COURSE TITLE:DATA STRUCTURE

Course Code :CSE 213

MIDTERM

10/10/2018

COURSE TITLE:DATA STRUCTURE

Course Code:CSE 213

Midterm

NAME: Parizat Binta Kabir

ID : 171442508

1. Data Structure

A data structure is a collection of data and the relationships among them, and the operations that can be applied to the data.

Section:

- Array
- Stack
- Queue
- Linked list

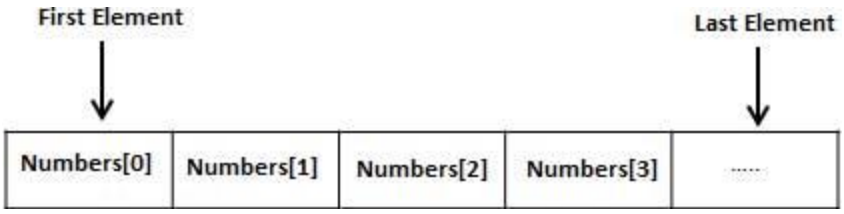
Our course Data Strucre includes above all this topic in the mid term.

2.Array

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



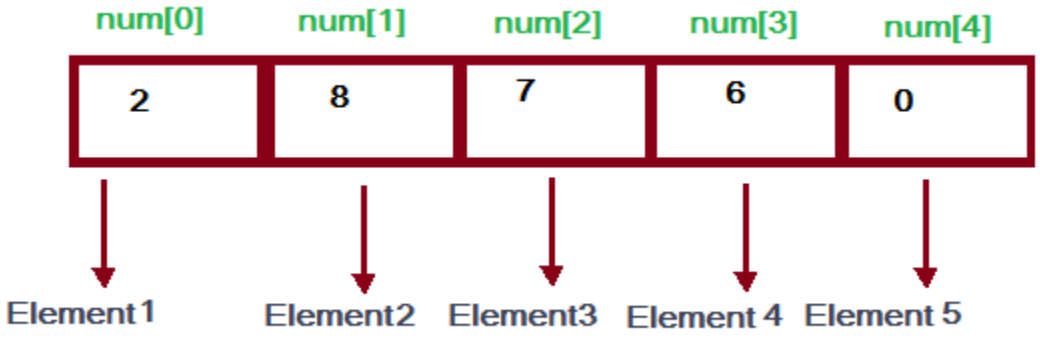
2.1 One Dimensional Array

A one-dimensional array (or single dimension array) is a type of linear array. Accessing its elements involves a single subscript which can either represent a row or column index.

As an example consider the C declaration `int anArrayName[10];`

Syntax : datatype anArrayname[sizeofArray];

```
int   a[n];
for(i=1 to n)
scanf("%d" , & a[i]);
```



One Dimensional Array Operation

- Insert,update,delete,search.
- Travarsal/display/print.

2.2 Two Dimensional Array

The two dimensional array in C language is represented in the form of rows and columns, also known as matrix. It is also known as array of arrays or list of arrays

Like 1D arrays, we can access individual cells in a 2D array by using subscripting expressions giving the indexes, only now we have two indexes for a cell: its row index and its column index. The expressions look like:

```
    a[i][j] = 0;
or
    x = a[row][col];
```

We can initialize all elements of an array to 0 like:

```
    for(i = 0; i < MAX_ROWS; i++)
        for(j = 0; j < MAX_COLS; j++)
            a[i][j] = 0;
```

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Two Dimensional Array Operation

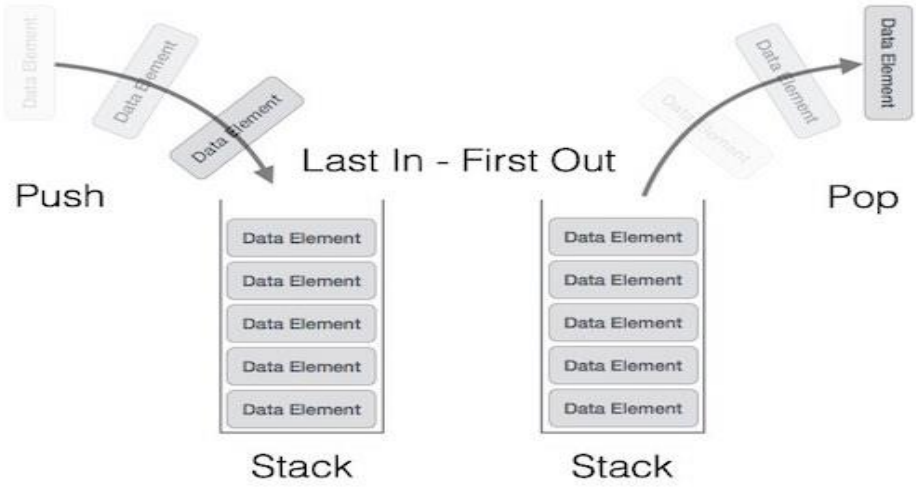
- Insert,update,delete,search.
- Traversal/Display/Print.

3.Stack

A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle.It is named stack as it behaves like a real world stack.It's an ADT(Abstract Data Type).

Stack Operations

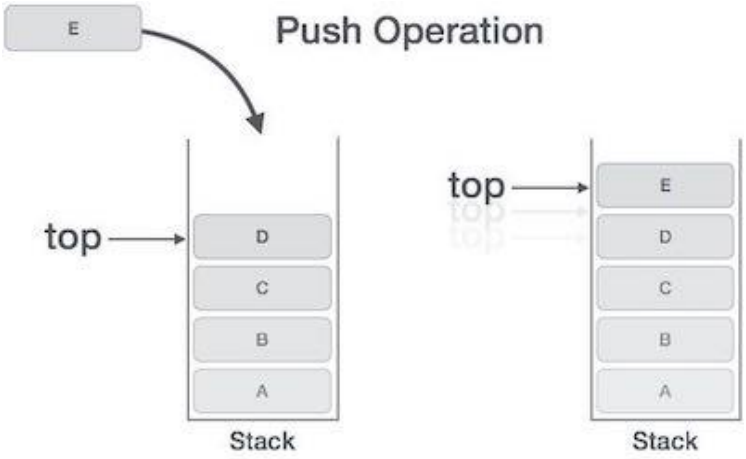
- Push () pushing (storing) an element on the stack.
- POP() removing (accessing)an element from the stack.
- isFull () check if the stack is full.
- isEmpty () check if the stack is empty.



Stack Implementation Using Array

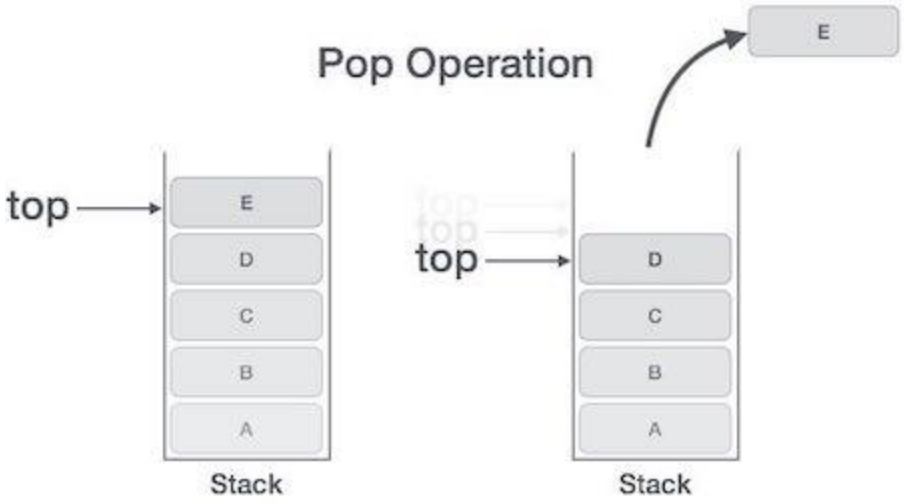
```
#define  size  10;
int  stack[size];
int top =-1;

isEmpty(){
return (top == -1)? true  :  false  ;
}


isFull(){
return (top == size)? true  :  false  ;
}
```

Stack Push Using Array




```
void push(int data){
top ++ ;
stack [top]  = data;
}
```

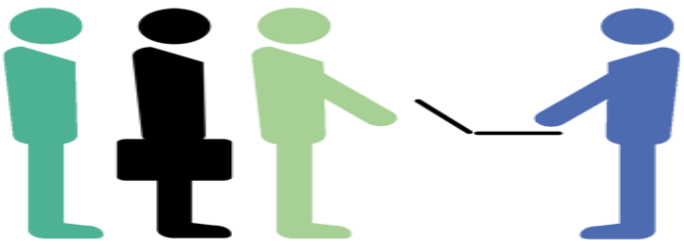
Stack POP Using Array




```
int pop(){
int data;
data = stack[top];
top --;
return data;
}
```

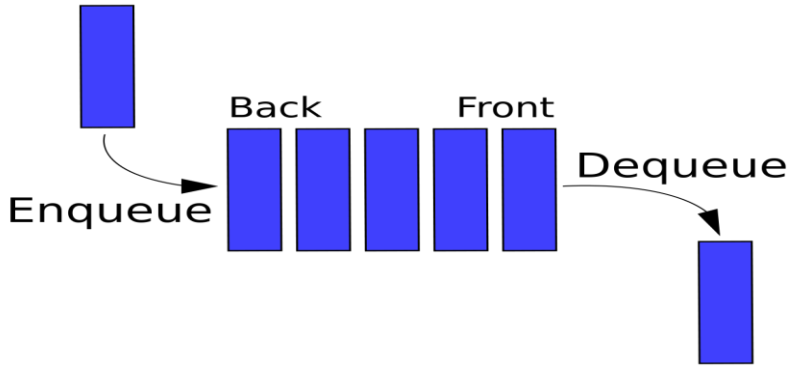
4. Queue

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodol0gy, i.e., the data item stored first will be accessed first.



Queue Operations

- enqueue() add (store) an element to the queue.
- dequeue () remove (access) an item from the queue.
- isfull() checks if the queue is full.
- isempty() checks if the queue is empty.



Queue Using Array

```
#define  size 10

int queue [size];

int head = -1;

isEmpty(){

        return (head == tail)? true  :  false  ;

        }

isFull(){

        return (head == size-1)? true  :  false  ;
        }
```

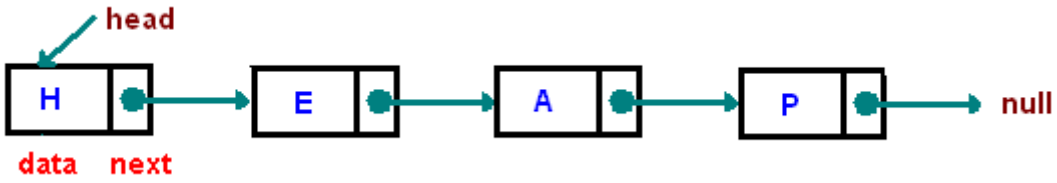
Queue Using Array

```
void enqueue(int data){

head++;

stack[head]=data;
```

```
}

void dequeue(){

int data;

head++;

data = stack[tail] ;

tail++ ;

return data;

}
```

5.Linked List

A linked list is a linear data structure where each element is a separate object.



Each element (we will call it a **node**) of a list is comprising of two items - the data and a reference to the next node. The last node has a reference to null. The entry point into a linked list is called the **head** of the list. It should be noted that head is not a separate node, but the reference to the first node. If the list is empty then the head is a null reference.

There are three types of linked list.Such as:

- Single Linked List
- Double Linked List
- Circular Linked List

5.1 Single Linked List

Single linked list is the list where every element has linked to its next element in the sequence.

```
struct node{

int data;

struct node*next;   } ;
```

Single Linked List Operation

- Insertion (Beginning and end of the list).
- Deletion(Beginning and end from the list).
- Display.

Insert Beginning Of The List

```
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;
  }

}
```

Remove Beginning From The List

```
void removeBeginning()
{
   if(head == NULL)
   printf("\n\nList is Empty!!!");
   else
   {
      struct Node *temp = head;
      if(head->next == NULL)
      {
       head = NULL;
       free(temp);
      }
      else
      {
      head = temp->next;
      free(temp);
      }
   }
}
```

Display or Traversal of The List

```
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);

   }

}
```

5.2 Stack Using Linked List

I. **Push() :** Insert the element into linked list at the beginning and increase the size of the list. O(1) operation.

II. **Pop() :** Return the element first node from the linked list and move the head pointer to the second node. Decrease the size of the list. O(1) operation.

III. **getSize():** Return the size of linked list.

IV. **displayStack():** Print the linked list.

V. When head pointer is Null stack is empty.

5.3 Queue Using Liked List

- Queue using linked list is basically the same of insert node at the end of the list and remove node from the beginning of the list.
- Insert is same as enqueue function and delete is same as the dequeue function.
- When head pointer is NULL the the queue is empty.

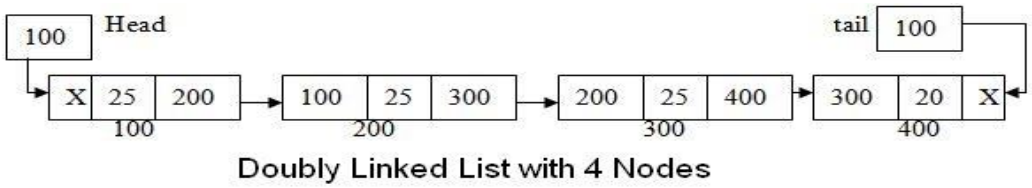
5.4 Double Linked List

Double linked list is the list where every element has linked to its previous element and next element in the sequence.

```
struct node{

struct node*prev ;

int data ;

struct node*next } ;
```

Double Linked List Operation

- Insertion(Beginning and end of the list)
- Deletion(Beginning and end from the list)
- Display/Traversal
- In double linked list we can traverse both directions.



Doubly Linked List with 4 Nodes

Insert Beginning In Double Linked List

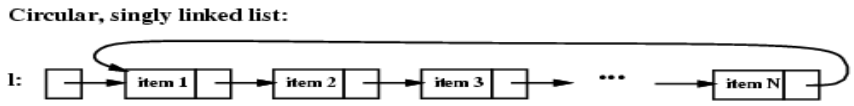
```
void insertAtBeginning(int value)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  newNode->previous = NULL;
  if(head == NULL)
  {
    newNode->next = NULL ;
    head = newNode;
  }
  else
  {
    newNode->next = head;
    head = newNode;
  }

Printf("Insertion success");

}
```

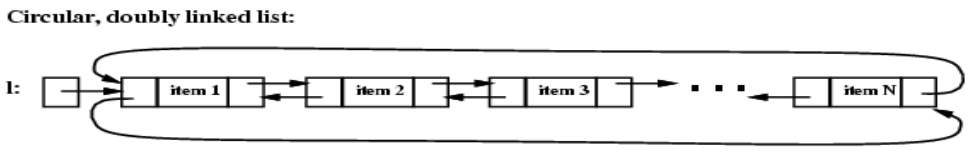
Delete Beginning From Double Linked List

```
void removeBeginning()
{
  if(head == NULL)
  printf("\n\nList is Empty!!!");
  else
  {
    struct Node *temp = head;
    if(temp -> previous == temp -> next)
    {
     head = NULL;
     free(temp);
    }
    else
    {
    head = temp->next;
    head -> previous = NULL ;
    free(temp);
    }
```

5.5 Circular Linked List

Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.





6.References

[1] programming9, https://www.programming9.com/

[2]tutorials point, https://www.tutorialspoint.com/java

[3]Data Structure, https://www.studytonight.com/data-structures/