Data Structures Using C

Notes

Subject: **Data Structure with C**

Topic: **Introduction to Data Structure**

Why Data Structures?

Problem solving is more relatd too understanding the problem, designing a solution and Implementing the solution, then What exactly is a solution?

In a very crisp way, it can be demonstarte as a solution which is equals to a program and it is also approxiamtley equals to algorithm.

Algorithm

An algorithm is a sequence of steps that take us from the input to the output. An algorithm must be Correct. It should provide a correct solution according to the specifications. Finite. It should terminate and general. It should work for every instance of a problem is efficient. It should use few resources (such as time or memory).

Data organization

Any algorithm we come up with will have to manipulate data in some way. The way we choose to organize our data directly affects the efficiency of our algorithm.

Solution = algorithm + data organization, Both components are strongly interconnected.

Information

The study of Computer Science includes study of information. Information in the substratum of entire field. In computer all information stored in form of a collection of bits, it is smallest unit of information, it has only one value

Data Type

It is representation of information using a set of value and set of operations required for deriving further results and consider an example A day’s rain is expressed in discrete form as millimeter.

This value can be subjected to a set of operation such as add to derive the total rain in a year, Division to derive average rain in a year. Unstructured or **scalar**: Integer, float, char and Pointer, **homogenous**: Array, string, enum, structure and Union, **heterogeneous** :ADT like list, queue, stack, tree and Graph.

Data structure

It is way of organizing value with help of existing data types, ex: Accumulation of rain data for one year and apply some operation to derive statistical results. Data of 365 days need integer to store 365 value in the list- one dimension and 10 different regions require to store – 2D. It is a aggregation of different type of data by which the stored data can be made more explanatory. Hence, the Data structure is require through knowledge of data types available in a Programming Language.

Data structure can be also defined as, it is the mathematical model which helps to store and retrieve the data efficiently from primary memory. It helps to consistently maintain the data as well as the implem-n functions of interest for data.

Data structure(definition by Prof. S Sahani)

It is data object together with the relationship which exist the relationship among the instance and among the individual elements that compose instance. Relationship provided by specifying the function of interest. When study data structure, we are concern with representation of data object as well as the implementation functions of interest for data object. Representation of each data object should facilitate an efficient implementation of function.

Atomic and composite data

Atomic data types is a single and non decomposable entity. Set of atomic data type having identical properties and consider an example the book price: Rs:250.

Composite data type, which cannot be broken out into subfield that having meaning consider an example Mobile number.

Data structure

It is an aggregation of atomic and composite data types into a set with defined relationships. Structure is set of rules that holds data together. An arrangement of data in a computer's memory. Algorithms manipulate the data in these structures in order to accomplish some task. Consider an example like inserting an item, search for an item, sorting. In other worlds , it is conceptual and concrete ways to organize data for efficient storage and manipulation.

Why do we need data structure ?

Computer takes on more and more complex tasks and its software implementation and maintenance is difficult, also clean conceptual frame work allows more efficient more correct code. Argument against: Packages are already written, Why not just read documentation of their interfaces and use them? The more you know, the better you can choose the tools, You can modify tools, You can create entirely new tools, You are to become experts !

We will learning the following concepts

What are some of the common data structures, What are some ways we can implement them , How can we analyze their efficiency , How can we use them to solve some, practical problems and Known data structures are tools for solving your future problems.

The following ways are possible to used the data structure. As an actual way to store real-world data, let we consider an example queues as a tool to be used only within a program, and graphs as a model of real-world situations.

Importance of different data structure

Each data structure has different advantages and disadvantages, and will be useful for different types of applications and for example of fast access of memory , if we know the

Data in Different volume

The amount of data and instructions are stored in different hierarchical memory is varies. The CPU can hold very small amount of data than cache memory and cache accommodate lesser than main memory, similarly main memory have less space than secondary memory. The data structure can comfortable access the data from main memory, if it is beyond the main memory , but the scope of data structure limited to main memory only. If at all needed to access the data from secondary memory then we need a concept called data mining which is explained in the figure 1.

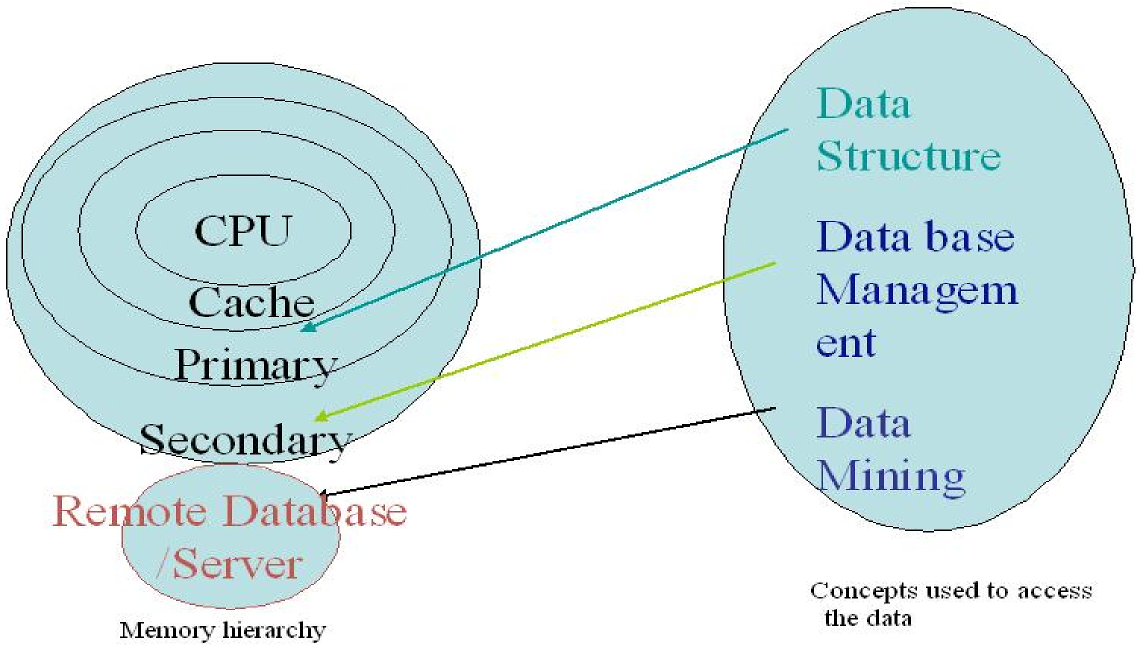

Figure 1: Concept used to access the data

UNIT- 1

Derived Data types

Structure:

1.1 Introduction

1.2 Type Definition – *typedef*

1.3 Enumerated types – *enum*

1.4 Structure

1.4.1 Accessing a structure

1.4.2 Pointer with structure

1.4.2.1 Pointer to structure

1.4.2.2 Pointer as structure member

1.4.3 Complex Structures

1.4.4. Arrays with structures

1.4.4.1 Array of structures

1.4.4.2 Array as a structure member

1.4.5 Structures and Function

1.5 Unions

1.1 Introduction

Before discussing the derived data types, let us understand first need for them. To do this, consider two complex numbers of the form **x+*i*y**, where x is real part and iy is the imaginary part. Any operation that has done on them in programming language requires four variables of floating point type. Since domain of problem is just about arithmetic operation on two complex number involving four variables can managed by a programmer. But problem domain complexity increases with the increase in number of complex numbers because the variables involved also increases, when realizing a complex number with the help of primitive data type like floating point.

Solution to the above problem is by use of structure which a derived data type. As a name indicates "derived" is built on basic data type of any language. Derived data types are also referred to as user-defined data types, this is because the user has flexibility to create his own type. Modern day application demands the use of derived type since the complexity of modern software is of industry strength software. We shall see the ease of using a derived data type in coming subsection of this report.

1.2 Type Definition – ***typedef***

Type definition *typedef,* gives a name to data type by creating a new type that can be used anywhere type is permitted. Advantage of using a *typedef* is that complex name can be replaced with easier ones which increases the readability of the program. The general format of *typedef* is shown.

```
typedef     type      IDENTIFIER;
```

Remember that *typedef* is standard keyword in C and that IDENTIFIER is traditional in upper case making it distinguishable. let us take an example.

```
typedef   int  INTEGER;
```

int is redefined as INTEGER.

1.3 **Enumerated types –** ***enum***

The enumerated type , *enum*, is built on the integer types. Enumerated types have enumeration constants which are assigned a integer value. This allows us to use symbolic names instead of values which makes the program more readable. By default enumerated constants starts with a value zero. The two common formats of enumeration in C is given below:

```
enum {enumeration constants} variable_identifier;

                    Or

enum tag {enumeration constants};

enum tag variable_identifier;
```

For example-

1) *enum* months {jan,feb,mar,apr,may,jun,jul,aug,sept,oct,nov,dec};
2) *enum* colors{Red,Black,White,Green,Blue};
   enum colors paint1,paint2;

In the above example, the list of constant identifiers are the symbolic names enclosed with the braces. So the default value assigned to the constant say Red is 0. But this can be changed

enum colors{Red=100,Black=200,White=300,Green=400,Blue=500};

Now the value set to Red is 100, Black is 200 and so on. Type definition is often used with enumerated types making it a powerful declaration consider the examples shown below

```
typedef enum {Red,Black,White,Green,Blue}COLORS;

COLORS paint1;
```

A program to illustrate enumeration for various dept. in institution is given below. It contents only the core logic.

```
void main()
{
        enum staff_dept
        { CSE, ISE, EEE, ENC, IEM,TE};

        struct staff
        {
            char name[25];
            int basic;
            enum staff_dept dept;
        }stf;

        stf  s;

        strcpy(s.name,”Koundinya”);
        s.basic =25000;
        s.dept = CSE;

        printf(“\n NAME=%s”,s.name);
        printf(“\n Salary =%s”,s.basic);
        printf(“\n Department=%s”,s.dept);
    }
```

1.4 Structure

Today's application requires complex data structures to support them. A structure is a collection of related elements where element belongs to a different type. Another way to look at a structure is a template – a pattern. For example graphical user interface found in a window requires structures typical example for the use of structures could be the file table which holds the key data like logical file name, location of the file on disc and so on. The file table in C is a type defined structure - FILE.

Each element of a structure is also called as field. A field has a many characteristic similar to that of a normal variable . An arrary and a structure are slightly different. The former has a collection of homogeneous elements but the latter has a collection of heterogeneous elements. The general format for a structure in C is shown

```
struct {field_list} variable_identifier;

 struct  struct_name
{
          type1  fieldname1;
          type2  fieldname2;
                   .
                   .
                   .
          typeN  fieldnameN;
};
struct struct_name variables;
```

The above format shown is not concrete and can vary, so different ' flavours of structure declaration is as shown.

```
struct
{
   ….
} variable_identifer;
```

```
struct tag
{
        …….
};
struct tag variable_identifers;
```

```
typedef struct

{

        ……..

} TYPE_IDENITIFIER;

TYPE_IDENTIFIER  variable_identifers;
```

Example

```
 struct mob_equip;
 {
          long int IMEI;
          char rel_date[10];
          char model[10];
          char brand[15];
 };
```

The above example can be upgraded with *typedef.* A program to illustrate the working of the structure is shown in the previous section.

```
typedef struct mob_equip;

  {
          long int IMEI;
          char rel_date[10];
          char model[10];
          char brand[15];
          int count;
 } MOB;  MOB m1;
```

1.4.1 Accessing a structure

A structure variable or a tag name of a structure can be used to access the members of a structure with the help of a special operator '.' –also called as member operator . In our previous example To access the idea of the IMEI of the mobile equipment in the structure mob_equip is done like this

m1.IMEI;

Since the structure variable can be treated as a normal variable All the IO functions for a normal variable holds good for the structure variable also with slight. The scanf statement to read the input to the IMEI is given below

scanf ("%d",&m1.IMEI);

Increment and decrement operation are same as the normal variables this includes postfix and prefix also. Member operator has more precedence than the increment or decrement. Say suppose in example quoted earlier we want count of student then

m1.count++; *++m1.count;*

1.4.2 Pointer with structure

The role of pointer with structure is criual as this concept is used further in implementation of the linked lists. There can be two cases with pointer with structures i.e., there can be pointer member in structure Or structure variable itself can be pointer which is nothing but pointer to structure. This further discussed in coming subsections and prerequisite required to learn the coming subsection is the basic of pointer concepts in C.

1.4.2.1 Pointer to structure

Structure, like other primitive types can also be accessed using a pointer. Let us take an example to better understand this

```
typedef struct
{
     float real;
     float  imag;
} COMPLEX;
```

The above declaration is a template for complex numbers with real and imaginary parts. This declaration is made use of in some other module or function like this

```
COMPLEX  c1;
COMPLEX  *ptr;
```

Now,

```
ptr = &c1;
```

*ptr refer to whole structure itself. Pointer contains the starting address of the structure. Member of the structures can be accessed like this

```
(*ptr).real;
```

Note that the parenthesis is absolutely necessary and omitting it will be a mistake. This is because precedence of the member operator is more than the indirection operator. The expression *(ptr.real) changes the meaning i.e., there is structure with ptr with member real which must be of pointer type.

1.4.2.2 Pointer as structure member

There is possibility that the pointer itself can be member of structure say suppose in previous example if imaginary is of pointer type i.e.,

```
typedef struct
{
     float real;
     float  *imag;
} COMPLEX;
```

Now COMPLEX c1;

Then the member imaginary can be accessed by

```
c1.*imag;
```

This means to say that the structure called c1 with its member imag which is of pointer type. To the same there can pointer to structure COMPLEX. Then placement of the indirection operator is bit confusing.

```
(*ptr).*imag;
```

The can made much clearer by the use of selection operator → like this

```
ptr→imag;
```

The above use of selection operator is less ambiguous and clear. Requirement of the subsections discussed now purely depends on the need of the problem requirement and the way its being used by a programmer. Hence based on this we decide upon the access strategies. This must be carefully implemented in program Or else may lead bugs in the program.

1.4.3 Complex Structures

As mentioned earlier structures helps us while dealing with the complex problems. We can have a structure as the member of a structure i.e., a structure includes another structure, i.e., an nested structure. There is no limit on the number of structures that can be nested but, this leads to a seldom if the number of nesting's exceeds three levels.

For example we are trying to implement the student information system for a particular institution that includes the details of the students like the name, USN, attendance in various subjects and the marks obtained in various subjects. Here if student is a structure then the attendance in various subjects along with the marks obtained forms a structure within student and can be shown below .

```
struct stud
{
        char name[20];
        char usn[10];
        struct attend
        {
                float subject[5];
                int marks[5];
        }a;
}std;
```

To access the inner structure attend we have to make use of the tag name given to the outer structure stud i.e,

```
std.a.subject[0];
```

1.4.4. Arrays with structures

The concept of arrays can be used with the structures also i.e, same as the pointers we end up in three cases i.e., an array of structures or array member in a structure and the third one is the combination of these .   The discussion is noe elaborated on these cases in the sub sections.

1.4.4.1   Array of structures

Now in the previous example of the student structure the std is a template for a single student in a class but in reality we have many students studying in a institute therefore this requires as many structure variables as the number of students, making it a cumbersome process of accessing the data so the solution to this problem is by using arrays of structures i.e.,

std student[50];

Accessing the details of a particular student say 12 is done by simply accessing it with the same subscript of the array i.e..,

student[12].name;

The initialization for the above array is the same as what we deal the normal arrays. Further an example will be given at the end of the section illustrating the use of all the subsections.