Lab 2 **Structures**

2.1. Introduction

There are many instances in programming where we need more than one variable in order to represent something. For example, to represent yourself, you might want to store your name, your birthday, your height, your weight, or any other number of characteristics about yourself. Structures are a way of storing many different values in variables of potentially different types under the same name. This makes it a more modular program, which is easier to modify because its design makes things more compact. Structs are generally useful whenever a lot of data needs to be grouped together--for instance, they can be used to hold records from a database or to store information about contacts in an address book. In the contacts example, a **struct** could be used that would hold all the information about a single contact--name, address, phone number, and so forth.

As structure is a group of data elements grouped together under one name. These data elements, known as members, can have different types and different lengths. Data structures are declared in C++ using the following syntax:

```
struct structure_name {
member_type1 member_name1;
member_type2 member_name2;
member_type3 member_name3;
.
.
} object_names;
```

where structure_name is a name for the structure type, object_name can be a set of valid identifiers for objects that have the type of this structure. Within braces { } there is a list with the data members, each one is specified with a type and a valid identifier as its name.
The first thing we must know is that a data structure creates a new type: Once a data structure is declared, a new type with the identifier specified as structure_name is created and can be used in the rest of the program as if it was any other type. For example:

```
struct product{
      int weight;
      float price;
};

product apple;
product banana, melon;
```

It is important to clearly differentiate between what is the structure type name, and what is an object (variable) that has this structure type. We can instantiate many objects (i.e. variables, like apple, banana and melon) from a single structure type (product).

Once we have declared our three objects of a determined structure type
(apple, banana and melon) we can operate directly with their members. To do that we use a dot (.) inserted between the object name and the member name. For example, we could operate with any of these elements as if they were standard variables of their respective types:

```
apple.weight
apple.price
banana.weight
banana.price
melon.weight
melon.price
```

Each one of these has the data type corresponding to the member they refer to: apple.weight, banana.weight and melon.weight are of type int, while apple.price, banana.price and melon.price are of type float.
Here is an example program:

Example 2-1 Structures

Code

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

struct Books
{
      char title[100];
      char author[50];
      char subject[100];
      int book_id;

};

int main()
{
      //2 possible ways to declare a structure variable
      struct Books book1;
      Books book2;

      strcpy(book1.title, "Data Structures using C++");
      strcpy(book1.author, "DS Malik");
      strcpy(book1.subject, "Programming" );
      book1.book_id = 1234;

      strcpy(book2.title, "Data Structures & Algorithm Analysis in C++");
      strcpy(book2.author, "Mark Allen Weiss");
      strcpy(book2.subject, "Programming" );
      book2.book_id = 5678;

      cout<<"Book1 title : " << book1.title<<endl;
      cout<<"Book1 author : " << book1.author<<endl;
      cout<<"Book1 subject : " << book1.subject<<endl;
      cout<<"Book1 id: " << book1.book_id<<endl;
      cout<<endl;
      cout<<"Book2 title : " << book2.title<<endl;
      cout<<"Book2 author : " << book2.author<<endl;
      cout<<"Book2 subject : " << book2.subject<<endl;
      cout<<"Book2 id: " << book2.book_id<<endl;

}
```

Output

```
Book1 title : Data Structures using C++
Book1 author : DS Malik
Book1 subject : Programming
Book1 id: 1234

Book2 title : Data Structures & Algorithm Analysis in C++
Book2 author : Mark Allen Weiss
Book2 subject : Programming
Book2 id: 5678
```

2.2. Structures as Function Arguments

You can pass a structure as a function argument in very similar way as you pass any other variable or pointer. You would access structure variables in the similar way as you have accessed in the above example:

Example 2-2 Structure and functions

Code

```
//structure name passing as an argument to a function

#include<iostream>
#include<cstring>
using namespace std;

void printBook(struct Books book);

struct Books
{
      char title[100];
      char author[50];
      char subject[100];
      int book_id;

};

int main()
{
      //2 possible ways to declare a structure variable
      struct Books book1;
      Books book2;

      strcpy(book1.title, "Data Structures using C++");
      strcpy(book1.author, "DS Malik");
      strcpy(book1.subject, "Programming" );
```

```
	book1.book_id = 1234;

	strcpy(book2.title, "Data Structures & Algorithm Analysis in C++");
	strcpy(book2.author, "Mark Allen Weiss");
	strcpy(book2.subject, "Programming" );
	book2.book_id = 5678;

	cout<<"Book1: "<<endl;
	printBook(book1);

	cout<<endl<<"Book2: "<<endl;
	printBook(book2);



}

void printBook(struct Books book)
{
	cout<<"Book title : " << book.title<<endl;
	cout<<"Bookauthor : " << book.author<<endl;
	cout<<"Book subject : " << book.subject<<endl;
	cout<<"Book id: " << book.book_id<<endl;
}
```

Output

```
 Book1 title : Data Structures using C++
 Book1 author : DS Malik
 Book1 subject : Programming
 Book1 id: 1234

 Book2 title : Data Structures & Algorithm Analysis in C++
 Book2 author : Mark Allen Weiss
 Book2 subject : Programming
 Book2 id: 5678
```

2.3. Nesting structure

When a structure contains another structure, it is called nested structure. For example, we have two structures named Address and Employee. To make Address nested to Employee, we have to define Address structure before and outside Employee structure and create an object of Address structure inside Employee structure.
Syntax for structure within structure or nested structure

```
struct structure1
{
	- - - - - - - - - -
	- - - - - - - - - -
};

struct structure2
{
	- - - - - - - - - -
	- - - - - - - - - -
```

```
                        struct structure1 obj;
                };
```

Example 2-3 Nested structures

Code

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

struct movies{
      string title;
      int year;
};

struct friends{
      string name;
      string email;
      movies fav_mov;
}frnd1, frnd2;

friends *pfriends = &frnd2;

int main()
{
      frnd1.name = "Charlie";
      frnd1.fav_mov.title = "Harry Potter";
      frnd1.fav_mov.year = 2001;

      cout<<"Name: "<<frnd1.name;
      cout<<endl<<"Fav mov: "<<frnd1.fav_mov.title<<endl;
      cout<<"Movie year: "<<frnd1.fav_mov.year<<endl<<endl;

      pfriends->name = "Maria";
      pfriends->fav_mov.title = "Tangled";
      pfriends->fav_mov.year = 2010;
      pfriends->email = "abc@gmail.com";

      cout<<"Name: "<<pfriends->name;
      cout<<endl<<"Fav mov: "<<     pfriends->fav_mov.title  <<endl;
      cout<<"Movie Year: "<<pfriends->fav_mov.year <<endl;
      cout<<"Email: "<<pfriends->email<<endl;

}
```

Output

```
Name: Charlie
Fav mov: Harry Potter
Movie year: 2001

Name: Maria
Fav mov: Tangled
Movie Year: 2010
Email: abc@gmail.com
```

2.4. Pointers to Structure

As we have learnt a memory location can be accessed by a pointer variable. In the similar way a structure is also accessed by its pointer. The syntax for structure pointer is same as for the ordinary pointer variable. In general, the structure pointer is defined by the statement
struct-type *sptr;
Where **struct-type** refers to structure type specifier, and **sptr** is a variable that points to the structure. It must be ensured that the pointer definition must preceed the structure declaration. For example, a pointer to struct employee may be defined by the statement
struct employee *sptr;
In other words, the variable **sptr** can hold the address value for a structure of type **struct** employee.

We can create several pointers using a single declaration, as follows:

```
struct employee *sptr1, *sptr2, *sptr3;

We have a structure:
struct employee{
char name[30];
int  age;
float salary;
};
```

We define its pointer and variable as following:

```
struct    employee *sptr1, emp1;
```

A pointer to a structure must be initialized before it can be used anywhere in program. The address of a structure variable can be obtained by applying the address operator & to the variable. For example, the statement

```
sptr1 = &emp1;
```

2.4.1. Accessing Structure Members Using Arrow Operator

The members of a structure can be accessed using an arrow operator. The arrow operator -> (consisting of minus sign (-) followed by the greater than (>) symbol). Using the arrow operator, a structure member can be accessed by the expression

```
sptr1->member-name
```

Where sptr holds the address of a structure variable, and member-name is the name of a member belonging to the structure. For example, the values held by the members belonging to the structure emp1 are given by the expression:

```
sptr1->name
sptr1->age
sptr1->salary
```

Example 2-4 Pointer to structures

Code

```
// pointers to structures
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

struct movies_t {
  string title;
  int year;
};

int main ()
{
  string mystr;

  movies_t amovie;
  movies_t * pmovie;
  pmovie = &amovie;

  cout << "Enter title: ";
  getline (cin, pmovie->title);
  cout << "Enter year: ";
  getline (cin, mystr);
  (stringstream) mystr >> pmovie->year;

  cout << "\nYou have entered:\n";
  cout << pmovie->title;
  cout << " (" << pmovie->year << ")\n";

  return 0;
}
```

Output

```
Enter title: Invasion of the body snatchers
Enter year: 1978

You have entered:
Invasion of the body snatchers (1978)
```

2.5. Example Problems

Example 2-5 Structures Example Problem

Problem Statement

A University needs a student management system. A student has a name and registration number. Design a C++ program that can store a list of students and then display students' details on console.

Your program must:

Use struct for storing student details.

Have a function to take the students' data from the user

Have a function to display the students' data on console screen

Solution

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

struct Student{
	string name;
	int regNo;
};

void initialize(Student students[], int SIZE);
void display(Student students[], int SIZE);

int main()
{
	const int SIZE = 3;
	Student students[SIZE];
	initialize(students, SIZE);
	display(students, SIZE);
	return 0;
}

void initialize(Student students[], int SIZE)
{
	cout<<"Read Data"<<endl;
	for(int i=0; i < SIZE; i++)s
	{
		cout<<"-------------------------------------------------"<<endl;
		cout<<"\tStudent "<<i+1<<endl;
		cout<<"-------------------------------------------------"<<endl<<endl;
		cout<<"Enter Reg. No.: ";
		cin>>students[i].regNo;
		cin.ignore();
		cout<<"Enter name: ";
		getline(cin, students[i].name);
		cout<<endl;
	}
}

void display(Student students[], int SIZE)
{
	cout<<"----------------------------------------"<<endl;
	cout<<"\tList of students "<<endl;
	cout<<"------------------------------ "<<endl;
	cout<<"Reg. No\t\tName"<<endl;
	cout<<"------------------------------ "<<endl;
	for(int i=0; i < SIZE; i++)
	{
		cout<<students[i].regNo<<"\t\t"<<students[i].name<<endl;
	}
}
```

Output

```
Read Data
------------------------------------------
        Student 1
------------------------------------------

Enter Reg. No.: 01
Enter name: Ali

------------------------------------------
        Student 2
------------------------------------------

Enter Reg. No.: 02
Enter name: Ahmad

------------------------------------------
        Student 3
------------------------------------------

Enter Reg. No.: 03
Enter name: Talha

------------------------------------------
        List of students
------------------------------------------
Reg. No         Name
------------------------------------------
1               Ali
2               Ahmad
3               Talha

 ---------------------------------------------
```