BLG252E-OBJECT ORIENTED PROGRAMMING

THIRD PRACTICE SESSION

Question #1

Taken from Final Exam (2011 Spring)

```
class A{
private:
void privatef(){};
public:
void publicf(){};
void print();
void print(int);
protected:
void protectedf(){};
};
```

```
void A::print(){
cout<<"This is an A
object"<<endl;
}
void A::print(int x){
cout<<"The x value
is:"<<x<<endl;
}
```

```
class B: public A{
public:
void print(string);
};
```

```
void B::print(string s){
cout<<s<<endl;
protectedf();
}
class C:A{
};
```

a) What is the difference between method overloading and overriding. Explain and give samples from the code which is given above.

(Hint: Discuss the methods that can be invoked on an A object and a B object.)

b) Discuss the access rights on the inheritance hierarchy given above. (Which methods could be invoked on A, B, C objects and within A,B,C classes.)

c) What is polymorphism? Override the void publicf() method in the class B. And tell the required changes in the code to make it work as polymorphic. Write also a main function which will actuate this polymorphic structure.

d) What would you do if you want to prevent any class user from ever making an object of the A class.

SOLUTION

a)

Overloading is defining multiple methods in the same class with the same name but different signatures.

Example: A::print() and A::print(int)

Overriding is defining two methods, one in a parent class and one in a child class, that generally have the same signature. The method in the derived class overrides hides the ones in the derived class.

Example: A::print()/A::print(int) and B::print(string)

On an A object: public methods of A class can be invoked **void publicf(){};** **void print();** **void print(int);**	**On a B object:** public methods of A class can be invoked due to public inheritance. **void publicf(){};** Also public methods of B class can be invoked. **void print(string);**	**On a C object:** Due to private inheritance, no method can be invoked.
Within A class: all methods of A class can be invoked	**Within B class:** public and protected methods of A class can be invoked **void publicf(){};** **void protectedf(){};** Also all methods of B class can be invoked. **void print(string);**	**Within C class:** public and protected methods of A class can be invoked **void publicf(){};** **void print();** **void print(int);** **void protectedf(){};**

```
class A{
private:
void privatef(){};
public:
virtual void publicf();
void print();
void print(int);
protected:
void protectedf(){};
};
void A::publicf(){
cout<<"A::publicf invoked"<<endl;
}
```

```
class B: public A{
public:
void print(string);
void publicf();
};
void B::publicf(){
cout<<"B::publicf invoked"<<endl;
}
int main(){
A a_obj;
B b_obj;
A *a_ptr;
a_ptr = &a_obj;
a_ptr->publicf();
a_ptr = &b_obj;
a_ptr->publicf();
return 0;
}
```

Polymorphism means taking many shapes and occurs in classes related by inheritance. A call to a member function will cause a different function to be executed depending on the type of the object that gets the message.

d) Make publicf() in A as pure virtual

virtual void publicf()=0;

Question #2 (Final 2011)

Container is a user-written **template class** which can hold an array in the dynamic memory (array) and a single data member (data**).** Array elements and the data member can be from the built-in (such as integers, doubles, and characters etc.) or even user-defined data types. The types of the elements of the array and the data may not be the same. The class includes three services:

setElement sets an element of the array. The index and the new value are given as parameters. Index bound checking is performed and an exception is thrown if necessary. If the element is already stored in any index of the array, an exception is thrown.

smaller compares two containers. A container is smaller than another if the value of the data is smaller or the size of the array is smaller.

operator [] returns the specified element of the array. Index bound checking is performed and an exception is thrown if necessary.

An example main program is given in the next slide.

a) Examine the given part of a program and write the template class Container (declaration and also bodies of all methods except copy constructor and operator=) in C++ programming language. Do not use containers and algorithms of the STL.

b) ClassA and ClassB are user-written classes. Their objects can be stored in the template class Container as shown in the main function. Write only the declarations of ClassA and ClassB, which can be used with class Container.

```
Container<int,char> c1(3,'d'); // Container c1(int*, char), array size: 3, data = 'd'
try{
c1[0]= 1; // ptr[0] is assigned to 1
c1.setElement(1, 2); // ptr[1] is assigned to 2
c1.setElement(2, 2); // Exception: element 2 is already stored in the array
}
catch(const string & msg){ // exception handler
cout << msg << endl;}
char charArray[]= {'a','b'};
Container<char,int> c2(charArray,2,3); // Container c1(char*, int), array size: 2,
data = 3
```

```
int intArray[] = {1,2,3,4};
Container<int,char> c3(intArray,4,'e'); // Container c1(int*, char), array size: 4, data = 'e'
if (c1.smaller(c3)) // returns true
cout << "The first container data or the size of its array is smaller";
ClassA* ptrClassA = new ClassA[2]; // ClassA is a user defined class
ClassA ca;
ClassB cb; // ClassB is a user defined class
Container<ClassA, ClassB> c4(ptrClassA, 2, cb); // Container c4 (ClassA*, ClassB)
c4.setElement(0,ca); // ptr[0] is assigned to ca
```

SOLUTION (a)

```
template <class TypeA, class TypeB>
class Container{
TypeA *array;
TypeB data;
int size;
public:
Container(int, TypeB);
Container(TypeA*, int, TypeB);
void setElement(int, TypeA);
TypeA& operator[](int) const;
bool smaller (const Container<TypeA,TypeB>&) const;
~Container();
};
```

```
template <class TypeA, class TypeB>
Container<TypeA,TypeB>::Container(int aNum, TypeB nData){
array = new TypeA[aNum];
size = aNum;
data = nData;
for (int i= 0; i<aNum;i++)
array[i] = 0;
}
template <class TypeA, class TypeB>
Container<TypeA,TypeB>::Container(TypeA* nPtr, int aNum, TypeB nData){
array = new TypeA[aNum];
size = aNum;
data = nData;
for (int i= 0; i<aNum;i++)
array[i] = nPtr[i];
}
```

```
template <class TypeA, class TypeB>
void Container<TypeA,TypeB>::setElement(int index, TypeA element){
int i;
if (index<0 || index >= size)
throw string("Index out of bounds!\n");
for(i=0;i<size;i++)
if (element == array[i])
break;
if (i!=size)
throw string("This element is already in the array\n");
else
array[index] = element;
}
```

```
template <class TypeA, class TypeB>
TypeA& Container<TypeA,TypeB>::operator[](int index) const{
if (index<0 || index >= size)
throw string("Index out of bounds!\n");
return array[index];
}
template <class TypeA, class TypeB>
bool Container<TypeA,TypeB>::smaller (const Container<TypeA,TypeB>& other)
const{
return (data < other.data || size < other.size);
}
template <class TypeA, class TypeB>
Container<TypeA,TypeB>::~Container(){
delete []array;
}
```

SOLUTION (b)

```
// only declarations for ClassA and ClassB
class ClassA{
public:
	ClassA();
	bool operator==(const ClassA &);  //to be used in setElement method of Container
};

class ClassB{
public:
	ClassB();
	bool operator<(const ClassB &);   //to be used in smaller method of Container
};
```