References

Chapter 11 – Structured Data

Starting Out With C++. From Control Structures through Objects (Eighth Edition)

A **variable** can hold a value and an **array** can hold multiple items of the same type. But in the real world things may have more complex structure. For example, a person consists of a name, an age, a height, and many other attributes. A student can take many courses and each course has a name, credit, score and others. Thus we can see another abstract data structure is needed. In C++ this abstract data type is called **struct**, which has a general format like:

```
struct structName
{
        type1 component1;
        type2 component2;
         . . .
        typeN componentN;
};
```

Note: The ";" symbol after the closing "}" is needed.

Using the format we may define a Student struct like the following.

```
struct Student
{
        int Id;
        string Name, Course;
        int Credit, Score;
};
```

- In this struct there are six attributes: Id, Name, Course, Credit, and Score. Name and Course are string and all others are integers.
- Multiple components of same type can be in a comma-separated list.

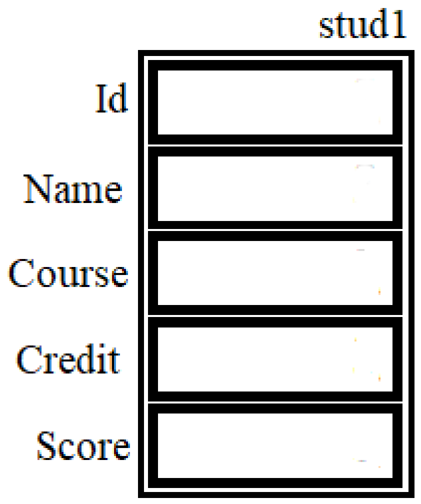
- The struct let us specify that a certain name is associated with an ID, a course, a credit, a score and a grade. (This is a very simply-minded design since only one course can be associated with a name. But this is Ok for now.)

Create object out of struct

The Student struct declared above is just a declaration. We need to create an instance out of the declaration. An instance, which can be seen as a variable, is created in the following way.

```
Student stud1;
```

The Student is somewhat like a blueprint, and we can create a solid instance (or object) out of the blueprint. In this case its name is stud1. After the above instantiation is completed, we have the following object created in the memory.



- 76 bytes of memory are allocated for the new object. (Depending on the operating system and IDE, the size of memory allocated may be different.) This is because that an int field takes 4 bytes and a string takes 32 bytes. After calculating the sum for 4+32+32+4+4, we have 76. (You may have 80 in your test. This is because the system allocates 8 bytes for either one or two ints in a struct.)

- Since none of the fields has been initialized, there is only garbage in these fields.
- The object is named stud1, which is quite similar to the variables which we had learned earlier.

Exercise 1 – Test the size of a struct object and its field

Run the following program see what numbers are displayed.

```
#include <iostream>

using namespace std;

struct Student
{
        int Id;
        string Name, Course;
        int Credit, Score;
};

int main()
{
         Student st;

         cout << "Size of  Student:" << sizeof(Student) << endl;
         cout << "Size of  st:" << sizeof(st) << endl;
         cout << "Size of  st.Id:" << sizeof(st.Id) << endl;
         cout << "Size of  st.Name:" << sizeof(st.Name) << endl;

         return 0;
}
```

Initialization

The Student object stud1 can initialized in the following way.

```
Student stud1 = {12546,"Amy", “CS1”, 4, 81};
```

It can also be initialized piece-by-piece after it is already declared:

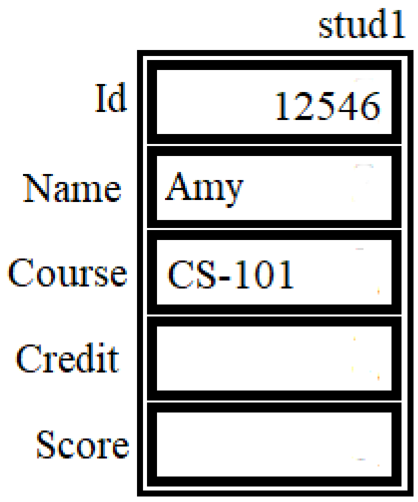
```
Student stud1;

stud1.Id = 12546;

stud1.Name = "Amy";

stud1.Course = “CS-101";
```

Remember the dot (.) symbol "zooms in" to a component of a struct. The symbol before a dot is the whole and symbol after is the part. After the three assignments are executed, stud1 in the memory now looks like this:



Exercise 2 – A struct for citizen

Design a strut for the citizen of a country. Instantiate an object, assign some values to the object and display the values to verify that the values were properly assigned.

Exercise 3 – Nested structs

A struct can be embedded inside another struct. In this case multiple dots have to be used in order to access the inner struct. The following sample shows how this can be implemented.

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

struct Course
{
   string CourseName;
   int Credit;
   int Score;
};

struct Student
{
   int Id;
   string Name;
   Course Cr;
};

int main()
{

   Student stud1;
   stud1.Id = 12546;
   stud1.Name = "Amy";
   stud1.Cr.CourseName = "CS-101";
   stud1.Cr.Credit = 4;
   stud1.Cr.Score = 4;

   return 0;
}
```

Note:

Course is a struct of three fields, i.e. CourseName, Credit, and Score. Course is a field of Student, which is given a field name Cr. When a Student object named stud1 is created, two dots are needed in order to access the fields of Course. The following assignment:

```
        stud1.Cr.CourseName = "CS-101";
```

indicates that the CourseName of Cr or srud1 is assigned a value "CS-101."