsize)) s= String(buf);
  return is;
}
```

FILE IO using ifstream, ofstream, fstream

```
 #include <fstream.h>
 ifstream ins("infile");
 ofstream outs("outfile");
 if (!ins) { cerr << "cannot open infile \n"; exit (1);}
```

Alternatively,

```
 ifstream ins;
 ins.open("infile");
 .
 .
 ins.close();
```

opens file in binary format

```
ifstream ins("infile",ios::binary);

ins.eof();

ins.get();

ins.put();
```