Lab 4:
Structures I

- Structure Declaration
- Structure Definition
- Structure Variables
- Structure Membership
- Arrays of structures

STRUCTURES

Arrays require that all elements be of the same data type. Many times it is necessary to group information of different data types. An example is a materials list for a product. The list typically includes a name for each item, a part number, dimensions, weight, and cost.

C and C++ support data structures that can store combinations of character, integer floating point and enumerated type data. They are called a ***STRUCTS.***

Often multiple variables must be passed from one function to another, and often these variables have different data types. Thus it is useful to have a container to store various types of variables in. Structs allow the programmer to do just that

A *struct* is a derived data type that consists of members that are each fundamental or derived data types.

struct is used to declare a new ***data-type***. Basically this means grouping variables together.

Structures Definition

Before a structure is created, it is necessary to define its overall composition. The format of the a structure is provided in the shape of a *template* or pattern which is then used to creat***structure variables*** of the same composition. The template is composed of the names and attributes of the data items to be included in the structure. The definition begins with the keyword ***struct***which is followed by a structure declaration consist of a set of user-defined data names and data types. These entries are separated by semicolons and enclosed within a pair of curly brackets. The definition ends with a semicolon. Thus, in general, the structure definition has the form

```
struct tag-name{
                    type var-1;
                    type-var-2;
                    . . . . . .
                    typevar-n;
                };
```

where***tag-name*** is the user-supplied name that identified the structure template; ***type*** refers to any valid data type such as *char, int, float, and so forth*; and *var-1, var-2, ….var-n* are user-defined variables

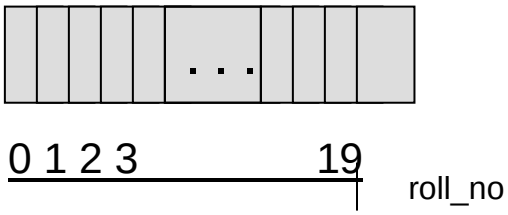
names, arrays or pointers. The components of a structure are commonly referred to as ***members or field.***

Example 4.1:

```
struct employee
{
        char name[30];
        int age;
        float salary;
}
```

Representation in Memory

```
struct student
{
  char name[20];
introll_no;
};
struct student st;
```



Structures Variables

This code fragment illustrates how structure variables of the type ***struct*employee** are defined.

```
struct employee{
        char name[30];
        int age;
        float salary;
};
structemployee emp1, emp2, emp3;
```

In the above example three structure variables **emp1, emp2, and emp3** are created. Each structure consists of three members identified by name, age, and salary.

OR

```
struct employee
        {char name[30];
            int age;
            float salary;

        } emp1, emp2, emp3;
```

Structure membership

Members themselves are not variables they should be linked to structure variables in order to make them meaningful members.

The link between a member and a variable is established using the member operator '.' i.e. dot operator or period operator. We access individual members of a structure with the .operator. For example to assign a value, we can enter:

```
structx
{int a;
int b;
int c;
};

  main()
  {
struct x z;

z.a = 10; // assigns member 'a' of structure variable z  value  10.
z.b = 20;
z.c = 30;
  }
```

And to retrieve a value from a structure member:

```
struct x
  {
int a;
int b;
int c;
  };

  main()
  {    struct x z;
z.a = 10; // Assigning a value to a member through a structure variable
z.a++;  // incrementing the member variable.
cout<<z.a;  // Retrieving the value.
```

```
}
```

Example 4.2:

```
structlib_books
{
char title[20];
char author[15];
int pages;
float price;
} book1,book2,book3;
```

Now
Book1.price
is the variable representing the price of book1 and can be treated like any other ordinary variable. We can use scanf statement to assign values like

```
cin>>book1.pages;
```

Or we can assign variables to the members of book1

```
strcpy(book1.title,"basic");
strcpy(book1.author,"Balagurusamy");
book1.pages=250;
book1.price=28.50;
```

Accessing Structure Variables example:

Example 4.1:

```
int main ()
{
        struct Library
        {
                int ISBN, copies, PYear;
                char bookName[30], AuthorName[30], PublisherName[30];
        };

 Library libraryVariable;
LibraryVariable.ISBN = 1293;
        strcpy (libraryVariable.bookName, "Network Security");
        strcpy (libraryVariable.AuthorName, "Martin");
        strcpy (libraryVariable.PublisherName, "Waley");
        libraryVariable.copies = 4;
        libraryVariable.PYear = 1998;
```

```
cout<<"ISBN Number: "<<libraryVariable.ISBN<<endl ;
cout<<"Library book Name: "<<libraryVariable.bookName<<endl;
cout<<"Author :"<<libraryVariable.AuthorName<<endl;
cout<<"Publisher :  "<<libraryVariable.PublisherName;
cout<<"No. of Copies : "<<libraryVariable.copies;
cout<<"Year : "<<libraryVariable.PYear;
    }
```

Creating multiple structure variables.

```
void main (void)
{
        struct Library
        {
                int ISBN, copies, PYear;
                char bookName[30], AuthorName[30], PublisherName[30];
        };
        Library libraryVariable1, libraryVariable2, libraryVariable3, libraryVariable4;
        Library LibraryArray [4]; // alternative and easiest way
}
```

Assignment a structure variable to another Structure Variable

The value of a structure variable can be assigned to another structure variable *of the same type*, e.g:

```
Library libraryVariable1, libraryVariable2;
strcpy (libraryVariable1.bookName , "C Programming");
libraryVariable1.ISBN = 1293;
libraryVariable2 = libraryVariable1;
```

STRUCTURE WITHIN A STRUCTURE/ NESTED STRUCTURES

Structures can be implemented with functions and arrays. Moreover, structures can be implemented as the member of other structure. This is termed as structure within structure.

When a structure is decleared as the member of another structue, it is called **Structure within a structure.** It is also known as **nested structure**.

Accessing Structure in Structure

```
cout<<universityVariable.libraryVariable.bookName;
cout<<universityVariable.city;
cout<<universityVariable.libraryVariable.bookName);
cout<<universityVariable.libraryVariable.ISBN);
```

Common Errors in Accessing Structures

```
struct x
  {
int a;
int b;
int c;
  };
main()
  {
struct x z;
z.a = 10;
z.a++;
cout<<" first member is “<< a;  // Error!
// a is not a variable. It is only the name of a member in a structure
cout<<”first member is "<<x.a;  // Error!
// x is not the name of a variable. It is the name of a type
  }
```

Arrays of structures

It is possible to define a array of structures for example if we are maintaining information of all the students in the college and if 100 students are studying in the college. We need to use an array than single variables. We can define an array of structures as shown in the following example:

```
structure information
{
intid_no;
char name[20];
char address[20];
char combination[3];
int age;
}
student[100];
```

An array of structures can be assigned initial values just as any other array can. Remember that each element is a structure that must be assigned corresponding initial values as illustrated below.

```
int main (void)
{      struct info
  {
intid_no;
  char name[20];
  char address[20];
int age;
  }std[100];


intI,n;
cout<<"Enter the number of students "<<endl;
```

```
cin>>n;
cout<<endl<<"Enter Id_no,name address combination agem"<<endl;
for(I=0;I <n;I++)
      {
cout<<endl<<"\tEnter Record for student "<<I+1<<endl;
cout<<"Student ID:"<<endl;
cin>>std[I].id_no;
cout<<"Student Name: "<<endl;
cin>>std[I].name;
cout<<"Student city: "<<endl;
cin>>std[I].address;
cout<<"Student Age: "<<endl;
cin>>std[I].age;
       }
cout<<endl<<"Student information"<<endl;
for (I=0;I<n;I++)
      { cout<<endl<<"Student ID:\n\t"<<std[I].id_no<<endl;
cout<<"Student Name: "<<std[I].name<<endl;
cout<<"Student city: "<<std[I].address<<endl;
cout<<"Student Age: "<<std[I].age;
      }
   }
```

Note:
Before doing your lab exercises run the examples.
Exercises will be provided by the instructors in the lab.