Batch: B2	**Roll No.:** 16010124107
Experiment / assignment / tutorial No. 3	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Dynamic implementation of Stack- Creation, Insertion, Deletion, Peek

Objective: To implement Basic Operations of Stack dynamically

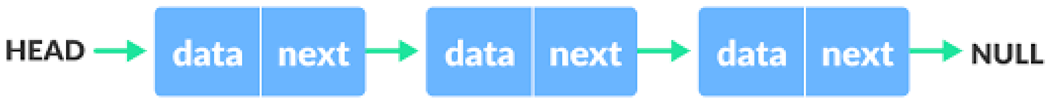
Expected Outcome of Experiment:

CO	Outcome
2	Apply linear and non-linear data structure in application development.

Books/ Journals/ Websites referred:

https://www.geeksforgeeks.org/dsa/implement-a-stack-using-singly-linked-list/

Introduction:



In a linked list representation of a stack, the topmost node (the node pointing to null in the image), is the top. When an element is inserted in the stack, it is inserted at the top of the linked list. So it takes its place after the third element in the diagram. When an element is popped from the linked list, it happens at the "end" of the linked list.

Therefore, we can replicate the working of a stack using a linked list by using only one end of it for insertion and deletion.

Linked List ADT:

- create() – create an empty list
- insert(pos, data) – insert data at position
- delete(pos) – delete node at position
- search(data) – find element
- display() – show all elements
- isEmpty() – check if list empty

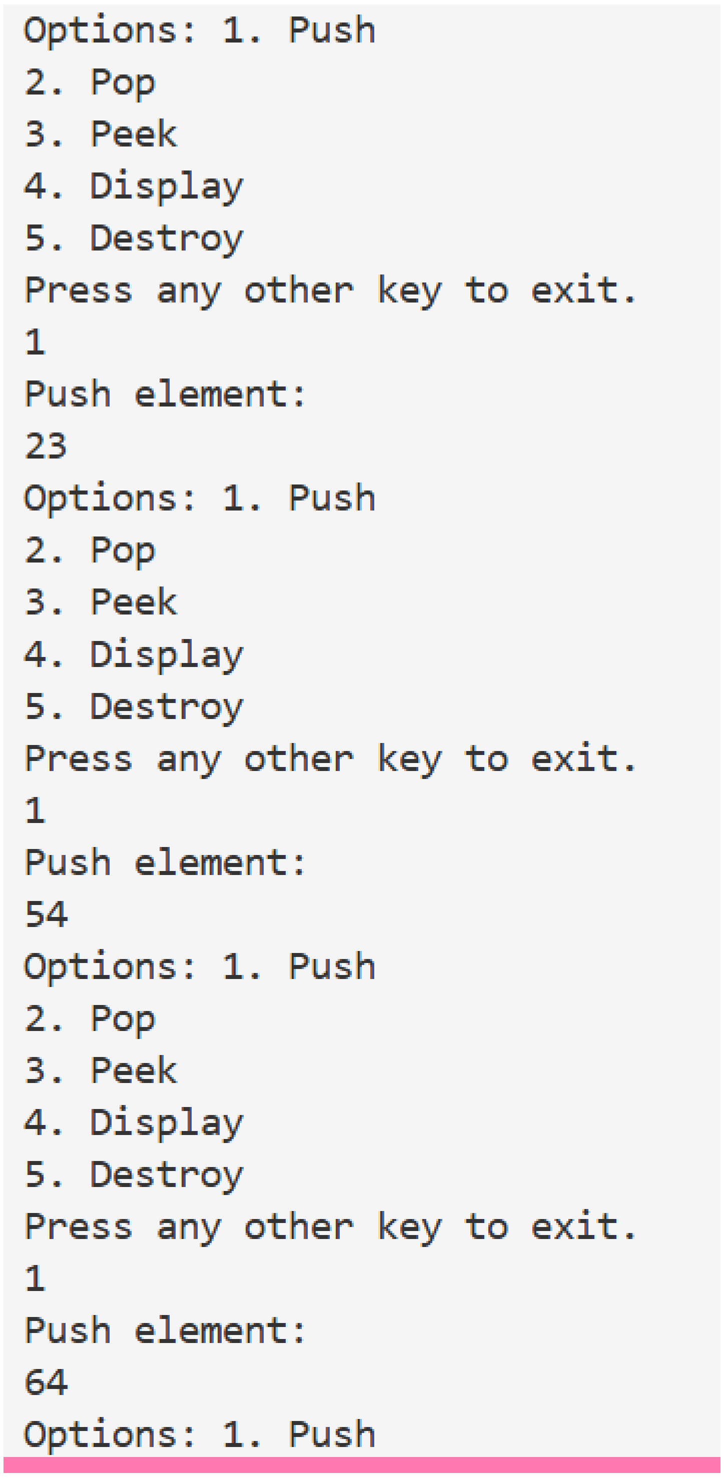
<u>*Algorithm for creation, insertion, deletion, traversal and searching an element in dynamic stack using linked list:*</u>

1. Start
2. Initialize top = NULL
3. Repeat until exit option chosen
4. Display menu
5. Read user choice
6. If choice = 1, create new node, insert at top
7. If choice = 2, delete node from top
8. If choice = 3, display top element
9. If choice = 4, display all elements
10. If choice = 5, delete all nodes
11. Else exit
12. Stop

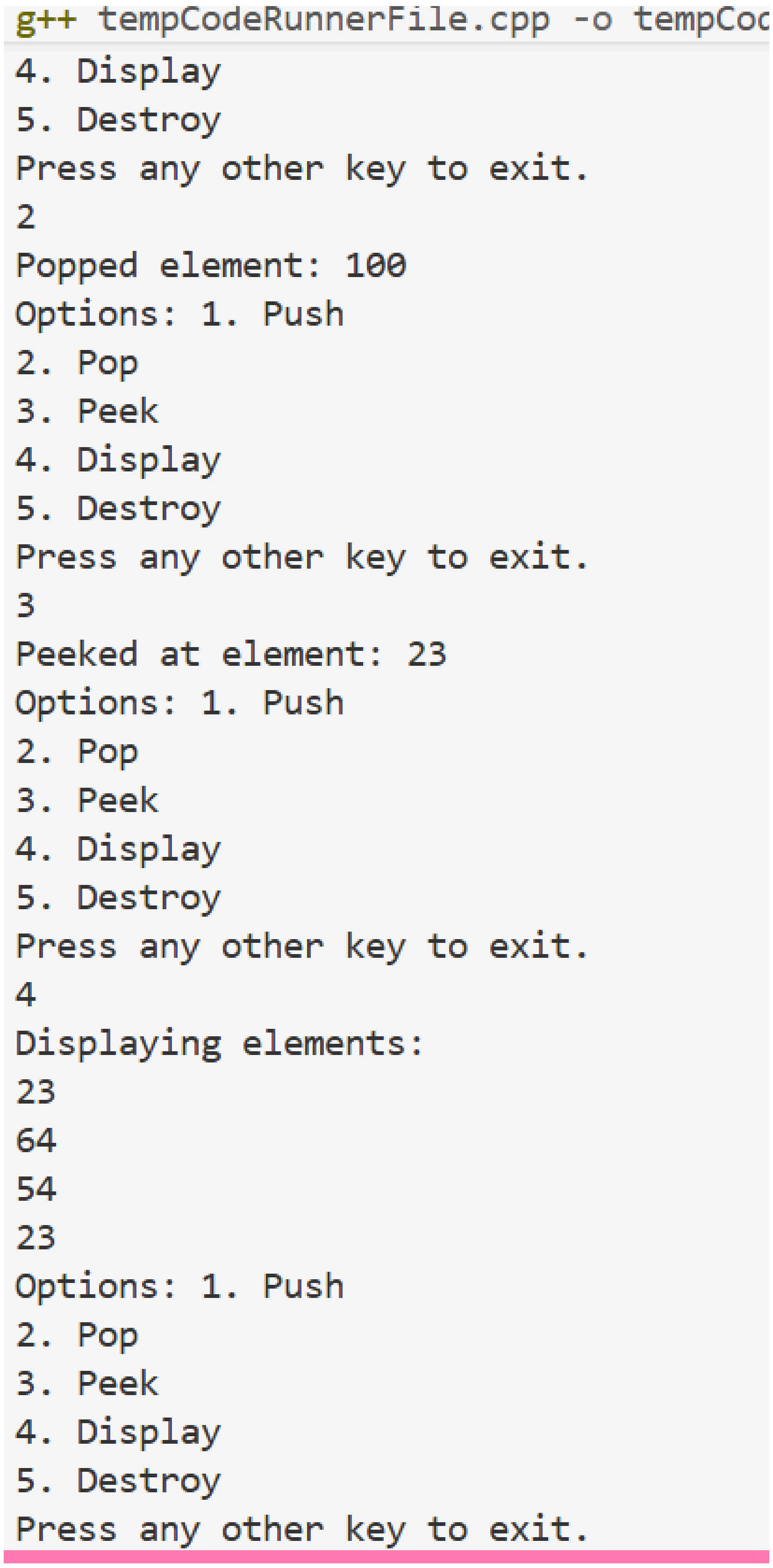
Program source code:

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

using namespace std;


struct ListNode

{

    int value;

    ListNode* next;

};


int main()

{

    ListNode* top = nullptr;

    while(true)

    {

        cout  <<  "Options:  1.  Push\n2.  Pop\n3.  Peek\n4.  Display\n5.
Destroy\nPress any other key to exit.\n";

    int choice;

    cin >> choice;

    switch(choice)

    {

        case 1:

            {ListNode* newnode = new ListNode();

            if(newnode == NULL)

            {
                cout << "Overload\n";

            }

            else

            {

                cout << "Push element:\n";

                int el;

                cin >> el;

                newnode->value = el;

                  newnode->next = top; //points to the nullptr initially,
and the topmost element lateron

                top = newnode;

            }

            break;


        }

        case 2:

        if(top==NULL )

        {

            cout << "Underflow\n";

        }

        else

        {

            cout << "Popped element: ";

            ListNode* temp = top;

            cout << temp->value << endl;
            top = top->next;

            delete temp;


        }break;




        case 3:

        {if(top==NULL )

        {

            cout << "Underflow\n";

        }

        else

        {

            cout << "Peeked at element: ";

            cout << top->value << endl;

        }

        break;}

        case 4:

        {if(top==NULL )

        {

            cout << "Underflow\n";

        }

        else

        {
            cout << "Displaying elements: \n";

            ListNode* temp = top;

            while(temp!=NULL)

            {

                cout << temp->value << endl;

                temp = temp->next;

            }

        }

        break;

    }

        case 5:

        {if(top==NULL )

        {

            cout << "Underflow\n";

        }

        else

        {

            cout << "Destroying elements: \n";

            while(top!=NULL)

            {

                ListNode* temp = top;

                cout << temp->value << endl;

                top = top->next;

                delete temp;
            }
        } break;
    }

        default:
        {
            cout << "Exiting" << endl;
            return 0;
        }
    }
}
}
```

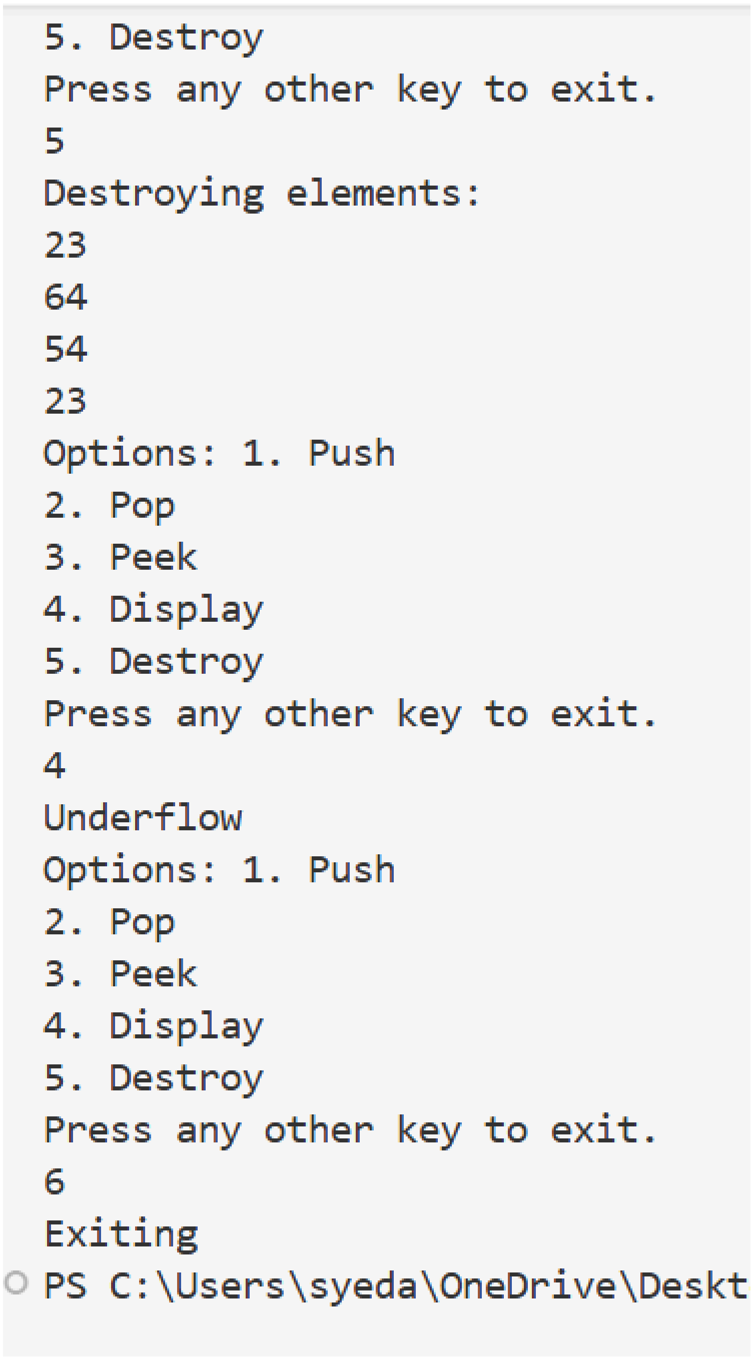
Output Screenshots:



```
Options: 1. Push
2. Pop
3. Peek
4. Display
5. Destroy
Press any other key to exit.
1
Push element:
23
Options: 1. Push
2. Pop
3. Peek
4. Display
5. Destroy
Press any other key to exit.
1
Push element:
54
Options: 1. Push
2. Pop
3. Peek
4. Display
5. Destroy
Press any other key to exit.
1
Push element:
64
Options: 1. Push
```



```
g++ tempCodeRunnerFile.cpp -o tempCoc
4. Display
5. Destroy
Press any other key to exit.
2
Popped element: 100
Options: 1. Push
2. Pop
3. Peek
4. Display
5. Destroy
Press any other key to exit.
3
Peeked at element: 23
Options: 1. Push
2. Pop
3. Peek
4. Display
5. Destroy
Press any other key to exit.
4
Displaying elements:
23
64
54
23
Options: 1. Push
2. Pop
3. Peek
4. Display
5. Destroy
Press any other key to exit.
```



```
5. Destroy
Press any other key to exit.
5
Destroying elements:
23
64
54
23
Options: 1. Push
2. Pop
3. Peek
4. Display
5. Destroy
Press any other key to exit.
4
Underflow
Options: 1. Push
2. Pop
3. Peek
4. Display
5. Destroy
Press any other key to exit.
6
Exiting
PS C:\Users\syeda\OneDrive\Deskt
```

Conclusion:-

The program implements a stack using a linked list. It performs push, pop, peek, display, and destroy operations dynamically. Memory is efficiently managed using node allocation and deletion.

Post lab questions:

1. Explain how Stacks can be used in Backtracking algorithms with examples

Stacks store states to retrace steps during backtracking. When a choice is made, it's pushed onto the stack. If the choice leads to a dead end, it's popped to backtrack. For example, in maze tracking algorithms, each cell visited is pushed onto the stack. When no path is found, the last cell is popped to try another path.

2. Discuss the advantages and disadvantages of using static & dynamic memory allocation for implementing a stack. How does the fixed size of the stack impact its performance and usability?
Advantages of static memory allocation:

1. Faster access to element
2. Less pointer management needed

Disadvantages:

1. Memory is unoptimized: we may over or underestimate memory needed.
2. Less mirrored to real world applications

Advantages of dynamic memory allocation

1. Allows for dynamic resizing
2. Lets users push and pop indefinitely, mirroring the real world applications

Disadvantages:

1. Complicated to implement due to pointer management
2. Can be slower than static memory because it needs constant memory allocation
3. Takes more space because it needs more memory to store pointer to the next element in each cell.

The fixed size of a stack increases its performance as memory needed is pre-allocated to the stack and hence does not need several reallocation function calls during runtime. It increases usability also since its faster and more preferred when the size of data is known beforehand, for example, in writing student records for a set class of students.