Data Structure

Report

By

Md. Abir Hossain

**ID:** 171442522

**Batch:** 44th

# wE have learned before our mid exam

* 1D Array
* 2D Array
* Stack
* Queue &
* Linked List.

# now we will discuss about these topics in briefly

* **1D & 2D Array**

An array is a collection of data that holds fixed number of values of same type. For example, if we want to store marks of 100 students, we can create an array for it.

float marks[100];

here, marks is the array name and float is its value type and we have declared array’s size to 100.

There are two types of array. One is 1D and another is 2D or multidimensional array.

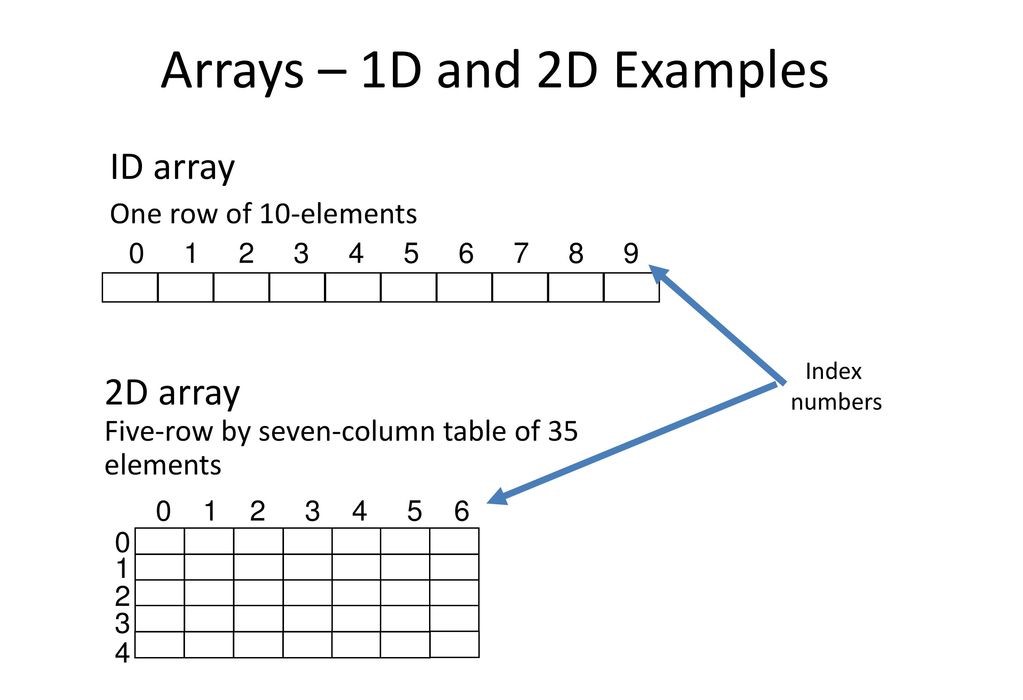
Array declaration for 1D & 2D will be,

**int** mark[5]; // 1D array declaration

**int** mark[2][3]; // 2D array declaration

in 2D array there’s will be row and column number.

Graphical examples of 1D and 2D are showing below,



Simple programs of 1D and 2D array,

**1D array:**

#include<stdio.h>

int main()

{

int a[10];

int i, n;

printf("How many element you want to save \n");

scanf("%d", &n);

printf("Enter element one by one \n");

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

{

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

}

printf("The List you entered\n");

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

{

printf("%d ", a[i]);

}

return 1;

}

**Output will be:**

How many element you want to save

3

Enter element one by one

10

20

30

The List you entered

10 20 30

**2D array:**

#include<stdio.h>

void main(){

int a[10][10];

int i,j,rows, columns;

printf("How many rows you want \n");

scanf("%d", &rows);

printf("How many columns you want \n");

scanf("%d", &columns);

printf("Enter your array Element one by one \n");

for(i=0;i<rows;i++){

for(j=0;j<columns;j++){

scanf("%d", &a[i][j]);

}

}

printf("Your array \n");

for(i=0;i<rows;i++){

for(j=0;j<columns;j++){

printf("%d\t ", a[i][j]);

}

printf("\n");

}

}

**Output will be:**

How many rows you want

2

How many columns you want

2

Enter your array Element one by one

2

2

2

2

Your array

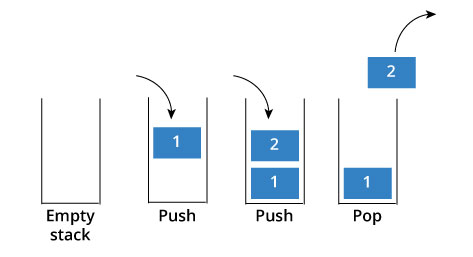
2 2

2 2

* **Stack**

A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack.

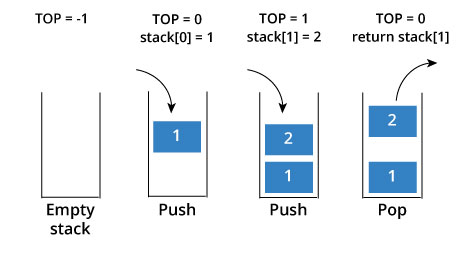
Basic concept of stack is showing graphically below,



**How stack works:**

The operations work as follows:

1. A pointer called TOP is used to keep track of the top element in the stack.
2. When initializing the stack, we set its value to -1 so that we can check if the stack is empty by comparing TOP == -1.
3. On pushing an element, we increase the value of TOP and place the new element in the position pointed to by TOP.
4. On popping an element, we return the element pointed to by TOP and reduce its value.
5. Before pushing, we check if stack is already full
6. Before popping, we check if stack is already empty.



**Stack Implementation in C language:**

#include <stdio.h>

int stack[20]; //Stack declaration

int head = -1; //Stack initially empty

void push(int data){

head++;

stack[head] = data;

}

int pop(){

int data = stack[head];

head--;

return data;

}

void printstack(){

printf("Data in your stack\n");

int i;

for(i=0;i<=head;i++){

printf("%d ",stack[i]);

}

}

void main()

{

push(5);

push(7);

push(10);

printstack();

int data = pop();

printf("\nYour pop data: %d \n",data);

printstack();

}

**Output will be:**

Data in your stack

5 7 10

Your pop data: 10

Data in your stack

5 7

**Use of Stack:**

Although stack is a simple data structure to implement, it is very powerful. The most common uses of a stack are:

* **To reverse a word** - Put all the letters in a stack and pop them out. Because of LIFO order of stack, you will get the letters in reverse order.
* **In compilers** - Compilers use stack to calculate the value of expressions like 2+4/5\*(7-9) by converting the expression to prefix or postfix form.
* **In browsers** - The back button in a browser saves all the URL’s you have visited previously in a stack. Each time you visit a new page, it is added on top of the stack. When you press the back button, the current URL is removed from the stack and the previous URL is accessed.
* **Queue:**

Queue arranges element in such a way that new element always gets added in the back and also when comes the removal it removes the last or oldest element or which is the first element in the queue. It is also known as FIFO or First In First Out.

Basic of Queue is showing graphically below,



**How Queue Works:**

Queue operations work as follows:

1. Two pointers called FRONT and REAR are used to keep track of the first and last elements in the queue.
2. When initializing the queue, we set the value of FRONT and REAR to -1.
3. On enqueing an element, we increase the value of REAR index and place the new element in the position pointed to by REAR.
4. On dequeueing an element, we return the value pointed to by FRONT and increase the FRONT index.
5. Before enqueing, we check if queue is already full.
6. Before dequeuing, we check if queue is already empty.
7. When enqueing the first element, we set the value of FRONT to 0.
8. When dequeing the last element, we reset the values of FRONT and REAR to -1.



**Queue implementation in C language:**

#include<stdio.h>

#define SIZE 100 //Queue size declaration

int queue[SIZE], head = -1, tail = 0; //Empty Queue declaration

void enQueue(int value){

if(head == SIZE-1)

printf("\nQueue is Full!!!");

else{

head++;

queue[head] = value;

}

}

void deQueue(){

if(head == -1)

printf("\nQueue is Empty!!! ");

else{

printf("\nDeleted: %d", queue[tail]);

tail++;

}

}

void display(){

if(head == -1)

printf("\nQueue is Empty!!!");

else{

int i;

printf("\nQueue elements are:\n");

for(i=tail; i<=head; i++)

printf("%d\t",queue[i]);

}

}

void main()

{

enQueue(10);

enQueue(20);

enQueue(30);

enQueue(40);

display();

deQueue();

display();

deQueue();

display();

}

**Output will be:**

Queue elements are:

10 20 30 40

Deleted: 10

Queue elements are:

20 30 40

Deleted: 20

Queue elements are:

30 40

* **Linked List**

A linked list is represented by a pointer to the first node of the linked list. The first node is called head. If the linked list is empty, then value of head is NULL. In C, we can represent a node using structures. Linked list consists many node structures those are point each other and create a link.

**struct** node //Node Structure

{

int data;

struct node \*next;

};

**How to traverse a linked list**

Displaying the contents of a linked list is very simple. We keep moving the temp node to the next one and display its contents.

When temp is NULL, we know that we have reached the end of linked list so we get out of the while loop.

struct node \*temp = head;

printf("\n\nList elements are - \n");

while(temp != NULL)

{

printf("%d --->",temp->data);

temp = temp->next;

}

The output of this program will be:

List elements are -

1 --->2 --->3 --->

**How to add elements to linked list**

We can add elements to either beginning, middle or end of linked list. Here we will see only the beginning & end operations.

**Add to beginning**

* Allocate memory for new node
* Store data
* Change next of new node to point to head
* Change head to point to recently created node

For an example,

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = 4;

newNode->next = head;

head = newNode;

**Add to end**

* Allocate memory for new node
* Store data
* Traverse to last node
* Change next of last node to recently created node

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = 4;

newNode->next = NULL;

struct node \*temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = newNode;

**How to delete from a linked list**

You can delete either from beginning, end or from a particular position.

**Delete from beginning**

* Point head to the second node

head = head->next;

**Delete from end**

* Traverse to second last element
* Change its next pointer to null

struct node\* temp = head;

while(temp->next->next!=NULL){

temp = temp->next;

}

temp->next = NULL;

**Complete program for linked list operations is showing below,**

#include <stdio.h>

struct Node

{

int data;

struct Node \*next;

};

struct Node \*head = NULL;

void insertAtBeginning(int value)

{

struct Node \*newNode;

newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = value;

newNode->next = NULL;

if(head == NULL)

{

head = newNode;

}

else

{

newNode->next = head;

head = newNode;

}

}

void display()

{

if(head == NULL)

{

printf("\nList is Empty\n");

}

else

{

struct Node \*temp = head;

printf("\n\nList elements are - \n");

while(temp->next != NULL)

{

printf("%d ",temp->data);

temp = temp->next;

}

printf("%d ",temp->data);

}

}

void insertAtEnd(int value)

{

struct Node \*newNode;

newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = value;

newNode->next = NULL;

if(head == NULL)

head = newNode;

else

{

struct Node \*temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = newNode;

}

}

void removeBeginning()

{

if(head == NULL)

printf("\n\nList is Empty!!!");

else

{

struct Node \*temp = head;

if(temp->next == NULL)

{

head = NULL;

free(temp);

}

else

{

head = temp->next;

free(temp);

}

}

}

void removeEnd()

{

if(head == NULL)

{

printf("\nList is Empty!!!\n");

}

else

{

struct Node \*temp1 = head,\*temp2;

if(temp1->next == NULL)

{

head = NULL;

}

else

{

while(temp1->next != NULL)

{

temp2 = temp1;

temp1 = temp1->next;

}

temp2->next = NULL;

}

free(temp1);

}

}

int main()

{

insertAtEnd(10);// 10

insertAtEnd(20); // 10 20

insertAtBeginning(40); // 40 10 20

insertAtBeginning(50); // 50 40 10 20

insertAtEnd(100); // 50 40 10 20 100

display();

removeBeginning(); // 40 10 20 100

display();

removeEnd(); // 40 10 20

display();

return 0;

}

**Output will be:**

List elements are -

50 40 10 20 100

List elements are -

40 10 20 100

List elements are -

40 10 20