UNIT - V

STRUCTURES

Topics

- Declarations
- Nested Structures
- Array of structures
- Structure to function
- Unions
- Difference between Union and Structure

What is a Structure?

- Structure is a method of packing the data of different types.
- When we require using a collection of different data items of different data types in that situation we can use a structure.
- A structure is used as a method of handling a group of related data items of different data types.

A structure is a collection of variables under a single name. These variables can be of different types, and each has a name which is used to select it from the structure. A structure is a convenient way of grouping several pieces of related information together.

A structure can be defined as a new named type, thus extending the number of available types. It can use other structures, arrays or pointers as some of itsmembers, though this can get complicated unless you are careful.

Defining a Structure

A structure type is usually defined near to the start of a file using a typedef statement. typedef defines and names a new type, allowing its use throughout the program. typedefs usually occur just after the #define and #include statements in a file.

Here is an example structure definition.

```
typedef struct
 {
char name[64];
char course[128];
int age;
int year;
} student;
```

This defines a new type student variables of type student can be declared as follows.

student st_rec;

Notice how similar this is to declaring an int or float.

The variable name is st_rec, it has members called name, course, age and year.

Accessing Members of a Structure

Each member of a structure can be used just like a normal variable, but its name will be a bit longer. To return to the examples above, member name of structure st_rec will behave just like a normal array of char, however we refer to it by the name .

st_rec.name

Here the dot is an operator which selects a member from a structure. Where we have a pointer to a structure we could dereference the pointer and then use dot as a member selector. This method is a little clumsy to type. Since selecting a member from a structure pointer happens frequently, it has its own operator -> which acts as follows. Assume that st_ptr is a pointer to a structure of type student We would refer to the name member as.

st_ptr -> name

/* Example program for using a structure*/

```
#include< stdio.h >

void main()

{

int id_no;

char name[20];

char address[20];

char combination[3];

int age;

}newstudent;

printf(”Enter the student information”);
printf(”Now Enter the student id_no”);

scanf(“%d”,&newstudent.id_no);

printf(“Enter the name of the student”);

scanf(“%s”,&new student.name);

printf(“Enter the address of the student”);

scanf(“%s”,&new student.address);printf(“Enter the cmbination of the student”);

scanf(“%d”,&new student.combination);printf(Enter the age of the student”);

scanf(“%d”,&new student.age);

printf(“Student information\n”);

printf(“student id_number=%d\n”,newstudent.id_no);

printf(“student name=%s\n”,newstudent.name);

printf(“student Address=%s\n”,newstudent.address);

printf(“students combination=%s\n”,newstudent.combination);

printf(“Age of student=%d\n”,newstudent.age);

}
```

Arrays of structure:

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
{
int id_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.

```
#include< stdio.h >
{
struct info
{
int id_no;
char name[20];
char address[20];
char combination[3];
int age;
}
struct info std[100];
int I,n;
printf(“Enter the number of students”);
scanf(“%d”,&n);
printf(“ Enter Id_no,name address combination age\m”);
for(I=0;I < n;I++)
scanf(“%d%s%s%s%d”,&std[I].id_no,std[I].name,std[I].address,std[I].com
binat
ion,&std[I].age);
printf(“\n Student information”);
for (I=0;I< n;I++)
printf(“%d%s%s%s%d\n”,
”,std[I].id_no,std[I].name,std[I].address,std[I].combination,std[I].age);
}
```

Structure within a structure:

A structure may be defined as a member of another structure. In such structures the declaration of the embedded structure must appear before the declarations of other structures.

```
struct date
{
int day;
int month;
int year;

};

struct student

{

int id_no;

char name[20];

char address[20];

char combination[3];

int age;

structure date def;

structure date doa;

}oldstudent, newstudent;
```

the sturucture student constains another structure date as its one of its members.

UNIONS

Unions like structure contain members whose individual data types may differ from one another. However the members that compose a union all share the same storage area within the computers memory where as each member within a structure is assigned its own unique storage area. Thus unions are used to observe memory. They are useful for application involving multiple members.

Where values need not be assigned to all the members at any one time. Like structures union can be declared using the keyword union as follows:

```
union item
{
int m;
float p;
char c;
}
code;
```

this declares a variable code of type union item. The union contains three members each with a different data type. However we can use only one of them at a time. This is because if only one location is allocated for union variable irrespective of size. The compiler allocates a piece of storage that is large enough to access a union member we can use the same syntax that we use to access structure members. That is

```
code.m
code.p
code.c
```

are all valid member variables. During accessing we should make sure that we are accessing the member whose value is currently stored.

For example a statement such as -

```
code.m=456;
code.p=456.78;
printf("%d",code.m);
```

Would prodece erroneous result..

Enum declarations

There are two kinds of enum type declarations. One kind creates a named type, as in

enum MyEnumType { ALPHA, BETA, GAMMA };

If you give an enum type a name, you can use that type for variables, function arguments and return values, and so on:

```
enum MyEnumType x; /* legal in both C and C++ */
MyEnumType y; // legal only in C++
```

The other kind creates an unnamed type. This is used when you want names for constants but don't plan to use the type to declare variables, function arguments, etc. For example, you can write

```
enum { HOMER, MARGE, BART, LISA, MAGGIE };
```

Values of enum constants

If you don't specify values for enum constants, the values start at zero and increase by one with each move down the list. For example, given

enum MyEnumType { ALPHA, BETA, GAMMA };

ALPHA has a value of 0, BETA has a value of 1, and GAMMA has a value of 2.

If you want, you may provide explicit values for enum constants, as in

enum FooSize { SMALL = 10, MEDIUM = 100, LARGE = 1000 };

There is an implicit conversion from any enum type to int. Suppose this type exists:

enum MyEnumType { ALPHA, BETA, GAMMA };

Then the following lines are legal:

int i = BETA; // give i a value of 1

int j = 3 + GAMMA; // give j a value of 5

On the other hand, there is *not* an implicit conversion from int to an enum type:

MyEnumType x = 2; // should NOT be allowed by compiler

MyEnumType y = 123; // should NOT be allowed by compiler

Note that it doesn't matter whether the int matches one of the constants of the enum type; the type conversion is always illegal.

Typedefs

A typedef in C is a declaration. Its purpose is to create new types from existing types; whereas a variable declaration creates new memory locations. Since a typedef is a declaration, it can be intermingled with variable declarations, although common practice would be to state typedefs first, then variable declarations. A nice programming convention is to capitalize the first letter of a user-defined type to distinguish it from the built-in types, which all have lowercase names. Also, typedefs are usually global declarations.

Example:

Use a Typedef To Create A Synonym for a Type Name

```
typedef int Integer; //Integer can now be used in place of int
int a,b,c,d; //4 variables of type int
Integer e,f,g,h; //the same thing
```

In general, a typedef should never be used to assign a different name to a built-in type name; it just confuses the reader. Usually, a typedef associates a type name with a more complicated type specification, such as an array. A typedef should always be used in situations where the same type definition is used more than once for the same purpose. For example, a vector of 20 elements might represent different aspects of a scientific measurement.

Example:

Use a Typedef To Create A Synonym for an Array Type

```
typedef int Vector[20]; //20 integers
Vector a,b;
int a[20], b[20]; //the same thing, but a typedef is preferred
```

Typedefs for Enumerated Types

Every type has constants. For the "int" type, the constants are 1,2,3,4,5; for "char", 'a','b','c'. When a type has constants that have names, like the colors of the rainbow, that type is called an enumerated type. Use an enumerated type for computer representation of common objects that have names like Colors, Playing Cards, Animals, Birds, Fish etc. Enumerated type constants (since they are names) make a program easy to read and understand.

We know that all names in a computer usually are associated with a number. Thus, all of the names (RED, BLUE, GREEN) for an enumerated type are "encoded" with numbers. In eC, if you define an enumerated type, like Color, you cannot add it to an integer; it is not type compatible. In standard C++, anything goes. Also, in eC an enumerated type must always be declared in a typedef before use (in fact, all new types must be declared before use).

Example:

Use a Typedef To Create An Enumerated Type

```
typedef enum {RED, BLUE, GREEN} Color;

Color a,b;

a = RED;

a = RED+BLUE; //NOT ALLOWED in eC

if ((a == BLUE) || (a==b)) cout<<"great";
```

Notice that an enumerated type is a code that associates symbols and numbers. The char type can be thought of as an enumeration of character codes. The default code for an enumerated type assigns the first name to the value 0 (RED), second name 1 (BLUE), third 2 (GREEN) etc. The user can, however, override any, or all, of the default codes by specifying alternative values.

Text Books:

1.Alexis Leon and Mathews Leon (2001), Introduction to Information Technology, Tata McGraw-Hill.

2.R.G. Dromey (2001), How to Solve it by Computer, Prentice Hall of India.

3.Al Kelley and Ira Pohl (1998), A Book on C Programming in C, 4th Edition, Pearson Education.