EEE102 C++ Programming and Software Engineering II

Lab Practice 6

Arrays and Pointers

Notice:

- The aim of this lab is for you to become familiar with the usage of arrays, vectors, pointers and the simple DMA (Dynamic Memory Allocation).
- Practice with the exercises. These parts are not for submission.

1. Arrays and vectors

Exercise 1.1 Answer the questions below.		
1	Assuming that m[200] is an integer array, what do the following expressions mean	
	a)	**`m`**
	b)	**`m+1`**
	c)	**`*(m+1)`**
	d)	**`&m[0]`**

2	Give the following code:
	`Char c[ ] = “Good Morning”;`
	`Char* pc = &c[2];`
	What would you expect the following statements to produce on the screen?	

	a)	**`cout << c;`**
	b)	**`cout << c[3];`**
	c)	**`cout << pc;`**
	d)	**`cout << *(pc-2);`**
3	True or false?	
	a)	Vector subscripts must be integers;
	b)	Vectors cannot contain string as elements;
	c)	A function cannot change the length of a vector that is passed by reference;
	d)	Elements of different columns in a two-dimensional array can have different types.

Exercise 1.2 Implement an algorithm to construct magic *n* x *n* squares when n is odd. Magic *n* x *n* square: sums of every row, column and diagonal are equal.
For example:

8	1	6
3	5	7
4	9	2

and

17	22	1	8	15
21	5	7	14	16
4	6	13	20	25
10	12	19	24	3
11	18	23	2	9

Hit: You may need to declare a 2-dimensional vector. Check the following link for help: http://www.yolinux.com/TUTORIALS/LinuxTutorialC++STL.html

Exercise 1.3

Write a function

```
vector<int> append (vector<int> a, vector<int> b);
```

that appends one vector after another.

For example:

If a is {1, 4, 9, 16} and b is {9, 7, 4, 9, 11}, then append returns the vector {1, 4, 9, 16, 9, 7, 4, 9, 11}.

2. Pointers and DMA

Exercise 2.1

Read the following programs, find out the problem of them and run them.

Line Code

```
// Programme 2.1: Dynamic memory allocation for single values

// A programme reads in two float point values and displays their
// average on the screen.

#include<iostream>
using namespace std;

int main(void)
{
    float *pfv1,*pfv2,*paverage; // declaration of 3 pointers

    //The following statements dynamically allocate memory for 3 float
    // point values

    pfv1=new float, pfv2=new float, paverage=new float;

    cout<<"Type 2 real numbers separated by a space"<<endl;
    cin>>pfv1 >>pfv2; // input two values and keep them at the memory
                      // addresses that pfv1 and pfv2 point to.

    *paverage=(*pfv1+*pfv2)/2;

    cout<<endl<<"The input values are : "<<*pfv1<<" and
"<<*pfv2<<endl;
    cout<<"Their average is "<<paverage<<endl;
}
```

Exercise 2.2

Implement two classes linked by pointers. Test your classes with DMA.

A class **Person** with three data members is defined as follows:

```
class Person
{
    string name;
    Car* firstcar;    // a pointer points to the first car this person owns
    Car* currentcar;  // a pointer points to the current car on access
public:
    Person(string namein, Car* carin=NULL, Car* currentcar=NULL);
    void set_person_name(string namein);
    void set_car(Car* carin);
    void display();
};
```

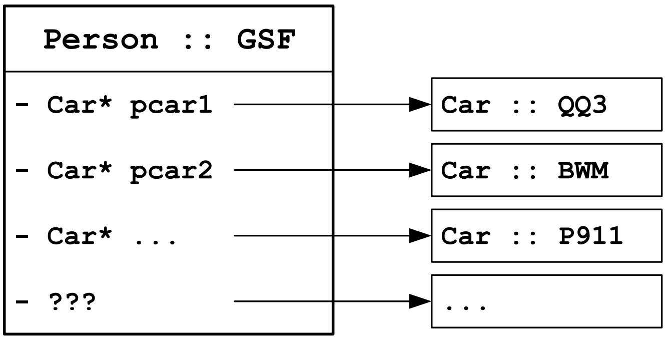
where **Car** is another class defined to illustrate cars.

The Person class is designed to keep a person’s name, and the information of the cars owned by him/her. The cars owned by people are represented by the **Car** class as defined below:

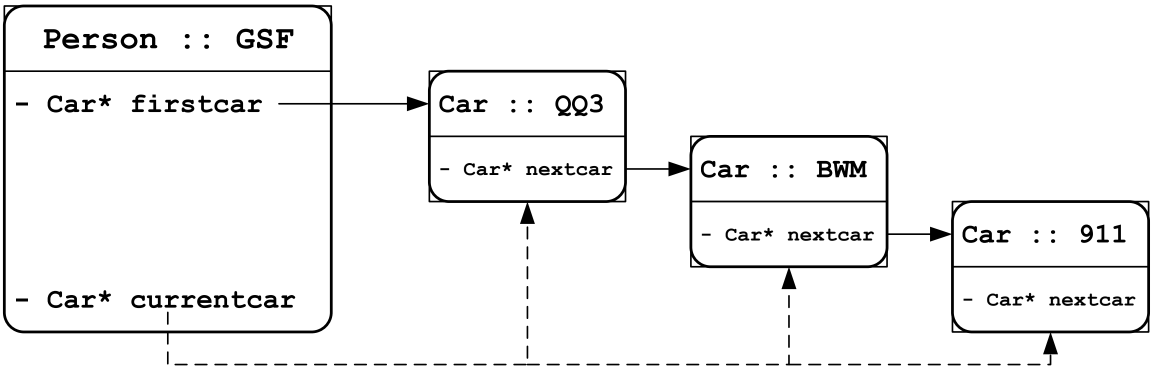
```
class Car
{
    string car_name;
    Car* nextcar;
public:
    Car(string cnamein="Not Given", Car* pcar=NULL);
    void set_car_name(string cnamein);
    void set_next_car(Car* pcar);
    string get_car_name();
    Car* get_nextcar();
};
```

The first pointer “**firstcar**” in **Person** class points to the first car this person owns. Since a person may have more than one car, we have to find a way to show all of them.

One thread is to use several pointers, each of them points to a car, as illustrated by the following figure. However, how many cars does one person have may not be determined during programming stage, so how many pointers should be contained in the Person class is undetermined.



Therefore, in the **Car** class, a pointer **nextcar** is created to point to the next car owned by the same person. If this car is the last one, then **nextcar=NULL**. As shown by the following figure, the multiple cars owned by a person can be linked by the pointers.

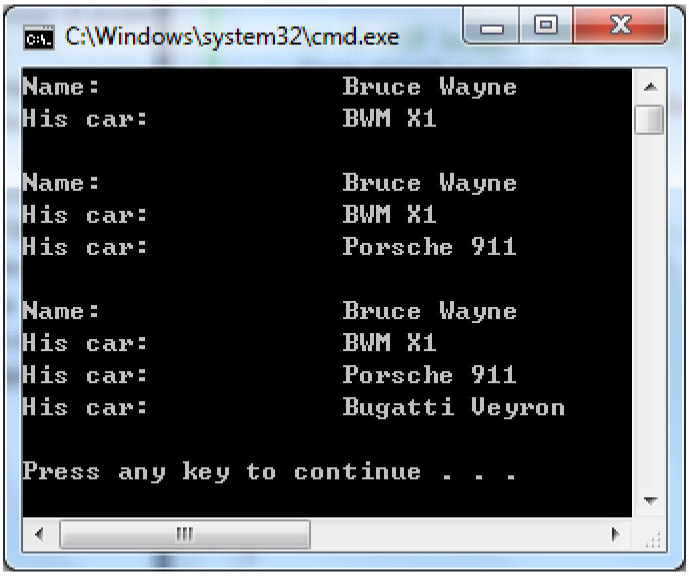


To visit all the cars in the list, a reference pointer is needed to show which car is currently under access, which is defined as "**currentcar**" in the **Person** class. When you go through the list of the cars, it can point to QQ3, BWM and 911 sequentially.

Complete the definition of the methods of these two classes. An example of testing function and its running result are shown below. Use it to test your code.

```
#include <iostream>
#include <string>
#include "person.h"
using namespace std;

int main()
{
    Person GSF("Bruce Wayne");
    // 1. GSF bought the first car: a BMW X1
    Car car1 = ("BWM X1");
    GSF.set_car(&car1);
    GSF.display();

    // 2. GSF bought the second car: a Porsche 911
    Car* pcar2 = new Car;
    pcar2->set_car_name("Porsche 911");
    GSF.set_car(pcar2);
    GSF.display();

    // 3. GSF bought the third car: a Bugatti Veyron
    Car* pcar3 = new Car;
    pcar3->set_car_name("Bugatti Veyron");
    GSF.set_car(pcar3);
    GSF.display();

    delete pcar3, pcar2;
    return 0;
}
```

Running result:



```
Name:                   Bruce Wayne
His car:                BWM X1

Name:                   Bruce Wayne
His car:                BWM X1
His car:                Porsche 911

Name:                   Bruce Wayne
His car:                BWM X1
His car:                Porsche 911
His car:                Bugatti Veyron

Press any key to continue . . .
```