WEEK 8: USER DEFINED TYPES – ENUM AND STRUCT

Objectives

Following the completion of this topic, students should be able to:

- Understand the basic concepts of the structure data type
- Know how to declare a structure data type
- Know how to define specific structure variables and initialise them
- Know how to handle functions with structures
- Know how to use nested structures
- Know how to declare and use arrays of structures

TASK 1. ENUMERATED TYPES

An enumerated type is a user-defined type whose values are defined by a list of named constants of type `int`. When defining an enumerated type, you can use any `int` values and can define any number of constants. For example, the following enumerated type defines a constant for the speed limit of each area. Speeds are in Km/h - 1 Km = 0.621371 Mile

```
enum speedLimit {school = 40, residential = 50,
                 builtUp = 60, highway = 100};
```

- If you do not specify any numeric values, the identifiers {`school, residential, builtUp, highway`} in an enumerated type definition are assigned sequential values beginning with 0.
- Definitions for enumerated types are *usually* placed in the global area so that they are accessible by all functions. Once an `enum` type has been defined, you can then declare `enum` variables. *Note: Do not confuse a definition of an enum type with a declaration of an enum variable*.

The following statement,

```
speedLimit speed;
```

declares an enumerated variable of type `speedLimit`. Once declared, values can be assigned to the variable. The following statement,

```
speed = school;
```

assigns the value of `school` to the variable `speed`. Thus, the following statement

```
cout << speed << endl;
```

will produce output of

```
40
```

In a new program, called *'task1.cpp'*:

a. Define an `enum` type named `direction` with the values `NORTH`, `SOUTH`, `EAST` and `WEST`. ***Note: Remember to compile the program regularly during construction.***

b. Declare a variable named `dir` of type `direction`.

c. Assign `NORTH` to the variable `dir`.

d. Output the value of the variable `dir`.

e. Advance `dir` to the next value in the list. Remember that arithmetic is not possible on enumerated types, so casting will be required.

f. Output the value of the variable `dir`.

g. Create a function which outputs the strings "NORTH", "SOUTH", "EAST" and "WEST" instead of the numeric values when passed a `direction` variable. Use a `switch` statement to accomplish this and **not** an `if` statement. The prototype is

```
void printDirection (direction dir);
```

h. Using a `for` loop, call `printDirection` four times. Declare a new `direction` variable and use this to control the loop and pass to the function.

i. Run the program. What is the output?

j. Modify the `for` loop to iterate 5 times. What happened?

TASK 2. BASICS OF STRUCTURES

An array is a list of data of the same type using a single variable name. However, sometimes it is necessary to store related data of varying types such as strings, integers and doubles, etc together in one record. In C++, this can be achieved by creating user defined structure types where each structure contains the relevant members (fields). The following statement,

```
struct myStruct
{
     int member1;
     char member2;
     char member3[10];
     double member4;
};
```

defines a structure type named `myStruct` with 4 separate data members. Like enumerated types, structure types are usually defined in the global area for the same reason: unlimited access to the type. Once defined, the type can be used to declare variables. The statement

```
myStruct struct1;
```

declares a variable named `struct1` of type `myStruct`. It is also possible to assign initial values to a structure variable during the declaration. The following statement:

```
myStruct struct1 = {1, 'a', "string", 34.5};
```

declares a variable named `struct1` of type `myStruct` and sets the value of `member1` to 1, `member2` to '`a`', `member3` to "`string`" and `member4` to 34.5.

- Values are assigned consecutively left to right to the first to last members.
- Omitted members are assigned the value 0. For example

  ```
  myStruct struct1 = {};
  ```

  will set all members to 0.

After declaration of the variable, each member ***must*** be accessed individually, as in array access. To identify each member in the structure variable, you must use the member operator '**.**' stating the variable name along with the member's name such as;

```
struct1.member1
```

For example:

```
struct1.member1 = 1;
```

will assign the value 1 to the member named member1 within the variable `struct1`.

At a member level, you are manipulating `ints, floats, doubles, char[],` etc and the rules that apply to manipulation of these types apply to structure members. For example, you cannot use the '=' operator to assign a string to a *c-string* variable. However, you can assign one structure variable to another of the same type. For example

```
myStruct struct2;
struct2 = struct1;
```

will result in each of the members in `struct1` being assigned to each of the `members` in `struct2.`

Create a program *'task2.cpp'* as follows:

a. Define a structure data type named `BookInfo` that stores the following data.
   i. Title: a string with a maximum of 50 characters.
   ii. Author: a string with a maximum of 30 characters.
   iii. Price:

b. Declare a `BookInfo` variable named `myBook1` and initialises its individual members (members) as "The Da Vinci Code", "Dan Brown" and 24.20 respectively.

c. Output the contents of `myBook1`, being sure to format the output appropriately. (Do not create a function for displaying. It should be done inside the main function).

d. Declare a second `BookInfo` variable named `myBook2` that will have the same values as that of `myBook1`. That is, `myBook2` will be a copy of `myBook1`. There are two ways to achieve this. What are they and which do you think is best?

e. Change the title and price of `myBook2` to "Digital Fortress" and 14.78 respectively.

f. Output the contents of both variables in a tabular format.

TASK 3. STRUCTURES AND FUNCTIONS

As with any other type, structure types may be formal parameters to functions. Likewise, they may be value or reference parameters. Complete the following.

a. Modify the code in*'task2.cpp'*, as *'task3.cpp'*, defining a function to output the contents of a `BookInfo` variable. Move the code from main, which currently does this, to the function. The function will print ***one*** `BookInfo` variable each time it is called.

b. Define a function that will get user input to set the values of the members of the given `BookInfo` variable.

c. Define a function that will reduce the price of the given `BookInfo` variable by the given percentage. For example, the function could be called as:

```
reducePrice (book1, 35);
```

d. Define a function that will compare two `BookInfo` variables to test if they are the identical. The return value of this function should indicate whether they are identical or not.

TASK 4. NESTED STRUCTURES

The members of a structure type can be of any built-in or user-defined type, so long as it has been defined prior to inclusion. Thus, a structure type can be a member of another structure type. These are referred to as nested structures. For example:

```
struct struct1
{
     int member1;
};
struct struct2
{
     int member1;
     struct1 member2;
};
```

- `struct1.member1` will enable access to the integer `member1` of `struct1`.
- `struct2.member1` will enable access to the integer `member1` of `struct2`.
- access to `member2` of `struct2` is different to that of the two examples above as `member2` is a structure itself. Therefore, it is necessary to identify the appropriate member within `struct2.member2` you require. I.e.
  s`truct2.member2.member1`

There is no limit to the depth of nested structures, but as always, readability prevails.

a. Using the program *'task2.cpp'* copied to *'task4.cpp'* as a basis, modify the structure `BookInfo` defined in Task 2 to include a structure named `PublisherInfo` whose members are:
   i. Name (of publisher): A string with a maximum of 50 characters.
   ii. Year (of publication):

b. Modify the initialisation of `myBook1` made in Task 1 to incorporate initialisation of the publisher and published year as "Doubleday" and 2004 respectively.

c. Modify the print and input functions to include the publisher data.

d. Test your program for correctness.

.

TASK 5. ARRAYS OF STRUCTURES

As with other types, you can construct arrays of structures. Assuming that a structure type named `Book` was previously defined, the following statement

```
Book myBooks[10];
```

declares an array of type `Book` with size of 10. To access each record in the array and the internal members you need to state the array name, the index number and the member name, such as:

```
myBooks[index].member
```

Now write a C++ program *'task5.cpp'*, which does the following:

a. Define a structure named `Car`, which contains 4 members:
   i. Make: A string with a maximum of 20 characters. (*E.g*. Holden, Toyota)
   ii. Model: A string with a maximum of 20 characters. (E.g. Camry, Commodore)
   iii. Year (of manufacture): an integer
   iv. Price: a real value

b. Declare an array of type `Car` named `carInStock` with 5 records and initialise them as follows:

carInStock[0]:	carInStock[1]:	carInStock[2]:	carInStock[3]:
make: Toyota	make: Holden	make: Mazda	make: Ford
model: Camry	model: Barina	model: Astina	model: Fairmont
year: 2002	year: 2001	year: 1993	year: 1995
price: 23290	price: 11000	price: 8200	price: 8500

c. Define a function that gets user input to populate the members of a `Car` variable and set the values of the last record in the array.

d. Define a function that identifies the `Car`, which is the most expensive in the array and prints its contents along with an appropriate message. Be sure to use a loop to iterate through the array.