Lab – Structure and pointers

Table of Contents

Getting Started

In this problem set, we will learn about an array of structure along with the following topics:

1. structure – struct, "using alias" in C++ instead of typedef struct in C/C++
2. using new/delete instead of malloc()/free()
3. pointer, pointer arithmetic in a for loop,
4. knowing about pass-by-value, pass-by-pointer, and pass-by-reference
5. passing and using a pointer as a function return value of the structure
6. not using magic numbers
7. using #if and #endif

Structure and Array of Structure

The structure is used to store set of values about an object/entity. We sometime want to store multiple such structure variables for hundreds or objects then Array of Structure is used.

Declaration of Structure in C++

Struct and typedef struct are nicely simplified in C++. It requires to use typedef or struct keyword a lot less than before. Once you declare struct, you may use it as shown below:

```
struct Car {
    string model;
    double speed;
};

Car cars[100];      // you don't need `struct Car` anymore in C++
```

Accessing Structure Fields (members) in Array

We can access individual members of a structure variable as

```
array_name[index].memberName
```

You may access the structure members using a member access operator.

```
cars[5].speed = 10;
```

"using" for alias in C++

In C++, "using" for alias replaces typedef struct in C.

```
struct Car {
    string model;
    double speed;
};

using pCar = Car*;
```

Caution: Once you understand pointers and feel confident, we will not use the alias which hides pointer. Sometimes, that is undesirable. Therefore, we this style of coding temporarily since you are not familiar with pointers.

Use malloc() or new? – That is a question.

When you use any C++ class object in a data structure, use the **new** operator. Otherwise, you may use either **new** or **malloc().**   Don't mix C-style allocation with C++ objects. They don't play well together. Remember that **new** comes with **delete**, malloc() comes with free(). Don't mix them. As shown below, we need to use **new** since Car structure contains a **string**, a C++ object. It is simpler and easier to use. For example, The following code would not work since Car structure uses C++ class called string.

```
Car* list = (Car *) malloc(nCars * sizeof(Car));
Car* p = list;
```

The following two examples work fine.

```
Car* list = new Car[nCars];
Car* p = list;
```

This example code uses "using alias".

```
pCar list = new Car[nCars];
pCar p = list;
```

JoyQuiz: Why are we trying to use a pointer and **new Car[100]** instead of simply using such as **cars[100]**?

Step 1 – Warming-up

Implement a program in a function which gets three car models and speeds and store in the structure and print them one by one.
- Ask the user to enter a model and its speed
- Use **struct Car** array of structure
- Use **NCARS**.

Sample run:

```
C:\WINDOWS\system32\cmd.exe                                    —    □    ×

        Testing Options
        1 - step 1: Using a fixed sized array of structure
        2 - step 2  Using `new` and an array notation
        3 - step 3: Using `new` and a pointer notation
        4 - step 4: Using `pCar`, print_cars(), set_model(), set_speed()
        5 - step 5: Printing car list in main()
        6 - step 6: Using call-by-reference, set_model(), set_speed()
        7 - step 7: Find and fix a memory leak.
        Enter an option(0 to quit): 1
[1/3]
        Enter a model: Porsche
        Enter a speed: 140
[2/3]
        Enter a model: Corvette
        Enter a speed: 170
[3/3]
        Enter a model: Genesis
        Enter a speed: 200
        (1) Porsche      140
        (2) Corvette     170
        (3) Genesis      200
```

Step 2 – No magic number NCARS, using an array notation

In this step, we want to remove the magic number NCARS. Now you must ask the user to enter the number of cars and allocate the memory dynamically for the array of structure to store cars' model and speed.
- Allocate a pointer to an array structure instead of using a fixed size array.
- Don't use NCARS, but use nCars which user entered.
- Use the array notation [] to access members of the structure.

Step 3 – Using pointer and -> notation

In this step, it is the same as step 2 except you use a pointer to access members of the structure instead of the array notation.
- Don't use the array notation [], but use -> notation to access members of a structure.
- Don't forget incrementing the pointer in for loop

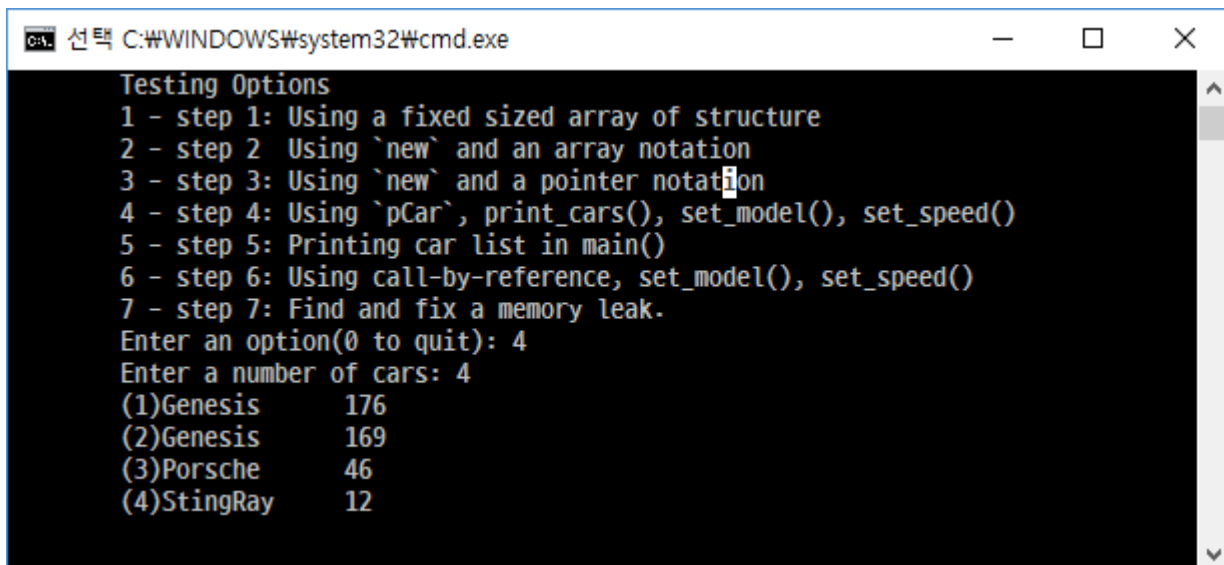# Step 4 – Using alias, implementing set_model(), set_speed()

In this step, it is the same as step 3 except you generate models and speeds randomly instead of asking them to users.

- Use **using pCar = Car*** instead of **Car***
  For example, use **pCar list;** don't use **Car* list**

```
struct Car {
   string model;
   double speed;
};

using pCar = Car*;
```

- Implement set_model() and each model will be selected from a list (or an array of string) randomly.
  void set_model(string *model)
- Implement set_speed() and each speed will be set between 1 and MAX_SPEED = 100 randomly.
  void set_speed(double *speed)
- Implement **print_cars()** which takes two arguments as shown below and print the contents of the Car structure.:
  void print_cars(pCar list, int nCars)

Sample Run:

```
선택 C:₩WINDOWS₩system32₩cmd.exe                                    —    □    ×

    Testing Options
    1 - step 1: Using a fixed sized array of structure
    2 - step 2  Using `new` and an array notation
    3 - step 3: Using `new` and a pointer notation
    4 - step 4: Using `pCar`, print_cars(), set_model(), set_speed()
    5 - step 5: Printing car list in main()
    6 - step 6: Using call-by-reference, set_model(), set_speed()
    7 - step 7: Find and fix a memory leak.
    Enter an option(0 to quit): 4
    Enter a number of cars: 4
    (1)Genesis       176
    (2)Genesis       169
    (3)Porsche       46
    (4)StingRay      12
```

Step 5 – Passing and getting a pointer

In this step, it is the same as Step 4, except it does not print but returns the pointer of the car list such that the caller of the function prints if necessary.

- Make this function return the pointer to the car list.
- In **main()** function, print the list of cars, not in **step5()** function.

Step 6 – Call-by-reference in C++, Call-by-pointer in C/C++

In this step, it is the same as Step 4, except it does use **Call-by-reference** available in C++.

- Copy **step4()** and rename it as **step6().**
- Copy **set_model()** and **set_speed()** as they are, but change the parameter such that each one uses Call-by-reference. Place them above **step6().**
- Now you have multiple function definitions (two set_model() functions, two set_speed() functions) but with different signatures. Two or more functions having same name but different argument(s) are known as overloaded functions.
  The compiler is smart enough to find the right function based on the actual argument during the run time. This technology is called a **function overloading** available in C++, not in C.

Step 7 – Make sure that you don't have a memory leak

In this step, go through the code and make sure that it does not have any memory leak. If you miss one. There will be a penalty.

Submitting your solution

- Include the following line at the top of your every source file with your name signed.
  On my honour, I pledge that I have neither received nor provided improper assistance in the completion of this assignment.
  Signed: _____ Section: _____ Student Number: _____

- Make sure your code **compiles** and **runs** right before you submit it. Every semester, we get dozens of submissions that don't even compile. Don't make "a tiny last-minute change" and assume your code still compiles. You will not get sympathy for code that "almost" works.
- If you only manage to work out the Problem partially before the deadline, you still need to turn it in. However, don't turn it in if it does not compile and run.
- Place your source files in the folder you and I are sharing.
- After submitting, if you realize one of your programs is flawed, you may fix it and submit again as long as it is **before the deadline**. You will have to resubmit any related files together, even if you only change one. You may submit as often as you like. **Only the last version** you submit before the deadline will be graded.

Files to submit

Submit **one source file; struct.cpp.**
NOTE: Don't forget that you have good comments on your source code.

Due and Grade points

- Due: End of Class, Mach 25, 2019
- Grade: 2.0 points total for completion
  - Step 1 ~ 7: 0.25 point per step