

1 Introduction to Data Structure and Algorithms

1.1 Abstract Data Type (ADT)

Problem solving with a computer means processing data. To process data, we need to define the data type and the operations to be performed on the data. The definition of the data type and the definition of the operation to be applied to the data is part of the idea behind an abstract data type (ADT), i.e. to hide how the operation is performed on the data. In other words, the user of the ADT needs only to know that a set of operations are available for the data type, but does not need to know how they are applied.

Definition:

An abstract data type is a data type packaged with the operations that are meaningful for the data type. We then encapsulate the data and the operations on the data and hide them from the user.

A useful tool for specifying the logical properties of a data type is the ADT. Fundamentally, a data is a collection of values and a set of operations which can operate on those values and this form a mathematical construct that may be implemented. The ADT refers to the basic mathematical concept that defines the data type.

- A useful tool for specifying the logical properties of a data type is the abstract data type or ADT.
- Fundamentally, a data type is a collection of values and set of operation on those values.
- That collection and those operations form a mathematical construct that may be implemented using a particular hardware or software data structure.
- The term "Abstract Data type" refers to the basic mathematical concept that defines the data type.
- Formally, an abstract data type is a data declaration packaged together with the operations that are meaningful on the data type.
- In other words, we can encapsulate the data and the operation on data and we hide them from user.

Note:

Abstract Data Type

- **Model** of a Data Type
 - Properties of the data
 - Operation that can be performed on that data
- **Definition:** Abstract Data Type (ADT) is a **mathematical model** with a collection of operation defined on that model.



- **Example of ADT**
 - **Integer:**, -2, -1, 0, 1, 2,



- ADT for points in 2D plane


```

struct Point {
    int x, y;
};
double findDistance(struct Point a, struct Point b)
{
    double distance;
    distance = sqrt((a.x - b.x) * (a.x - b.x) + (a.y-b.y) *(a.y-b.y));
    return distance;
}
      
```


1.2 Introduction to Data Structures

- Data structure is a way of organizing all data items and establishing relationship among those data items.
- Data structures are the building blocks of a program.
- Data structure mainly specifies the following four things:
 - Organization of data.
 - Accessing methods
 - Degree of associativity
 - Processing alternatives for information

To develop a program of an algorithm, we should select an appropriate data structure for that algorithm. Therefore algorithm and its associated data structures form a program.

Algorithm + Data structure = Program

A **static data structure** is one whose capacity is fixed at creation. For example, array. A **dynamic data structure** is one whose capacity is variable, so it can expand or contract at any time. For example, linked list, binary tree etc.

- Data are simply value or set of values. A data item refers to a single unit of values.
- Data structure is the structural representation of logical relationships between elements of data. In other words a data structure is a way of organizing data items by considering its relationship to each other.
- Data structure can also be defined as the logical or mathematical model of a particular organization of data is called data structure.
- Data structure mainly specifies the structured organization of data, by providing accessing methods with correct degree of associativity. Data structure affects the design of both the structural and functional aspects of a program.
- Data structures are the building blocks of a program. Hence proper selection of data structure increases the productivity of programmers due to the proper designing and the use of efficient algorithms.
- If the problem is analyzed and divided into sub problems, the task will be much easier i.e., divide, conquer and combine.
- A complex problem usually cannot be divided and programmed by set of modules unless its solution is structured or organized.
- This is because when we divide the big problems into sub problems, different programmers or group of programmers will program these sub problems.
- By choosing a particular structure (or data structure) for the data items, certain data items become friends while others loses its relations.
- The representation of a particular data structure in the memory of a computer is called a storage structure. That is, a data structure should be represented in such a way that it utilizes maximum efficiency.
- The data structure can be represented in both main and auxiliary memory of the computer.
- A storage structure representation in auxiliary memory is often called a file structure.
- The data structure and the operation on organized data items can integrally solve the problem using computer

Data structure = Organized data + Operations

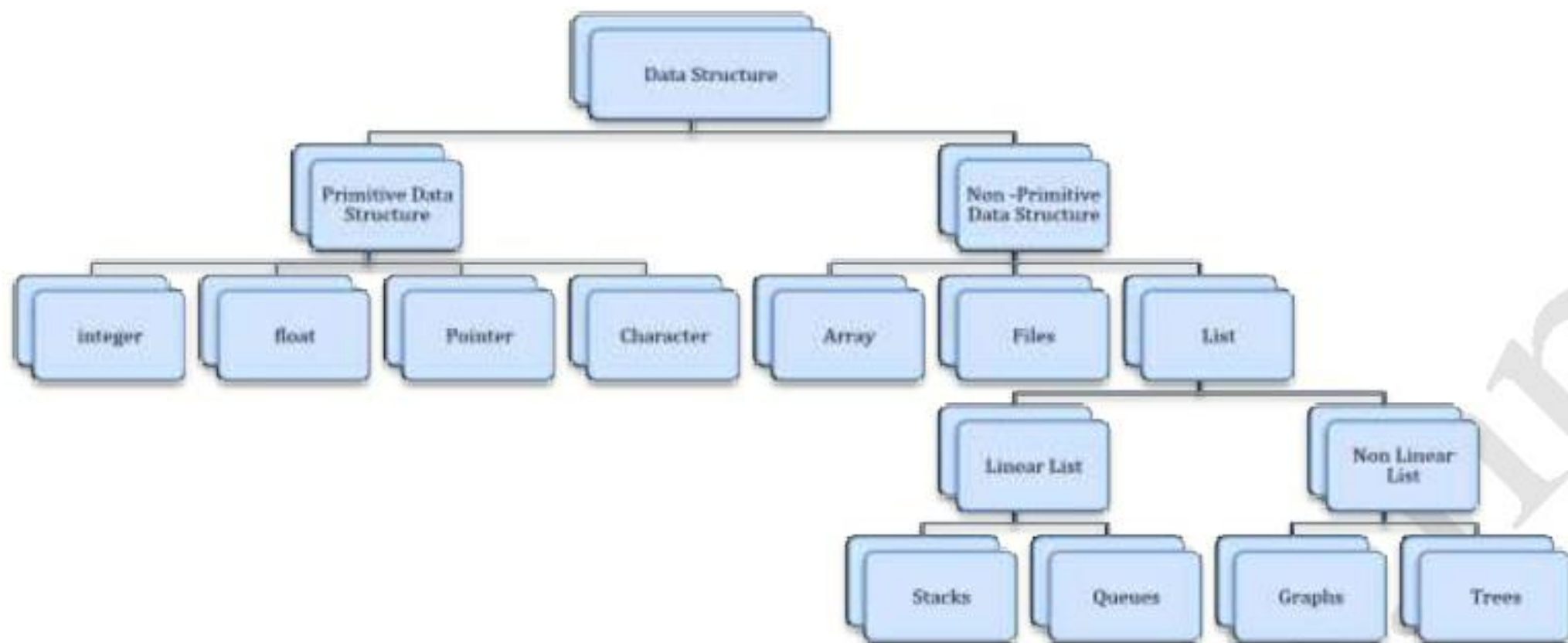
- In short Data Structure is an agreement about:
 - How to store a collection of objects in memory,
 - What operations we can perform on that data,
 - The algorithms for those operations, and
 - How time and space efficient those algorithms are

Data Structure Application Example

- How does Google quickly find web pages that contain a search term?
- What's the fastest way to broadcast a message to a network of computers?
- How can a subsequence of DNA be quickly found within the genome?
- How does your operating system track which memory (disk or RAM) is free?

1.3 Classification of Data Structure

Data structures are normally divided into two broad categories



1.3.1 Primitive Data Structures

- These are the basic data structures and are directly operated upon by the machine instructions, which is in a primitive level. They are integers, floating point numbers, characters, string constants, pointers etc.

1.3.2 Non-primitive Data Structures

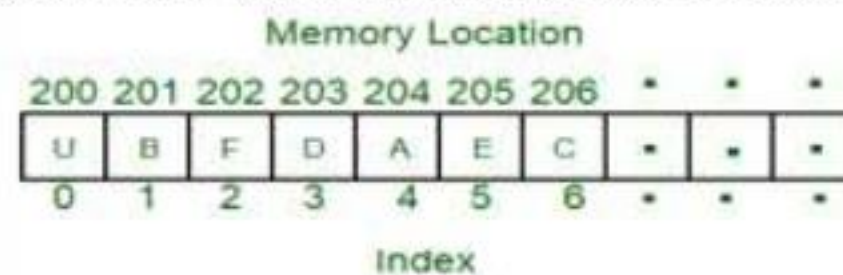
- It is a more sophisticated data structure emphasizing on structuring of a group of homogeneous (same type) or heterogeneous (different type) data items. Array, list, files are examples.

Note:

- ✓ The most commonly used operations on data structures are broadly categorized into four types
 - CREATE
 - SELECTION
 - DESTROY or DELETE
 - UPDATION
- ✓ Other operations performed on data structures include
 - Searching
 - Sorting
 - Merging

1.4 Array

- It is the collection of related data items, stored in a continuous way.
- The very common linear structure is array. Since arrays are usually easy to traverse, search and sort, they are frequently used to store relatively permanent collections of data.
- An array is a list of a finite number n of homogeneous data elements (i.e., data elements of the same type) such that:
 - The elements of the array are referenced respectively by an index consisting of n consecutive numbers.
 - The elements of the array are stored respectively in successive memory locations.
- **An array is a collection of memory locations which allows storing homogeneous elements.**



- The first element of the array is referred with subscript 0(zero) & the subscript of last element is one less than size of an array i.e. $(n-1)$.
- The elements of an array A may be denoted by the subscript notation (or bracket notation),
 $A[0], A[1], A[2], \dots, A[N-1]$
- The number K in $A[K]$ is called a subscript or an index and $A[K]$ is called a subscripted variable.
- An array can be single dimensional or multidimensional. The number of subscript give the dimension of an array.

An array lets you declare and work with a collection of values of the same type (homogeneous). For example, you might want to create a collection of five integers. One way to do it would be to declare five integers directly: `int a, b, c, d, e;` Suppose you need to find average of 100 numbers. What will you do? You have to declare 100 variables.

For example: `int a, b, c, d, e, f, g, h, i, j, k, l, m, n... etc.,`

An easier way is to declare an array of 100 integers: `int a[100];`

The General Syntax is:

`datatype array_name [size];`

Example: `int a[10];`

Subscript

Some common operations performed in an array are (Show that an array is an ADT:

- Creating of an array
- Inserting new element at required position
- Deletion of any element
- Modification of any element
- Traversing of an array
- Merging of arrays

Insertion of new element at required position

Suppose we have an array

`A[7]={ 10,50,8,7,30,75,85};`

Suppose we want to insert 15 in array A , at location with index 4, it means the elements 30, 75 and 85 must shift 1 position downwards.

Deletion of any element from an array:

Suppose we want to delete the element $A[4]=30$, then the elements following it were moved upward by one location.

Traversing of an array:

Traversing means to access all the elements of the array, starting from first element upto the last element in the array one-by-one

Merging of two arrays:

Merging means combining elements of two arrays to form a new array. Simplest way of merging two arrays is the first copy all elements of one array into a third empty array, and then copy all the elements of other array into third array.

Suppose we want to merge two arrays $A[7]$ and $B[5]$. The new array says C will be having $(7+5) = 12$ elements.

1.5 Structure

A structure is a collection of one or more variables, possibly of different types, grouped together under a single name.

An array is a data structure in which all the members are of the same data type. Structure is another data structure in which the individual elements can differ in type. Thus, a single structure might contain integer elements, floating-point elements and character elements. The individual structure elements are referred to as members. Defining a structure: A structure is defined as

```
struct structure_name
{
    member 1;
    member 2;
    .....
    member n;
};
```

Structure variable declaration:

```
struct structure_name s1, s2, s3;
```

We can combine both template declaration and structure variable declaration in one statement.

Eg,

```
struct Student
{
    char name[2];
    int roll;
    char sec;
    float marks;
} s1, s2, s3;
```

Accessing members of a structure:

There are two types of operators to access members of a structure. Which are:

- Member operator (dot operator or period operator (.))
- Structure pointer operator (->).

1.6 Unions:

Both structure and unions are used to group a number of different variables together. Syntactically both structure and unions are exactly same. The main difference between them is in storage. In structures, each member has its own memory location but all members of union use the same memory location which is equal to the greatest member's size.

Declaration of union:

The general syntax for declaring a union is:

```
union union_name
{
    data_type member1;
    data_type member2;
    data_type member3;
    .....
    data_type memberN;
};
```

union variable declaration:

```
union Student s1, s2, s3;
```


we can combine both template declaration and union variable declaration in one statement.

Eg,

```
union Student
{
    char name[2];
    int roll;
    char sec;
    float marks;
} s1, s2, s3;
```

Differences between structure and unions

The amount of memory required to store a structure variable is the sum of sizes of all the members. On the other hand, in case of a union, the amount of memory required is the same as member that occupies largest memory.

1.7 Pointer

A pointer is a variable that holds address (memory location) of another variable rather than actual value. Also, a pointer is a variable that points to or references a memory location in which data is stored. Each memory cell in the computer has an address that can be used to access that location. So, a pointer variable points to a memory location and we can access and change the contents of this memory location via the pointer. Pointers are used frequently in C, as they have a number of useful applications. In particular, pointers provide a way to return multiple data items from a function via function arguments.

Pointer Declaration

Pointer variables, like all other variables, must be declared before they may be used in a C program. We use asterisk (*) to do so. Its general form is:

data-type *ptrvar;

For example,

```
int* ptr;
float *q;
char *r;
```

Reasons for using pointer:

- A pointer enables us to access a variable that is defined outside the function.
- Pointers are used in dynamic memory allocation.
- They are used to pass array to functions.
- They produce compact, efficient and powerful code with high execution speed.
- The pointers are more efficient in handling the data table.
- They use array of pointers in character strings result in saving of data storage space in memory. Sorting strings using pointer is very efficient.
- With the help of pointer, variables can be swapped without physically moving them.
- Pointers are closely associated with arrays and therefore provides an alternate way to access individual array elements.

Pointer initialization:

Once a pointer variable has been declared, it can be made to point to a variable using an assignment statement as follows:

```
int marks;

int *marks_pointer;

marks_pointer=&marks;
```


✚ Memory Allocation in C

Static memory allocations or Compile time

In static or compile time memory allocations, the required memory is allocated to the variables at the beginning of the program. Here the memory to be allocated is fixed and is determined by the compiler at the compile time itself.

Dynamic memory allocations or Run time

In most of the real time problems, we cannot predict the memory requirements. Dynamic memory allocation does the job at run time

The process of allocating and freeing memory at run time is known as Dynamic Memory Allocation. This reserves the memory required by the program and returns this resource to the system once the use of reserved space utilized.

There are 4 library functions in C, **malloc()**, **calloc()**, **free()** and **realloc()** for memory management. These functions are defined within header file **stdlib.h** and **alloc.h**.

✚ Passing (call) by Value and Passing (call) by Reference

Arguments can generally be passed to functions in one of the two ways:

- Sending the values of the arguments (pass by value)
- Sending the addresses of the arguments (pass by reference)

Pass by value: In this method, the value of each of the actual arguments in the calling function is copied into corresponding formal arguments of the called function. With this method the changes made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function.

Pass by reference: In this method, the addresses of actual arguments in the calling function are copied into formal arguments of the called function. This means that using these addresses we would have an access to the actual arguments and hence we would be able to manipulate them.

✚ Pointers and Arrays

An array name by itself is an address, or pointer. A pointer variable can take different addresses as values. In contrast, an array name is an address, or pointer, that is fixed.

Pointers and One-dimensional Arrays

In case of one dimensional array, an array name is really a pointer to the first element in the array. Therefore, if x is a one-dimensional array, then the address of the first array element can be expressed as either $\&x[0]$ or simply x . Moreover, the address of the second array element can be expressed as either $\&x[1]$ or as $(x+1)$, and so on. In general, the address of array element $(x+i)$ can be expressed as either $\&x[i]$ or as $(x+i)$. Thus we have two different ways to write the address of any array element: we can write the actual array element, preceded by an ampersand; or we can write an expression in which the subscript is added to the array name. Since, $\&x[i]$ and $(x+i)$ both represent the address of the i th element of x , it would seem reasonable that $x[i]$ and $*(x+i)$ both represent the contents of that address, i.e., the value of the i th element of x . The two terms are interchangeable. Hence, either term can be used in any particular situation. The choice depends upon your individual preferences.

1.8 Class

- In C++ programming it is possible to separate program specific data types through the use of classes.
- Classes define types of data structures and the functions that operate on those data structures.
- A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package.
- Instances of these data types are known as objects and can contain member variables, constants, member functions, and operators defined by the programmer.
- Syntactically, classes are extensions of the C struct, which cannot contain functions or overloaded operators.
- A class is a way to bind the data and its associated functions together. It allows the data (and function) to be hidden, if necessary, from external use.
- When defining a class we are creating a new abstract data type that can be treated like any other built-in data type.
- The data inside the class are called member data and the functions are called member function.
- The binding of data and functions together into a single class type variable is called encapsulation, which is one of the benefit of object-oriented programming.
- The general form of declaring class is:


```

class class_name
{
    access-specifier1: member_data1;
                        member_data2;
                        -----
                        -----
                        member_function1;
                        -----
                        -----
    access-specifier2: member_data1;
                        -----
                        member_function1;
    access-specifier: member_data;
                        -----
                        member_function;
};

```

In above declaration, **class** is keyword. `class_name` is any identifier name. The number of member data and member function depends on the requirements. An object is an instance of a class i.e. variable of a class. The general form of declaring an object is

```
class_name object_name;
```

OR

```

class class_name
{
    private:
        variable_declaration;
        function_declaration;
    public:
        variable_declaration;
        function_declaration;
};

```

The **class** declaration is similar to a **struct** declaration. The keyword **class** specifies that what follows is an abstract data of type `class_name`. The body of a class is enclosed within braces and terminated by a semicolon. The class body contains the declaration of variables and functions. These functions and variables are collectively called class members. They are usually grouped under section, namely **private** and **public** to denote which of the members are private and which of them are public. The keyword **private** and **public** are known as visibility labels. Note that these keywords are followed by a colon.

🚩 Differentiate ADT with C++ class

A class is basically a block of code that contains anything your program needs to run whereas an ADT is an abstract way to store data.

- Classes have a slightly different terminology than ADT's and add other characteristics, like:
 - for example may live in packages
 - their members are called attributes and methods
 - attributes and methods have a certain visibility constraint
- An ADT is an abstract data structure with associated operations. A class is a programming construct associated with object oriented programming. Often a class will implement an ADT, but they are conceptually different beasts.
- In other words you don't need an Object Oriented language to do Object Oriented programming. It's convenient to work in a language that supports classes; but it's not necessary. It's not the language that makes a design Object Oriented. It's the use of ADTs.

So ADTs are the concept; and Classes are one (but not the only) implementation.

Example: A Stack ADT defines the basic stack operations like push and pop (but says nothing of how these operations should be implemented), while a Stack class would use either a linked-list or an array to actually implement these operations.