

Name : Amruta Pede

Reg. no. :2020Bit070

Practical 1

Write c/ c++ code to implement concept of :

- 1) Stack
- 2) Queue
- 3) Linkedlist
- 4) Tree
- 5) Graph

1)Stack

```
#include<bits/stdc++.h>
```

```
using namespace std;
```

```
class Stack {
```

```
    int size;
```

```
    int * arr;
```

```
    int top;
```

```
public:
```

```
    Stack() {
```

```
        top = -1;
```

```
        size = 1000;
```

```
        arr = new int[size];
```

```
    }
```

```
    void push(int x) {
```

```
        top++;
```

```
        arr[top] = x;
```

```
    }
```

```

int pop() {
    int x = arr[top];
    top--;
    return x;
}

int Top() {
    return arr[top];
}

int Size() {
    return top + 1;
}

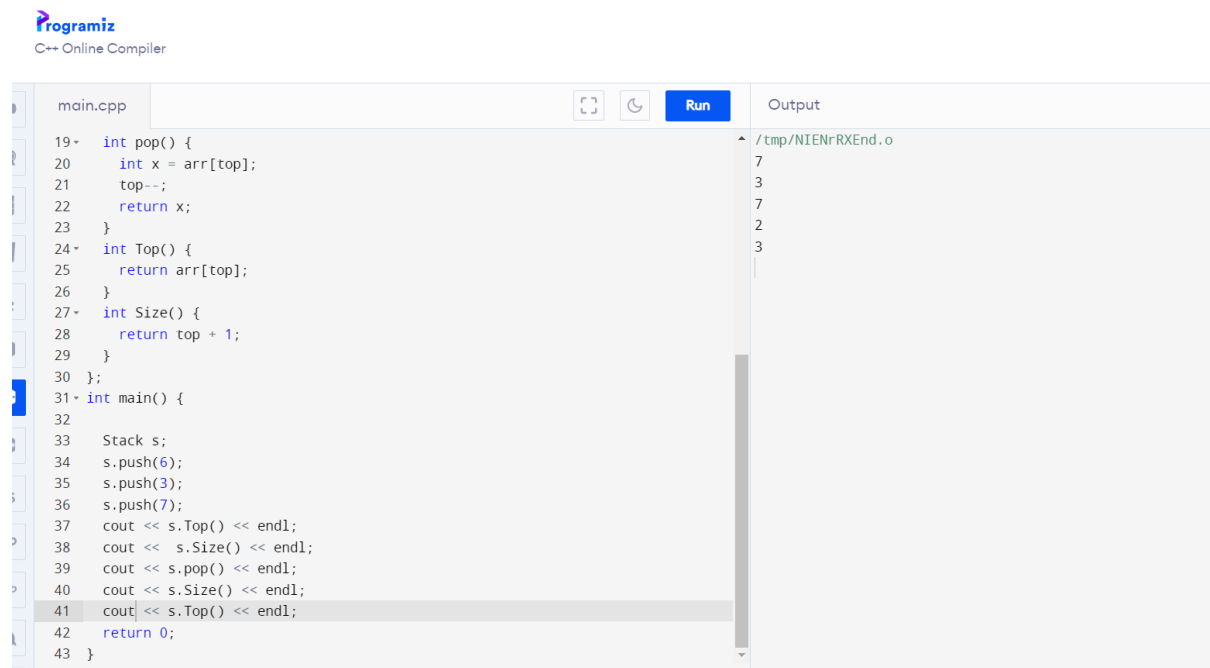
};

int main() {

    Stack s;
    s.push(6);
    s.push(3);
    s.push(7);
    cout << "Top of stack is before deleting any element " << s.Top() << endl;
    cout << "Size of stack before deleting any element " << s.Size() << endl;
    cout << "The element deleted is " << s.pop() << endl;
    cout << "Size of stack after deleting an element " << s.Size() << endl;
    cout << "Top of stack after deleting an element " << s.Top() << endl;
    return 0;
}

```

Output:



The screenshot shows the Programiz C++ Online Compiler interface. The code in `main.cpp` defines a stack structure with `pop()`, `Top()`, `Size()`, and `main()` functions. The `main()` function pushes 6, 3, and 7 onto the stack, then prints the top element, size, and top element again after popping. The output window shows the results: 7, 3, 7, 2, 3.

```
main.cpp
19+ int pop() {
20+     int x = arr[top];
21+     top--;
22+     return x;
23+ }
24+ int Top() {
25+     return arr[top];
26+ }
27+ int Size() {
28+     return top + 1;
29+ }
30+ };
31+ int main() {
32+
33+     Stack s;
34+     s.push(6);
35+     s.push(3);
36+     s.push(7);
37+     cout << s.Top() << endl;
38+     cout << s.Size() << endl;
39+     cout << s.pop() << endl;
40+     cout << s.Size() << endl;
41+     cout << s.Top() << endl;
42+     return 0;
43+ }
```

Output

```
/tmp/NIENrRXEnd.o
7
3
7
2
3
```

2)Queue

Question :

Implement Queue Data Structure using Array with all functions like pop, push, top, size, etc. Implement Queue Data Structure using Array with all functions like pop, push, top, size, etc.

```
#include<bits/stdc++.h>
```

```
using namespace std;
```

```
class Queue {
```

```
    int * arr;
```

```
    int start, end, currSize, maxSize;
```

```
public:
```

```
    Queue() {
```

```
        arr = new int[16];
```

```
    start = -1;
    end = -1;
    currSize = 0;
}
```

```
Queue(int maxSize) {
    (* this).maxSize = maxSize;
    arr = new int[maxSize];
    start = -1;
    end = -1;
    currSize = 0;
}
```

```
void push(int newElement) {
    if (currSize == maxSize) {
        cout << "Queue is full\nExiting..." << endl;
        exit(1);
    }
    if (end == -1) {
        start = 0;
        end = 0;
    } else
        end = (end + 1) % maxSize;
    arr[end] = newElement;
    cout << "The element pushed is " << newElement << endl;
    currSize++;
}
```

```
int pop() {
    if (start == -1) {
        cout << "Queue Empty\nExiting..." << endl;
    }
    int popped = arr[start];
    if (currSize == 1) {
        start = -1;
        end = -1;
    } else
        start = (start + 1) % maxSize;
    currSize--;
    return popped;
}

int top() {
    if (start == -1) {
        cout << "Queue is Empty" << endl;
        exit(1);
    }
    return arr[start];
}

int size() {
    return currSize;
}

};
```

```

int main() {

    Queue q(6);

    q.push(4);

    q.push(14);

    q.push(24);

    q.push(34);

    cout << "The peek of the queue before deleting any element " << q.top() <<
endl;

    cout << "The size of the queue before deletion " << q.size() << endl;

    cout << "The first element to be deleted " << q.pop() << endl;

    cout << "The peek of the queue after deleting an element " << q.top() << endl;


    cout << "The size of the queue after deleting an element " << q.size() << endl;


    return 0;

}

```

Output:


Online Compiler
Interac

main.cpp

52

cout << "Queue is Empty" << endl;

53

exit(1);

54

}

55

return arr[start];

56

}

57

int size() {

58

return currSize;

59

}

60

}

61

};

62

63

int main() {

64

Queue q(6);

65

q.push(4);

66

q.push(14);

67

q.push(24);

68

q.push(34);

69

cout << "The peek of the queue before deleting any element " << q.top() <<

endl;

70

cout << "The size of the queue before deletion " << q.size() << endl;

71

cout << "The first element to be deleted " << q.pop() << endl;

72

cout << "The peek of the queue after deleting an element " << q.top() << endl

;

73

cout << "The size of the queue after deleting an element " << q.size() <<

endl;

Run

Output

/tmp/1whdYqzYwp.o

The element pushed is 4

The element pushed is 14

The element pushed is 24

The element pushed is 34

The peek of the queue before deleting any element 4

The size of the queue before deletion 4

The first element to be deleted 4

The peek of the queue after deleting an element 14

The size of the queue after deleting an element 3

3)linkedlist :

```
// A complete working C++ program to
// demonstrate all insertion methods
// on Linked List
#include <bits/stdc++.h>
using namespace std;

// A linked list node
class Node
{
    public:
    int data;
    Node *next;
};

// Given a reference (pointer to pointer)
// to the head of a list and an int, inserts
// a new node on the front of the list.
void push(Node** head_ref, int new_data)
{
    // 1. allocate node
    Node* new_node = new Node();

    // 2. put in the data
    new_node->data = new_data;

    // 3. Make next of new node as head
    new_node->next = (*head_ref);
```

```

// 4. move the head to point
// to the new node
(*head_ref) = new_node;
}

// Given a node prev_node, insert a new
// node after the given prev_node
void insertAfter(Node* prev_node, int new_data)
{
    // 1. check if the given prev_node
    // is NULL
    if (prev_node == NULL)
    {
        cout<<"The given previous node cannot be NULL";
        return;
    }

    // 2. allocate new node
    Node* new_node = new Node();

    // 3. put in the data
    new_node->data = new_data;

    // 4. Make next of new node
    // as next of prev_node
    new_node->next = prev_node->next;

    // 5. move the next of prev_node
    // as new_node
    prev_node->next = new_node;
}

```



```

// Given a reference (pointer to pointer)
// to the head of a list and an int,
// appends a new node at the end
void append(Node** head_ref, int new_data)
{

    // 1. allocate node
    Node* new_node = new Node();

    //used in step 5
    Node *last = *head_ref;

    // 2. put in the data
    new_node->data = new_data;

    /* 3. This new node is going to be
    the last node, so make next of
    it as NULL*/
    new_node->next = NULL;

    /* 4. If the Linked List is empty,
    then make the new node as head */
    if (*head_ref == NULL)
    {
        *head_ref = new_node;
        return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)

```

```

{
    last = last->next;
}

/* 6. Change the next of last node */
last->next = new_node;
return;
}

// This function prints contents of
// linked list starting from head
void printList(Node *node)
{
    while (node != NULL)
    {
        cout<<" "<<node->data;
        node = node->next;
    }
}

// Driver code
int main()
{ // Start with the empty list
    Node* head = NULL;

    // Insert 6. So linked list becomes 6->NULL
    append(&head, 6);

    // Insert 7 at the beginning.
    // So linked list becomes 7->6->NULL
    push(&head, 7);

```

```
// Insert 1 at the beginning. So linked list becomes 1->7->6->NULL
```

```
push(&head, 1);
```

```
// Insert 4 at the end. So linked list becomes 1->7->6->4->NULL
```

```
append(&head, 4);
```

```
// Insert 8, after 7. So linked list becomes 1->7->8->6->4->NULL
```

```
insertAfter(head->next, 8);
```



```
cout<<"Created Linked list is: ";
```


```
printList(head);
```

```
return 0;
```

```
}
```

Output :



main.cpp	Run	Output
<pre>120 // Insert 6. So linked list becomes 6->NULL 121 append(&head, 6); 122 123 // Insert 7 at the beginning. 124 // So linked list becomes 7->6->NULL 125 push(&head, 7); 126 127 // Insert 1 at the beginning. 128 // So linked list becomes 1->7->6->NULL 129 push(&head, 1); 130 131 // Insert 4 at the end. So 132 // linked list becomes 1->7->6->4->NULL 133 append(&head, 4); 134 135 // Insert 8, after 7. So linked 136 // list becomes 1->7->8->6->4->NULL 137 insertAfter(head->next, 8); 138 139 cout<<"Created Linked list is: "; 140 printList(head); 141 142 return 0; 143 } 144</pre>		<pre>/tmp/8XsIdTNVyd.o Created Linked list is: 1 7 8 6 4</pre>

4) Tree :

// C++ program of Inorder tree traversals

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
/* A binary tree node has data, pointer to left child  
and a pointer to right child */
```

```
struct Node {
```

```
    int data;
```

```
    struct Node *left, *right;
```

```
};
```

```
// Utility function to create a new tree node
```

```
Node* newNode(int data)
```

```
{
```

```
    Node* temp = new Node;
```

```
    temp->data = data;
```

```
    temp->left = temp->right = NULL;
```

```
    return temp;
```

```
}
```

```
/* Given a binary tree, print its nodes in inorder*/
```

```
void printInorder(struct Node* node)
```

```
{
```

```
    if (node == NULL)
```

```
        return;
```

```
    /* first recur on left child */
```

```

    printInorder(node->left);

    /* then print the data of node */
    cout << node->data << " ";

    /* now recur on right child */
    printInorder(node->right);
}

/* Driver code*/
int main()
{
    struct Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    // Function call
    cout << "\nInorder traversal of binary tree is \n";
    printInorder(root);

    return 0;
}

```

Output :

```
main.cpp  [Icons] [Run]
34  /* now recur on right child */
35  printInorder(node->right);
36  }
37
38  /* Driver code*/
39  int main()
40  {
41      struct Node* root = newNode(1);
42      root->left = newNode(2);
43      root->right = newNode(3);
44      root->left->left = newNode(4);
45      root->left->right = newNode(5);
46
47
48      // Function call
49      cout << "\nInorder traversal of binary tree is \n";
50      printInorder(root);
51
52
53      return 0;
54  }
55
```

Output

```
/tmp/ux40Cxjbu8.o
Inorder traversal of binary tree is
4 2 5 1 3
```

5)Graph :

// A simple representation of graph using STL

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

// A utility function to add an edge in an

// undirected graph.

```
void addEdge(vector<int> adj[], int u, int v)
```

```
{
```

```
    adj[u].push_back(v);
```

```
    adj[v].push_back(u);
```

```
}
```

// A utility function to print the adjacency list

// representation of graph

```
void printGraph(vector<int> adj[], int V)
```

```
{
```

```
    for (int v = 0; v < V; ++v) {
```

```
        cout << "\n Adjacency list of vertex " << v
```

```
        << "\n head ";
```

```
        for (auto x : adj[v])
            cout << "-> " << x;
        printf("\n");
    }
}
```

// Driver code

```
int main()
{
    int V = 5;
    vector<int> adj[V];
    addEdge(adj, 0, 1);
    addEdge(adj, 0, 4);
    addEdge(adj, 1, 2);
    addEdge(adj, 1, 3);
    addEdge(adj, 1, 4);
    addEdge(adj, 2, 3);
    addEdge(adj, 3, 4);
    printGraph(adj, V);
    return 0;
}
```

Output :

main.cpp		Run	Output
19	cout << "\n Adjacency list of vertex " << v		/tmp/giTyFQJ260.o
20	cout << "\n head ";		Adjacency list of vertex 0
21	for (auto x : adj[v])		head -> 1-> 4
22	cout << "-> " << x;		
23	printf("\n");		Adjacency list of vertex 1
24	}		head -> 0-> 2-> 3-> 4
25	}		Adjacency list of vertex 2
26			head -> 1-> 3
27	// Driver code		
28	int main()		Adjacency list of vertex 3
29	{		head -> 1-> 2-> 4
30	int V = 5;		
31	vector<int> adj[V];		Adjacency list of vertex 4
32	addEdge(adj, 0, 1);		head -> 0-> 1-> 3
33	addEdge(adj, 0, 4);		
34	addEdge(adj, 1, 2);		
35	addEdge(adj, 1, 3);		
36	addEdge(adj, 1, 4);		
37	addEdge(adj, 2, 3);		
38	addEdge(adj, 3, 4);		
39	printGraph(adj, V);		
40	return 0;		
41	}		
42			
43			