Course Title: Data Structure

Course Code: CSE 213

Supta Richard Philip

September 20, 2018

1 Data Structure

Data Structure

Data Structure is a way to store and organize data so that it can be used efficiently. All topics of Data Structure

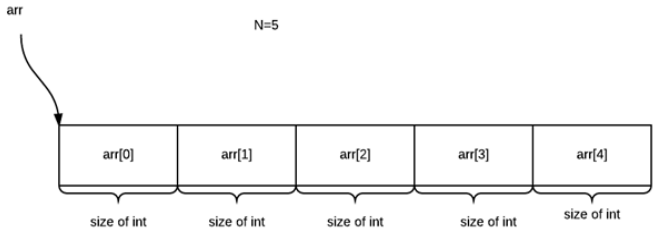
- Array, Pointer, Structure
- Stack, Queue, Linked List
- Tree and Graph
- Searching, Sorting

Our Data Structure course includes all topics of Data Structure such as Array, Pointer, Structure, Linked List, Stack, Queue, Graph, Searching, Sorting, etc.

2 Array

Array

Array is a data structure that contains a group of elements. Typically these elements are all of the same data type, such as an integer or string. When declare array that allocate a consecutive group of memory locations having same name and the same type. To refer to a particular location using positive index into the array.

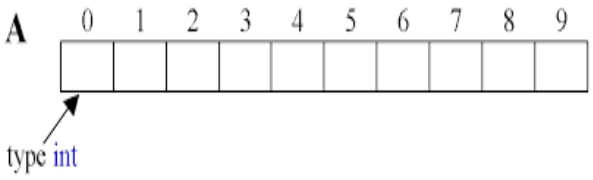


2.1 One Dimensional Array

One Dimensional Array

An array is a collection of similar type of data items and each data item is called an element of the array. The data type of the element may be any valid data type like char, int, float or double.

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



One Dimensional Array operations

- Insert, Update, Delete, Search
- Traversal/Display/Print

Write a C program using One dimensional Array and perform above operations. Complexity: $\mathcal{O}(n)$,

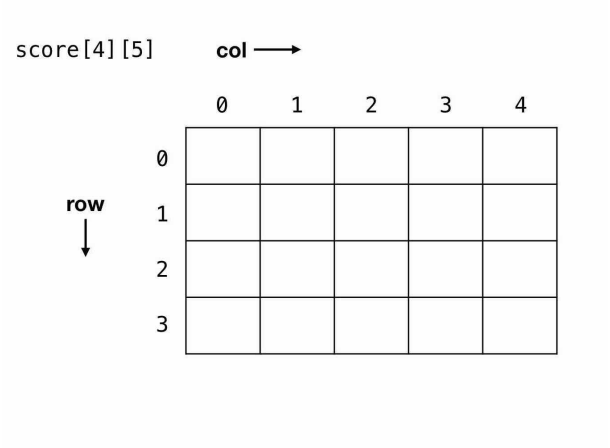
$$f(x) = \prod_{i=1}^{n} x_i$$

$$f(x) = \sum_{i=1}^{n} x_i$$

2.2 Two Dimensional Array

Two Dimensional Array

The 2D array is organized as matrices which can be represented as the collection of rows and columns. int A[rows][columns];



Two Dimensional Array operations

- Insert, Update, Delete, Search
- Traversal/Display/Print

Complexity: $\mathcal{O}(n^2)$,

$$f(x) = \prod_{i=1,j=1}^{n} x_{ij}$$

$$f(x) = \sum_{i=1,j=1}^{n} x_{ij} \; if \; i = j \; (diagonal \; operation)$$

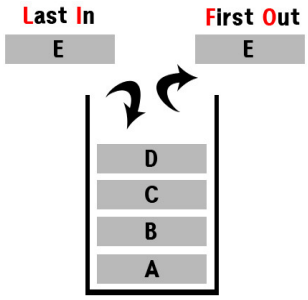
3 Stack

Stack

A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack.

Stack Operations

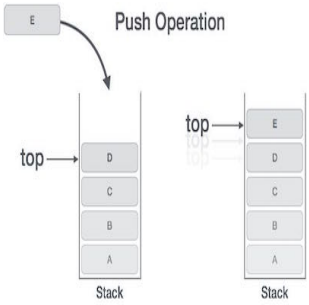
- push() Pushing (storing) an element on the stack.
- pop() Removing (accessing) an element from the stack.
- isFull() check if stack is full.
- isEmpty() check if stack is empty.



Stack implementation using array

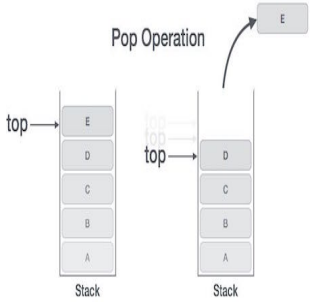
```
#define SIZE 10
int stack[SIZE];
int top = -1;

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

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

Stack push using array




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

Stack pop using array

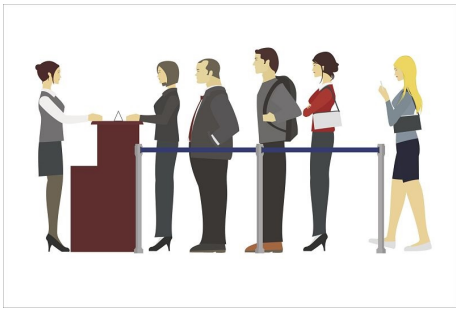



```
int pop() {
int data;
data = stack[top];
top--;
return data;
}
```

4 Queue

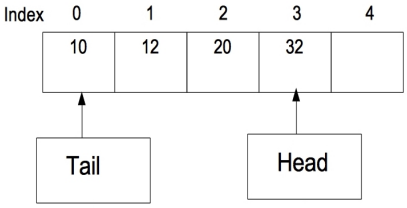
Queue

Queue is an abstract data structure, One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology.



Queue Operations

- enqueue() add (store) an item 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;
int tail = 0;

bool isEmpty() {
      return (head == tail) ? true : false;
}

bool isFull() {
      return (head==SIZE-1)? true : false;
}
```

Queue Using Array

```
void enqueue(int data) {
      head++;
      stack[head] = data;
}

int dequeue() {
int data;
data = stack[tail];
tail++;
return data;
}
```

5 Pointer and Structure

5.1 Pointer

Pointer

A pointer is a variable whose hold the address of another variable. The asterisk * used to declare a pointer.

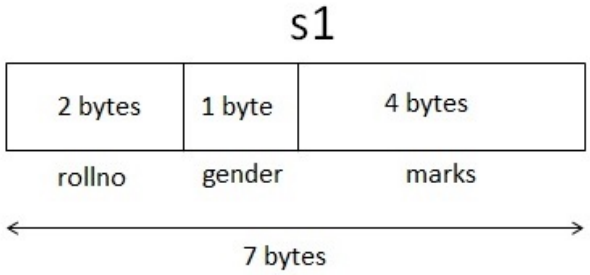
```
int    *ip;    /* pointer to an integer */
int  *ptr = NULL; /* Assign null to pointer */
```

5.2 Structure

Structure

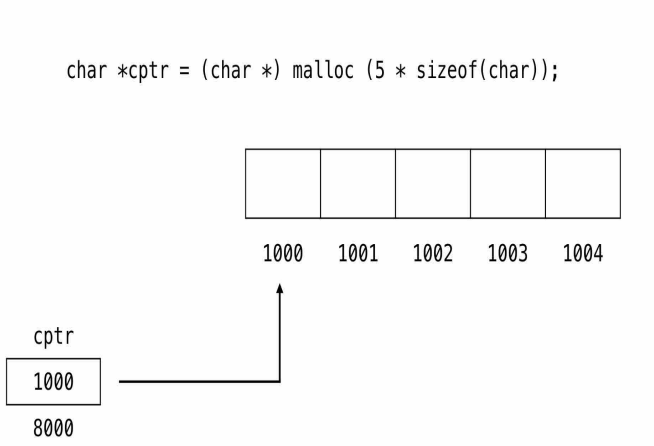
Structures are used to represent a record.

```
struct student{
      int rollno;
      char gender;
      float marks;
}s1;
```



5.3 Dynamic Memory Allocation

Dynamic Memory Allocation



6 Linked List

Linked List

Simply a list is a sequence of data, and linked list is a sequence of data linked with each other. There are three kind of linked list.

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

6.1 Single linked list

Single Linked List

Single linked list is a sequence of elements in which every element has link to its next element in the sequence.

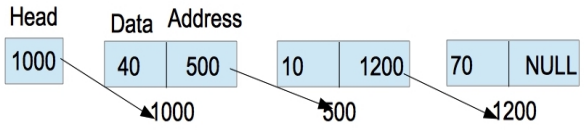
```
struct Node
{
    int data;
    struct Node *next;
};
```

Note: Insert end of the list and delete from end of the list, please check github repo.

Single Linked List operations

In a single linked list we perform the following operations.

- 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);
    }
}
```

6.2 Stack using Linked List

Stack using single Linked List

- Stack using linked list is basically simple logic of insert node at beginning of the list and remove node from the beginning of the list or insert node at end of the list and remove node from the end of the List.
- insert is same as push function and delete/remove is same as pop functions.
- when head pointer is NULL then the Stack is empty.

6.3 Queue using single Linked List

Queue using single Linked List

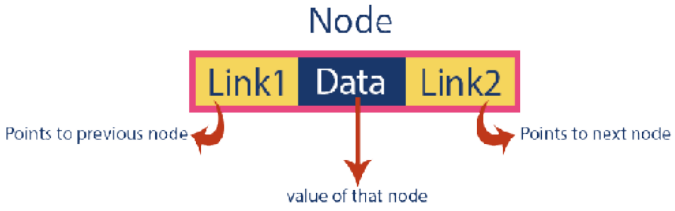
- Queue using linked list is basically the same logic of insert node at end of the list and remove node from the beginning of the List or vise verse.
- Insert is same as enQueue function and delete/remove is same as deQueue function.
- when head pointer is NULL then the Queue is empty.

6.4 Double linked list

Double Linked List

Double linked list is a sequence of elements in which every element has links to its previous element and next element in the sequence.

```
struct Node
{
    struct Node *prev;
    int data;
    struct Node *next;
};
```

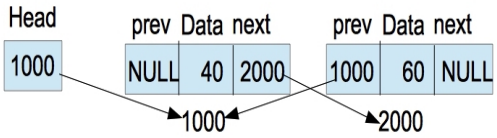


Operations on Double Linked List

In a double linked list we perform the following operations.

- Insertion (Beginning and end of the List)
- Deletion (Beginning and end from the List)
- Display/ Traversal

- In Single linked list, we can traverse in one direction.
- In double linked list, we can traverse both directions.



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("\nInsertion success!!!");
}
```

Delete Beginning from Double Linked List

```
void deleteBeginning(){
   if(head == NULL)
      printf("List 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);
      }
   }
}
```

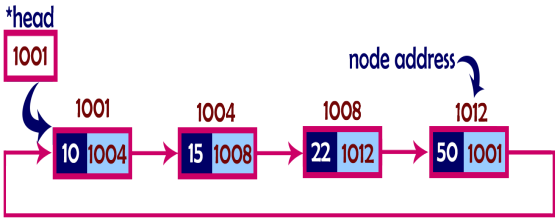
Short note for Double Linked List

- Insert in the end of double linked list and delete from the end of the double list, please check github repo.
- Display/ Traversal of the double linked list is similar as single linked list. Think about the logic of forward and backward traversal.

6.5 Circular Linked List

Circular Linked List

Circular linked list is a sequence of elements in which every element has link to its next element in the sequence and the last element has a link to the first element in the sequence.



7 References

References

References

[1] java point, https://www.javatpoint.com/

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

[3] DATA STRUCTURES, http://btechsmartclass.com/DS/U1_T1.html