Chapter-1
Introduction to Data Structure

Content

1.1 Concept of Data Structure and algorithm

Programs are written in order to solve a given problem using a computer. A program consists of two components, algorithm and data structure.

Program = Algorithm + Data Structure

An **algorithm** can be defined as step-by-step procedure for solving a given problem in finite number of steps. Multiple algorithms can be designed to solve a particular problem. However, the algorithms that provide the maximum efficiency should be used for solving the problem i.e. algorithms should work in minimal time and use minimal memory. One of the technique for improving the efficiency of algorithm is to choose suitable data structure.

Data Structure is a way of organizing all data items (a single unit of value) and establishing relationship among those data items. Data structures are the building block of program. To develop a program of an algorithm, we should select an appropriate data structure for that algorithm. If a program is built using improper data structure, then the program may not work as expected. So it is very much important to use right data structure for a program to have efficiency and performance

Example:

1. Suppose we have to write a program that enables a printer to print the document from multiple users in first-come-first service basis then using a data structure **Queue** that stores and retrieves in the order of their arrival would be much more efficient than a data structure that store and retrieve the request in random order.

2. Suppose we have to write a program to find maximum value in a set of 50 numbers then instead of using 50 different variables, the use of data structure **array** of size 50 would be more efficient.

Some of the **examples of data structures** are array, stack, queue, list, tree, graph etc.

1.2 Classification of Data Structure

a. Static and Dynamic Data Structure

- **Static data structures** are those whose size is fixed at compile time and doesn't grow or shrink at run time. E.g. Array.
- **Dynamic data structures** are those whose size is not fixed at compile time and that can grow or shrink at run time so as to make efficient use of memory. E.g. Linked List
- **Arrays and linked list** are **basic data structures** that are used to implement other data structure such as stack, queue, trees and graphs. So any other data structure can be static or dynamic depending on whether they are implementing using and array or a linked list.

b. Primitive and Non-Primitive Data Structure

- **Primitive Data structure** / types are the data types provided by a programming language as basic building block. So primitive data types are predefined types of data, which are supported by programming language. For example, **integer, float and character** are all primitive data types.
- **Non-Primitive data types** are those data types which are not defined by programming language but are instead created by the programmer by using primitive data types. For example **Array, Linked List, stack, queue, graph and tree** are non-primitive data types.

c. Linear and Non-Linear Data Structure

- A data structure is said to be **linear** if its elements from a sequence i.e. linear list. So here while traversing the data elements sequentially, only one element can directly be reached. Example **Array, stack, queue** are example of linear data structure
- A data structure is said to be **non linear** if its elements do not form a sequence. So here every data item is attached to several other data items. **Trees and graphs** are the examples of non-linear data structure.

1.3 Data type and Abstract Data Type (ADT)

A **data type** is a classification of data, which can store specific type of information. Data types are primarily used in computer programming in which variables are created to store data. Each variable is assigned a data type that determine what type of data the variable may contain.

Data type is a collection of values and set of operation on those values.

Example: In the statement int a=5, 'a' is a variable of data type integer which stores integer value 5 and allow different mathematical operations like addition, subtraction, multiplication and division.

We can perform some specific operations with data structure, so data structure with these operations are called **Abstract Data Types (ADT).** ADT have their own data type and methods to manipulate data. It is a useful tool for specifying the logical properties of a data type. A data type or data structure is the implementation of ADT. So ADT allow us to work with abstract idea behind a data type or data structure without getting bogged down in the implementation detail. The abstraction in this case is called an abstract data type.

Advantage of abstract data typing

1. **Encapsulation:** Encapsulation is a process by which we can combine code and data it manipulates into a single unit. While working with ADT, the user doesn't require knowing how the implementation works because implementation is encapsulated in a simple interface.
2. **Flexibility:** If different implementations of ADT are equivalent then they can be interchangeable in the code. So user have flexibility to select the one which is most efficient in different situations.
3. **Localization of change:** If ADT is used then if changes are made to the implementation then it doesn't require any changes in the code.

Example:

1. Stack is ADT which have pop and push operation for deleting and inserting element
2. Rational number of form P/Q is ADT which have standard operations like Addition(), Multiplication(), Subtraction(), Division(), LessThan(), Equal().

1.4 Data Structure Operations

Most frequently used operations are

1. Traversing:- Accessing each record so that some item in the record may be processed.
2. Searching:- Finding the location of the record with the given key value.
3. Inserting:- Adding new record to the data structure.
4. Deleting:- Removing record from the data structure.
5. Sorting:- Arranging the record in some logical order.
6. Merging:- Combining the record of different sorted file into single sorted file.

Example:

An Array A having n element is ADT which have following operations.

Create(A):- Create an array A
Insert(A,x):- Insert an element x into an array A in any location
Delete(A,x):- Remove element x
Traverse(A):- Access each element of an array A
Modify(A,x,y);- Change the old element x with new element y
Search(A,x):- Search element x in an Array A
Merge (A,B,C):- Combine the elements of array B and C and store in new array A

1.5 Review of Array, Structures, Unions, Class, Pointer

Array

An **array** is a collection of elements of same type placed in contiguous memory location and can be accessed individually using index to a unique given name. It means array can contain one type of data only, either all integer, all characters or all floating points numbers. An array can be single dimensional or multi-dimensional.

Write a C program to add two array and display it.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        clrscr();
        int a[5]={1,2,3,4,5};
        int b[5]={2,4,6,8,10};
        int c[5];
        for( int i=0;i<=4;i++)
        {
                c[i]=a[i]+b[i];
        }

        for(int j=0;j<=4;j++)
        {
```

```
                printf("%d\t",c[j]);
        }
        getch();
}
```

Write a C program to perform addition of following two matrices.

$$A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} \quad B=\begin{bmatrix}4 & 5 & 6\\ 7 & 3 & 2\end{bmatrix}$$

```
#include<stdio.h>
#include<conio.h>
void main()
{
        clrscr();
        int a[2][3]={1,2,3,4,5,6};
        int b[2][3]={4,5,6,7,3,2};
        int c[2][3];
        for (int i=0;i<=1;i++)
        {
                for (int j=0;j<=2;j++)
                {
                        c[i][j]=a[i][j]+b[i][j];
                }
        }
        for(int k=0;k<=1;k++)
        {
                for(int l=0;l<=2;l++)
                {
                        printf("%d\t",c[k][l]);
                }
                printf("\n");
        }
        getch();
}
```

Structures

A structure is a collection of variables referenced under one name, providing a convenient means of keeping related information together. Structures are one of the ways to create custom data type in C. Variable that make up the structure are called elements or fields or members and can be of any data type.

WAP that define a structure to store the record of an employee with id, name, sex and salary fields. Input the record from user and display that record.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        struct record
        {
                int id;
                char name[20];
                char gender[5];
                float salary;
        }e[10];
        //struct record e;
        printf("Enter id: ");
        scanf("%d",&e.id);
        printf("Enter name:-");
```

```
        scanf("%s",&e.name);
        printf ("Enterd gender:-");
        scanf("%s",&e.gender);
        printf("Enter salary:-");
        scanf("%f",&e.salary);
        printf("Id is %d Name is %s Gender is %s  Salary is %f",e.id,e.name,e.gender,e.salary);
        getch();
}
```

WAP to read 10 students records with fields rollno, name, class and marks in 5 different subjects, and finally display their records along with their percentage of marks obtained.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    struct record
    {
        int roll;
        char name[10];
        int clss;
        int mark[5];
    };
    struct record e[10];
    clrscr();
    for(int i=0;i<=1;i++)
    {
        printf("Enter roll number: ");
        scanf("%d",&e[i].roll);
        printf("Enter name: ");
        scanf("%s",&e[i].name);
        printf("Enter class: ");
        scanf("%d",&e[i].clss);
        for(int j=0;j<=4;j++)
        {
            printf("Enter mark: ");
            scanf("%d",&e[i].mark[j]);
        }
    }
    for(int k=0;k<=1;k++)
    {
        printf("\nRoll no= %d",e[k].roll);
        printf("\nName= %s",e[k].name);
        printf("\nClass =%d",e[k].clss);
        printf("\nMarks in five different subject with percentage");
        int total=0;
        float pct=0;
        for(int l=0;l<=4;l++)
        {
            total=total+e[k].mark[l];
            pct=total/5;
            printf("\t%d",e[k].mark[l]);
        }
        printf("\n Percentage =%f",pct);
    }
    getch();
}
```

Union

A union is like structure except that a member of a union shares the same space in the memory. Unions are basically used to conserver memory. The size required by a union variable is the size required by the member requiring the largest size. The primary differences between structure and union is, for structure, the amount of memory required to store structure variables is sum of the size of all the members but in case of union the amount of memory required is equal to that required by largest member.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        union record
        {
                int roll;
                int clss;
                char name[10];
        };
        union record e;
        clrscr();
        printf("Enter roll: ");
        scanf("%d",&e.roll);
        printf("\nEnter class: ");
        scanf("%d",&e.clss);
        printf("Roll no= %d",e.roll);
        printf("\nClass= %d",e.clss);
        getch();
}
```

What is the output if user input is 4 for roll and 5 for class?

Roll no= 5
Class= 5

Class

A class is a way to bind the data and its associated function together. It allows the data and functions to be hidden, if necessary from external use. When defining a class, we are creating a new user defined data type called abstract data type that can be treated like any other built in data type. Class is a user defined data type which the user can use in implementing a stack ADT or Queue ADT etc. Once a class has been declared with different operations, then we can create variables of that type by using class name called as object. So by using that object we can easily work with the operations defined within the class at abstract level without knowing the implementation details. Hence class is an ADT. A simple example of class is given below

```
Class compare
{
        int num1, num2;
        public:
        void setdata(int a, int b)
        {
        }
        void findgreatest()
        {
        }
        void findsmallest()
        {
        }
}
void main()
{
compare a; // creating object a of type Compare
}
```

The class 'compare' consists of several operations like setdata(int a, int b), findgreatest(), findsmallest(). So to work with these operations we have to create variable or object of class type i.e. compare.

Pointer: See yourself!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!