Lecture 3

Structures:

Structures are typically used to group several data items together to form a single entity. It is a collection of variables used to group variables into a single record. Thus a structure (the keyword **struct** is used in C++) is used. Keyword struct is a data-type, like the following C++ data-types (int, float, char, etc...). This is unlike the array, which all the variables must be the same type. The data items in a structure are called the members of the structure.

General Form of Structure:

```
1.  struct struct_name
2.  {
3.      DataType member1;
4.      DataType member2;
5.      DataType member3;
6.  } ;
```

For example:

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

The syntax for accessing a struct member is

```
structVariableName.memberName
```

The dot (.) is an operator, called the member access operator.

This example uses parts inventory to demonstrate structures.

```
#include <iostream>
#include <conio>
struct Data
 {
 int x; int y;
 };
void main()
{
 Data D;
D.x = 21;
 D.y = 23;
cout<<D.x<<endl<<D.y;
 getch();
 }
```

Example

Write program to input and print student's data (name ,number and address) using struct

```
#include <iostream>
#include <conio>
void main ()
{
```

```
struct student
{
int no;
char name;

char address;
};
student st;
cout<<"enter the student number:\n";
cin>>st.no;
cout<<"enter the student name:\n";
cin>>st.name;
cout<<"enter                address:\n";
cin>>st.address;
cout<<"student number ="<<st.no<<"\n";
cout<<"student name ="<<st.name<<"\n";
cout<<"student address ="<<st.address<<"\n";
getch();
}
```

Functions and Structures:

A structure can be passed to a function as a single variable. The scope of a structure declaration should be an external storage class whenever a function in the main program is using a structure data types. The field or member data should be same throughout the program either in the main or in a function.

For example

```
#include <iostream>
#include <conio>
 struct Data
 {
 int x; int y;
 void print()
 {
 cout<<x<<" "<<y<<"\n";
}
 };
void main()
{
Data D;
 cout<<"plz enter two numbers\n";
cin>>D.x>>D.y;
D.print();
```

```
getch();
}
```

Initialize structure

You can initialize a structure at the time that it is declared. To give a structure variable a value, follow it by an equal sign and a list of the member values enclosed in braces. For example,

```
#include <iostream>
 #include <conio>
 struct Data
{
 int x; int y;
 void print()
{
cout<<x<<" "<<y<<"\n";
 }
};
void main()
{
Data D1 = {11,22};
Data D2 = D1;
D1.print();
 D2.print();
getch();
 }
```

Enumerated Data Types:

The enumerated data type is a programmer-defined type that is limited to a fixed list of values. A specifier gives the type a name and specifies the permissible values definitions then create variables of this type. Internally, the compiler treats enumerated variables as integers. Structures should not be confused with enumerated data type. Structures are a powerful and flexible way of grouping a diverse collection of data into a single entity.

General Form of EDT:

```
enum user_defined_name {
member_1;
member_2;
...
...
member_n;
};
```

Where **enum** is a keyword for defining the enumeration data type and the braces are essential. The members of the enumeration data type are the indivisual identifiers. Once the enumeration data type is defined, it can be declared in the following ways:

Storage_class enum user_defined_name var.1, var.2, …var.n

where the storage class is optional. For example:

1) Enum sample {

Mon, tue, wed, thu, fri, sat, sun };

Enum sample day1, day2, day3;

2) Enum drinks {

Cola, tea, koffi,rani }

Enum drinks d1, d2, d3;

3) Enum games {

Tennis, chess, football, swimming, walking };

Enum games student, staff;

The EDT declaration can be written in a single declaration as:

Enum sample { Mon, tue, wed, thu, fri, sat, sun }

Day1, day2, day3;

Which is exactly equivalent to:

1) Enum sample { Mon, tue, wed, thu, fri, sat, sun } day1;

Enum sample Day2, day3;

2) Enum sample { Mon, tue, wed, thu, fri, sat, sun };

Enum sample Day1;

Enum sample Day2;

Enum sample Day3;

The enumeration constants can be assigned to the variable like day1=mon;.

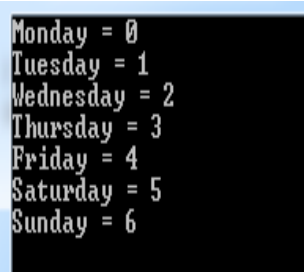
The enumeration constants are automatically assigned to integers starting from 0, 1, 2 etc. up to the last number in the enumeration.

Example 1:

Write C++ program to declare the EDT and to display the integer values on the screen.

```
#include <iostream>
#include <conio>
void main()
{
enum sample { mon, tue, wed, thu, fri, sat, sun }
day1, day2, day3, day4, day5, day6, day7;
day1=mon;
day2=tue;
day3=wed;
```

```
day4=thu;
day5=fri;
day6=sat;
day7=sun;
cout<<"Monday = "<<day1 <<endl;
cout<<"Tuesday = "<<day2 <<endl;
cout<<"Wednesday = "<<day3 <<endl;
cout<<"Thursday = "<<day4 <<endl;
cout<<"Friday = "<<day5 <<endl;
cout<<"Saturday = "<<day6 <<endl;
cout<<"Sunday = "<<day7 <<endl;
getch();
}
```



These integers are normally chosen automatically but they can also be specified by the programmer with negative or positive numbers, for example

Enum sample {mon, tue, wed=10, thu, fri, sat=-5, sun}

day1, day2, day3, day4, day5, day6, day7;

The C++ compiler assigns the enumeration constants as

Monday = 0

Tuesday = 1

Wednesday = 10

Thursday = 11

Friday = 12

Saturday = -5

Sunday =-4

Ex:-Write C++ program to declare the EDT and to display the difference between days.

```
#include <iostream>
#include <conio>
void main()
{
enum daysofweek { mon, tue, wed, thu, fri, sat, sun }
 day1, day2;
day1=mon;
day2=thu;
int diff=day2-day1;
cout<<"Days between="<<diff<<endl;
if (day1 < day2)
cout<<"day1 comes before day2 \n";
getch();
}
```