UNIT - I: Introduction to Data Structures & Algorithms [5 hrs.]

Data types, Data structure and Abstract date type, Dynamic memory allocation in C, Introduction to Algorithms, Asymptotic notations and common functions

**Data structure**

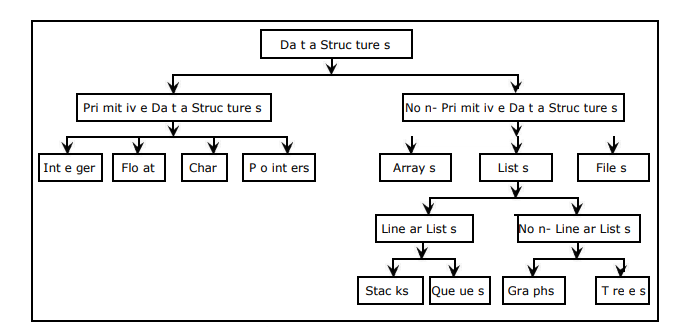
* A data structure defines a way of organizing all data items that considers not only the elements stored but also their relationship to each other.
* The term data structure is used to describe *the way data is stored*.
* To develop a program of an algorithm we should select an appropriate data structure for that algorithm. Therefore, data structure is represented as:

**Algorithm + Data structure = Program**

**Objectives of data structure:**

* It enables efficient data processing.
* Efficient and organized data so it becomes easy to access.
* It helps in data protection and management.
* It helps to easily identify relationship among data.

Types of data structure:



Data structures are divided into two types:

• Primitive data structures. (Built in data structure)//internally provide by HLL,

• Non-primitive data structures.(User -defined data structure)//formed by user according to their need.

1. **Primitive Data Structures (Built in data structure)** are the basic data structures that directly operate upon the machine instructions. They have different representations on different computers. Integers, floating point numbers, character constants, string constants and pointers come under this category. Such as integer, float and character datatypes.
2. **Non-primitive data structures** are more complicated data structures and are derived from *primitive data structures*. They emphasize on grouping same or different data items with relationship between each data item. Arrays, lists and files come under this category. Eg. stack , queue, list ,tree and graph.

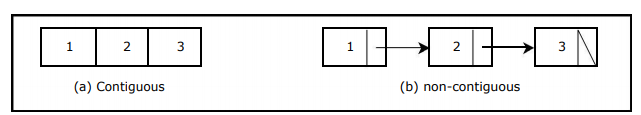
It is of following types:

1. Linear data structure// data stored in sequential manner
2. Non Linear data structure
3. **A linear data structure** has data elements connected to each other so that elements are arranged in a *sequential manner* and each element is connected to the element in front of it and behind it. This way, the structure can be traversed in a single run.

Linear data structures can be implemented easily as computer memory is also arranged in a linear manner. There are four types of linear data structures:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11 | 12 | 13 | 14 | 15 |  |  |  |  |

* Array
* Linked list
* Stack
* Queue



1. Contiguous: are made of single slabs of memory, some of these data structures are arrays, matrices, heaps, and hash tables.
2. Non-contiguous: are composed as distinct chunks of memory linked together by pointers (references). Some of this data structures are lists, trees, and graph adjacency lists.

[i] Array: An array is a collection of homogenous data type stored in a continuous memory location.

In c/c++ indexing start from 0,1,2,3,….,n-1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| 11 | 13 | 45 | 67 | 20 |

int a[5]=

The size of arry is (n-1)

Eg;

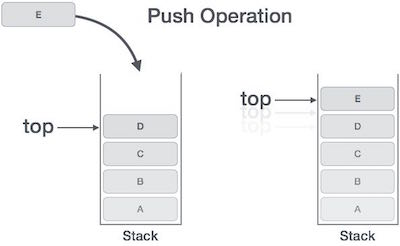
a[0],a[1],s[2],s[3],a[4]

**Stack**

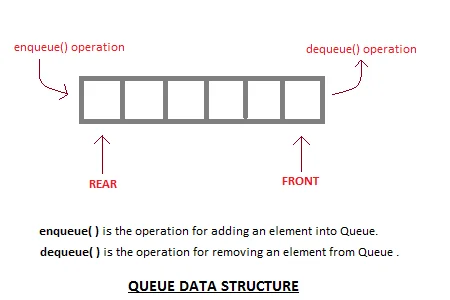
* A stack is a linear structure in which items may be added or removed only at
* one end.
* Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] |
| 11 | 12 | 13 | 14 |  |  |  |  |

A[8]

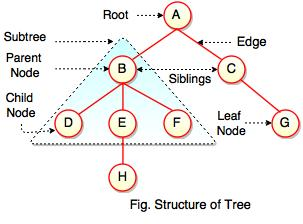
* Consider an example of plates stacked over one another in the canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time.

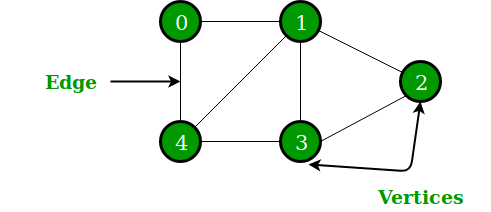
**Queue**

* A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO).
* A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first.
* The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

1. **Non-linear data structure:**

* A data structure is said to be non-linear if its elements do not form a sequence but it represents hierarchical relationship between its elements.
* They are : Linked list, Tree and Graph





**Array:**

Types of array

1. Single dimension

int a[5]={11,12,13,14,15}

eg:

void main()

{

int a[5]={11,12,13,14,15};

int i;

clrscr();

for(i=0;i<=4;i++)

{

printf(“%d\n”,a[i]);

}

getch();

}

1. Double dimension

The array which is used to represent and store data in a tabular form is called as two dimensional array. Such type of array specially used to represent data in a matrix from.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | [0][0] | [0][1] | [0][2] |
| 1 | [1][0] | [1][1] | [1][2] |

int a[3][3];

int a[3][3]={{1,2,3},{4,5,6},{7,8,9}};

#include<stdio.h>

#include<conio.h>

void main()

{

int a[3][3];

int r,c;

clrscr();

puts("Enter value in 3 x 3 matrix");

for(r=0;r<3;r++)

{

for(c=0;c<3;c++)

{

scanf("%d",&a[r][c]);

}

}

puts("Value in 3 x 3 matrix");

for(r=0;r<3;r++)

{

for(c=0;c<3;c++)

{

printf("%4d",a[r][c]);

}

printf("\n");

}

getch();

}

1. Multi dimension

Initialization in Three-Dimensional array is same as that of Two-dimensional arrays. The difference is as the number of dimension increases so the number of nested braces will also increase.

Syntax:

data\_ type array\_name [size1][size2]….[sizeN];

Method 1:

int x[2][3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,

11, 12, 13, 14, 15, 16, 17, 18, 19,

20, 21, 22, 23};

Better Method:

int x[2][3][4] =

{

{ {0,1,2,3}, {4,5,6,7}, {8,9,10,11} },

{ {12,13,14,15}, {16,17,18,19}, {20,21,22,23} }

};

#include<iostream.h>

int main()

{

int x[2][3][2] =

{

{ {0,1}, {2,3}, {4,5} },

{ {6,7}, {8,9}, {10,11} }

};

// output each element's value

for (int i = 0; i < 2; ++i)

{

for (int j = 0; j < 3; ++j)

{

for (int k = 0; k < 2; ++k)

{

cout << "Element at x[" << i << "][" << j<< "][" << k << "] = " << x[i][j][k]<< endl;

}

}

}

return 0;

}

Output:

Element at x[0][0][0] = 0

Element at x[0][0][1] = 1

Element at x[0][1][0] = 2

Element at x[0][1][1] = 3

Element at x[0][2][0] = 4

Element at x[0][2][1] = 5

Element at x[1][0][0] = 6

Element at x[1][0][1] = 7

Element at x[1][1][0] = 8

Element at x[1][1][1] = 9

Element at x[1][2][0] = 10

Element at x[1][2][1] = 11

1. Sort array in ascending.
2. Find the data in array.
3. Insert an element in an array.
4. Delete an element in an array.

**Representation of two dimensional array in memory**

* The two dimensional array is also stored in liner from while storage.
* This can be done in two ways:

**Row-Major order:**

* When the elements of the array are stored row-by-row it is called row major order. It means that, first store the first row of the array, then the second row of the array, then the third row and so on.
* Eg. The two elements A(0,1) & A(1,2) of the array A[1…3, 1…4] are show below:

Row 3

Row 2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | (0,1) |  |  |  |  | (1,2) |  |  |  |  | (2,3) |

Row 1

**Column-Major order:-**

* When the elements of the array are stored column-by-column it is called column major order. It means that first column of the array, then the second column of the array and so on.

Col3

Col2

col1

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | (0,1) |  |  |  | (1,2) |  |  |  | (2,3) |

Col4

**Address calculation for two dimensional Array**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** |  | **j** |  | **n** |
| **1** |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **i** |  |  |  |  | **A(i,j)** |  |  |
|  |  |  |  |  |  |  |  |
| **m** |  |  |  |  |  |  |  |

* Consider an array of dimensions m\*n (there are 1 to n rows and 1 to m columns) & the base address BA. w is the number of locations occupied by each element

**Row-major order: -**

There are(i-1(lb)) full rows. They have(i-1(lb))\*n elements.

The partial row i has (j-1(lb)) elements

Locations occupied by (i-1(lb)) full rows=(i-1(lb)\*n\*w

Location occupied by (j-1(lb)) element=(j-1(lb)\*w

Total number of location up to the element(i,j)

= (i-1(lb) \* n\*w+(j-1(lb))\*w

= w[n(i-1(lb) +(j-1(lb))]

Address of the element a[i,j] =BA+w[n(i-1(lb)]) +(j-1(lb)])]

*Assume that each element of an array are stored in row-major order occupies four units of storage. If a is deleared as by each of the following and the address of the first element of a is 100, find the address of the indicated array element:*

1. *Int a[10][20] address of a[2][1]*
2. *Int a[10][20] address of a[5][10]*

*A[5][10]=100+4\*[ 20\*(5-0)+(10-0)]*

*=100+4\*(100+10)*

*=100+440*

*=540*

Address of the element a[i,j] =BA+w[n(i-1(lb)]) +(j-1(lb)])]

*Answer: In c lower bound of an array is always zero(0).*

*A[2][1]= 100+4\* [ 20\*(2-0)+(1-0)]*

*=100 +4\*(20\*2+1)*

*=100+4\*41*

*=100+164*

*=264*

**Column -major order**

**Row-major order: -**

There are(j-lb) full columns each with m elements.

There are(i-1b) elements in the jth column.

Locations occupied by (j-lb) full columns=(j-1b)\*m\*w

Location occupied by(i-1b) element=(i-1b)\*w

Total number of location up to the element(i,j)

= (j-1b) \* m\*w+(i-1b)\*w

= w[(j-1b)m +(i-1b)]

Address of the element a[i,j] =BA+w[(i-1b) +m(j-1b)]

Assume that each element of an array are stored in column-major order occupies four units of storage. If a is deleared as by each of the following and the address of the first element of a is 100, find the address of the indicated array element:

i) Int a[10][20] address of a[2][1]

ii) Int a[10][20] address of a[5][10]

a[i,j] =BA+w[(i-1b) +m(j-1b)]

= 100 +4[ (2-0) + 10(1-0) ]

= 100+4[2+10]

= 100 +4\*12

= 100 +48

=148

ii) Int a[10][20] address of a[5][10]

a[i,j] =BA+w[(i-1b) +m(j-1b)]

=100+4[(5-0)+10(10-0)]

=100 +4[5+100]

= 100+4\*105

= 100+ 420

=520

Data structure operation:

There are six basic operation that can be performed in data structure:

1. Traversing: accessing and processing each element exactly one
2. Searching: Finding elements or location
3. Sorting: arranging in sequential order
4. Inserting
5. Deleting
6. Merging.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 1 | 2 |
| 0 | 11 | 12 | 13 |
| 1 | 14 | 15 | 16 |