06 - Array Sorting

Post-Lab Exercises

Exercise 1

Sorting objects with insertion sort (code in lab0602.cpp). Run and understand this program. Describe its working in detail. Remember that classes and structures are dereferenced in the same way and vector pointers can be created using a syntax slightly different than standard vector declaration. Take help from online language references (e.g. www.cppreference.com) to give your description of the code.

```
//objectSort.cpp
//demonstrates sorting objects (uses insertion sort)
#include <iostream>
#include <string>
#include <vector>
using namespace std;
////////////////////////////////////////////////////////////////
class Person
{
private:
    string lastName;
    string firstName;
    int age;
public:
    //--------------------------------------------------------------
    Person(string last, string first, int a) :    //constructor
    lastName(last), firstName(first), age(a)
    {

    }
    //--------------------------------------------------------------
    void displayPerson()
    {
        cout << "   Last name: " << lastName;
        cout << ", First name: " << firstName;
        cout << ", Age: " << age << endl;
    }
    //--------------------------------------------------------------
    string getLast()                      //get last name
    {
        return lastName;
    }
};  //end class Person
////////////////////////////////////////////////////////////////
class ArrayInOb
{
private:
    vector<Person*> v;                //vect of ptrs to Persons
    int nElems;                       //number of data items
```

```
public:
    //--------------------------------------------------------------
    ArrayInOb(int max) : nElems(0)    //constructor
    {
        v.resize(max);                 //size the vector
    }
    //--------------------------------------------------------------
    //put person into array
    void insert(string last, string first, int age)
    {
        v[nElems] = new Person(last, first, age);
        nElems++;                      //increment size
    }
    //--------------------------------------------------------------
    void display()                    //displays array contents
    {
        for(int j=0; j<nElems; j++)    //for each element,
            v[j]->displayPerson();       //display it
    }
    //--------------------------------------------------------------
    void insertionSort()
    {
        int in, out;

        for(out=1; out<nElems; out++)
        {
            Person* temp = v[out];      //out is dividing line
            in = out;                   //start shifting at out
            //until smaller one found,
            while( in>0 && v[in-1]->getLast() > temp->getLast() )
            {
                v[in] = v[in-1];         //shift item to the right
                --in;                    //go left one position
            }
            v[in] = temp;               //insert marked item
        }  //end for
    }  //end insertionSort()
    //--------------------------------------------------------------
};  //end class ArrayInOb
////////////////////////////////////////////////////////////////
int main()
{
    int maxSize = 100;              //array size
    ArrayInOb arr(maxSize);         //create array

    arr.insert("Evans", "Patty", 24);
    arr.insert("Smith", "Doc", 59);
    arr.insert("Smith", "Lorraine", 37);
    arr.insert("Smith", "Paul", 37);
    arr.insert("Yee", "Tom", 43);
    arr.insert("Hashimoto", "Sato", 21);
    arr.insert("Stimson", "Henry", 29);
    arr.insert("Velasquez", "Jose", 72);
    arr.insert("Vang", "Minh", 22);
    arr.insert("Creswell", "Lucinda", 18);
```

```
cout << "Before sorting:" << endl;
    arr.display();                  //display items

    arr.insertionSort();            //insertion-sort them

    cout << "After sorting:" << endl;
    arr.display();                  //display them again
    return 0;
}  //end main()
```

Code Explanation

The programme in lab0602.cpp demonstrates the use of the insertion sort algorithm to sort a list of employees in alphabetical order using their surnames as the sorting key.

Person Class

This programme makes use of two classes. The first class, Person, is a class used to represent an employee in terms of their first name, last name, and age. This class has a constructor which initializes these member variables for each instance of the class and a `displayPerson()` method which outputs these member variables to the console as a string. The class also has a `getLast()` function which returns the last name of the employee. This is the only getter provided by the class as part of its public interface because only the last name of an Employee object needs to be accessible outside the class in order for a list of such objects to be sorted using `ArrayInOb`.

`ArrayInOb` Class

The second class used by the programme is `ArrayInOb`, which implements a vector-based array of pointers to objects of the `Person` class. The vector uses pointers to `Person` objects because `ArrayInOb` class's primary role (sorting a list of `Person` objects) will require rearranging its objects. Moving actual objects around in memory is computationally more intensive than simply manipulating/switching the addresses to these objects, which is why the class choses to implement the vector with pointers. This is usually the case in any container class which implements a 'has-a' relationship with other classes.

Constructor

The ArrayInOb class has a constructor which resizes the container vector to accommodate the number of elements passed to it as an argument. The insert function uses its arguments to call the Person constructor with the new keyword, which dynamically reserves memory for the object and returns a pointer to it. After initializing the new `Person` object and adding it to the first available location in the vector, the constructor increments the number of elements in the vector/array.

`DisplayPerson`

The display method iterates over the entire vector and uses the arrow operator to access the member function displayPerson() for each pointer to a Person object stored in the vector. The arrow operator is the standard access operator for all pointer-type objects in C++. It is similar (but not identical) to dereferencing in that it access the object stored in the memory location pointed to by the pointer and then accesses a specific member variable or function associated with the resulting object.

Arrow Operator In `ArrayInOb`

The arrow operator is used throughout the rest of the programme to access member variables and member functions of dynamically created, pointer-based objects. For instance, in the `insertionSort` method for `ArrayInOb`, the while loop's condition uses the arrow operator to call the `getLast()` method on a pointer to a `Person` object. The same statement also uses the arrow operator to access the last name of a temporary pointer to a `Person` object created during the insertionSort() method.

`insertionSort()`

The fundamental operation of the `insertionSort()` function involves dividing an array or similar data structure into sorted and unsorted sections. The function then takes each element of the unsorted part, compares it to the elements already present in the sorted section, and inserts it into the right location in the sorted part based on some predefined sorting scheme (ascending order, descending order, etc.).

The `insertionSort()` implementation in this programme initially assumes that the first element in the vector of `Person` objects is sorted (by definition, a single-element data structure is sorted according to any arbitrary ordering scheme). The unsorted section therefore begins with index 1 i.e. the second element in the array. The function stores the address of the first object in the unsorted section of the array in a temporary variable (or rather, it stores the pointer to the first object in a pointer to a temporary `Person` object).

It then compares the last name of the `temp` object with the last names of the `Person` objects in the sorted section of the array. It continues to do so until it either reaches the end of the sorted array or until it finds a `Person` object with a last name that should come after the current last name, as per C++'s built-in string comparison rules. During the while loop, it will continue to shift `Person` objects up the array i.e. to higher indexes until it finds the insertion location. This is essentially making an insertion space for the object from the unsorted part in the sorted part of the array.

The loop terminates when the location for inserting the current `Person` object from the unsorted section is found. It then assigns the object to this position and increments the beginning index of the unsorted section.

The method repeats until the last element of the array is inserted in its right position.

Main

The `main()` method tests these classes and uses them to demonstrate the `insertionSort()` method as applied to a list of `Employee` objects. The method first defines the maximum number of employees that can be entered in the programme and uses this variable to initialize an `ArrayInOb` object called '`arr`'. This means `arr` is actually a vector which has been resized to have a storage capacity for 100 `Person` objects.

The next few lines insert `Employee` data into `arr`, which in turns calls the `Person` constructor with the arguments passed to it. `main` then displays the list of employees stored in `ArrInOb` before sorting and then calls the insertion sort method on `arr`. It then proceeds to display a list of employees that have been arranged in alphabetical order according to their last names.

Sort Stability

The insertion sort operation preserves the original order of `Employees` for employees with the same last name. Since Doc Smith, Lorraine Smith, and Paul Smith appear in the original list in this order and since they have the same last names, the sorted list will also display them in the same order. Such operations are said to be stable: they maintain the secondary ordering of items with the same search key.

Programme Output

```
Before sorting:
   Last name: Evans, First name: Patty, Age: 24
   Last name: Smith, First name: Doc, Age: 59
   Last name: Smith, First name: Lorraine, Age: 37
   Last name: Smith, First name: Paul, Age: 37
   Last name: Yee, First name: Tom, Age: 43
   Last name: Hashimoto, First name: Sato, Age: 21
   Last name: Stimson, First name: Henry, Age: 29
   Last name: Velasquez, First name: Jose, Age: 72
   Last name: Vang, First name: Minh, Age: 22
   Last name: Creswell, First name: Lucinda, Age: 18
After sorting:
   Last name: Creswell, First name: Lucinda, Age: 18
   Last name: Evans, First name: Patty, Age: 24
   Last name: Hashimoto, First name: Sato, Age: 21
   Last name: Smith, First name: Doc, Age: 59
   Last name: Smith, First name: Lorraine, Age: 37
   Last name: Smith, First name: Paul, Age: 37
   Last name: Stimson, First name: Henry, Age: 29
   Last name: Vang, First name: Minh, Age: 22
   Last name: Velasquez, First name: Jose, Age: 72
   Last name: Yee, First name: Tom, Age: 43
Program ended with exit code: 0
```