Object Oriented Programming (CT-260) Lab 07

Friend Functions, Friend Classes, Dynamic Array Class - Case Study

Objectivess

The objective of this lab is to familiarize students with the concept of dynamic arrays, friend functions and friend classes.

Tools Required

DevC++ IDE

Course Coordinator –
Course Instructor –
Lab Instructor –
Prepared By Department of Computer Science and Information Technology
NED University of Engineering and Technology

Dynamic Array

The Array class is defined with private data members size and ptr where size represents the size of the array, and ptr is a pointer to **dynamically** allocated memory to store the array elements.

Sample Code:

```
#include<iostream>
#include <cstdlib>
using namespace std;
class Array{
        int size;
        int *ptr;
        public:
        Array(int s=0):size(s), ptr(NULL){
                if(size>0)
                  ptr=new int[size];
                if(ptr==NULL)
                  exit(1);
        }
        Array(Array& ob){
                if(ob.size>0){
                   size=ob.size;
                   ptr=new int[size];
                   if(ptr==NULL)
                      exit(1);
                   for(int i=0;i<size;i++){
                            ptr[i]=ob[i];
                  }
                }
                else
                  exit(1);
        }
        int& operator[](int in){
                if(in>=size){
                cout<<"index "<<in<<" is out of bound"<<endl;
                exit(1);
                }
                return ptr[in];

        }
        Array operator+(Array& ob){
                if (size==ob.size){
                        Array temp(size);
                        for(int i=0; i<size;i++)
                            temp[i]=ptr[i]+ob[i];
                        return temp;
                }
                else{
                        cout<<"The size of array does not match!"<<endl;
                        exit(1);
                }
        }
```

```
        int operator*(Array& ob){
                if (size==ob.size){
                        int temp=0;
                        for(int i=0; i<size;i++)
                            temp+=ptr[i]*ob[i];
                        return temp;
                }
                else{
                        cout<<"The size of array does not match!"<<endl;
                        exit(1);
                }
        }

        ~Array(){
                delete[] ptr;
        }
};


int main(){
        Array a1(5), a2(5);
        for(int i=0;i<5;i++){
                a1[i]=i*2;
                a2[i]=i+3;

        }
        for (int i=0;i<5;i++)
              cout<< a1[i]<<endl;
        for (int i=0;i<5;i++)
              cout<< a2[i]<<endl;

        Array a3(a1);
        Array a4= a1+a2;
        for(int i=0;i<5;i++)
            cout<<a4[i]<<"\t";
        cout<<endl;
        cout<<a1*a2<<endl;

}
```

The constructor Array(int s=0) is a default constructor that initializes an Array object. It takes an optional parameter s to specify the size of the array. If s is greater than zero, it allocates memory for the array elements using new. If ptr is NULL after allocation, it means memory allocation failed, so it exits the program with an error code.

The copy constructor Array(Array& ob) is used to create a new Array object by copying the values from another Array object ob. It checks if ob.size is greater than zero and allocates memory for the new object. Then, it copies the elements from ob.ptr to the new object's ptr using a loop.

The overloaded indexing operator int& operator[](int in) allows accessing the elements of the array using the subscript operator []. It takes an index in and returns a reference to the

element at that index. Before returning the reference, it checks if the index is out of bounds (greater than or equal to size) and exits the program if it is.

Friend Functions

The concepts of encapsulation and data hiding dictate that nonmember functions should not be able to access an object's private or protected data. The policy is, if you're not a member, you can't get in. However, there are situations where such rigid discrimination leads to considerable inconvenience.

Friend as Bridges

Imagine that you want a function to operate on objects of two different classes. Perhaps the function will take objects of the two classes as arguments, and operate on their private data. In this situation there's nothing like a friend function. Here's a simple example, FRIEND that shows how friend functions can act as a bridge between two classes:

Sample Code 1:

```
#include<iostream>
using namespace std;
class beta;
class alpha{
 private:
    int data;
  public:
    alpha( ): data(3) { }
    friend int frifunc(alpha, beta);
};

class beta{
 private:
     int data;
 public:
     beta( ): data(7) { }
     friend int frifunc(alpha, beta);
};

int frifunc(alpha a, beta b){
    return(a.data+b.data);
}

int main( ){
    alpha aa;
    beta bb;
   cout  <<  frifunc(aa, bb) << endl;
    return 0;
}
```

In this program, the two classes are **alpha** and **beta**. The constructors in these classes initialize their single data items to fixed values (3 in **alpha** and 7 in **beta**).

We want the function **frifunc()** to have access to both of these private data members, so we make it a Friend function. It's declared with the friend keyword in both classes:

friend int frifunc(alpha, beta);

This declaration can be placed anywhere in the class; it doesn't matter whether it goes in the public or the private section.

An object of each class is passed as an argument to the function frifunc(), and it accesses the private data member of both classes through these arguments. The function doesn't do much: It adds the data items and returns the sum. The main() program calls this function and prints the result.

A minor point: Remember that a class can't be referred to until it has been declared. Class beta is referred to in the declaration of the function frifunc() in class alpha, so beta must be declared before alpha. Hence the declaration

class beta;

at the beginning of the program.

Sample Code 2:

```
#include<iostream>
using namespace std;
class Array{
        int size;
        int *ptr;
        public:
        Array(int s=0):size(s), ptr(NULL){
                if(size>0)
                  ptr=new int[size];
                if(ptr==NULL)
                  exit(1);
        }

        int& operator[](int in){
                if(in>=size){
                cout<<"index "<<in<<" is out of bound"<<endl;
                exit(1);
                }
                return ptr[in];

        }
        friend int dotproduct(Array, Array);

};

int dotproduct(Array a1, Array a2){
      if(a1.size=a2.size){
          int temp=0;
           for(int i=0;i<a1.size;i++)
              temp+=a1[i]*a2[i];
              return temp;
        }
        else{
             cout<<"Size mismatch error"<<endl;
```

```
            exit(1);
        }
}

int main() {
   Array a1(5), a2(5);
   for(int i=0;i<5;i++){
                a1[i]=i*2;
                a2[i]=i+3;
        }
        for (int i=0;i<5;i++)
             cout<< a1[i]<<endl;
        for (int i=0;i<5;i++)
            cout<< a2[i]<<endl;
        cout<<dotproduct(a1,a2)<<endl;
        return 0;
}
```

Friend Classes

The member functions of a class can all be made friends at the same time when you make the entire class a friend. The program FRICLASS shows how this looks.

Sample Code

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

class alpha{
private:
  int data1;
public:
          alpha(  ): data1(99) { }
          friend class beta;
};

class beta{
public:
        void func1(alpha a) { cout << "\ndata1=" << a.data1; }
        void func2(alpha a) { cout << "\ndata1=" << a.data1; }
};

int main( ){
  alpha a;
  beta b;
  b.func1(a);
  b.func2(a);
  cout  <<  endl;
  return 0;         }
```

In class alpha the entire class **beta** is proclaimed a friend. Now all the member functions of beta can access the private data of alpha (in this program, the single data item data1).

Note that in the friend declaration we specify that beta is a class using the class keyword:

friend class beta;

We could have also declared beta to be a class before the alpha class specifier, as in previous examples

class beta;

and then, within alpha, referred to beta without the class keyword:

friend beta;

Exercise

1. Create a class for 2D dynamic Array, of type integer, by using a double pointer, that is, a pointer to the array of pointers. The class must have the following methods.

- Default, parameterized, and copy constructor
- Overloaded assignment operator
- Overloaded +, -, and * operator
- Overloaded indexing operator
- Destructor.

Write a test program for verifying the working of the designed class.

---

2. By using the 2D Array class that you created in Problem 1, solve the following problem. Given a 2D Array (matrix) of integers of size m x n with the following two properties:

- Each row is sorted in non-decreasing order
- The first integer of each row is greater than the last integer of the previous row and an integer target, return true if the target is in the matrix otherwise false otherwise. **Hint:** Apply binary search algorithm.

Example 1:

1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3
Output: true

Example 2:

1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13
Output: false

3. A company has decided to update the salaries of the employees for which updation is required in the payroll system so that employees can be paid according to the revised budget. How can you implement the concept of friend class here? Consider there are 2 classes. One is "Employee" having private data members (name, id, designation, salary etc.) and the other is "Payroll". The function for updating salaries can be made inside the "Payroll" class that can access the private member "salary" of the "Employee" class and allow the required updation. Implement this scenario.

4. Repeat Q3 and implement this scenario using the friend function.