Week 2 Striver

1. Program to Implement Stack using array

```
#include <iostream>
using namespace std;
int stack[100], n=100, top=-1;
void push(int val) {
  if(top>=n-1)
  cout<<"Stack Overflow"<<endl;
  else {
    top++;
    stack[top]=val;
  }
}
void pop() {
  if(top<=-1)
  cout<<"Stack Underflow"<<endl;
  else {
    cout<<"The popped element is "<< stack[top] <<endl;
    top--;
  }
}
void display() {
  if(top>=0) {
    cout<<"Stack elements are:";
    for(int i=top; i>=0; i--)
    cout<<stack[i]<<" ";
    cout<<endl;
  } else
```

```
  cout<<"Stack is empty";
}
int main() {
  int ch, val;
  cout<<"1) Push in stack"<<endl;
  cout<<"2) Pop from stack"<<endl;
  cout<<"3) Display stack"<<endl;
  cout<<"4) Exit"<<endl;
  do {
    cout<<"Enter choice: "<<endl;
    cin>>ch;
    switch(ch) {
      case 1: {
        cout<<"Enter value to be pushed:"<<endl;
        cin>>val;
        push(val);
        break;
      }
      case 2: {
        pop();
        break;
      }
      case 3: {
        display();
        break;
      }
      case 4: {
        cout<<"Exit"<<endl;
        break;
      }
      default: {
```

```
        cout<<"Invalid Choice"<<endl;
      }
    }
  }while(ch!=4);
  return 0;
}
```

2. Implement Queue Using Arrays

```
#include <iostream>
using namespace std;
int queue[100], n = 100, front = - 1, rear = - 1;
void Insert() {
  int val;
  if (rear == n - 1)
  cout<<"Queue Overflow"<<endl;
  else {
    if (front == - 1)
    front = 0;
    cout<<"Insert the element in queue : "<<endl;
    cin>>val;
    rear++;
    queue[rear] = val;
```

```
  }
}
void Delete() {
  if (front == - 1 || front > rear) {
    cout<<"Queue Underflow ";
    return ;
  } else {
    cout<<"Element deleted from queue is : "<<
queue[front] <<endl;
    front++;;
  }
}
void Display() {
  if (front == - 1)
  cout<<"Queue is empty"<<endl;
  else {
    cout<<"Queue elements are : ";
    for (int i = front; i <= rear; i++)
    cout<<queue[i]<<" ";
      cout<<endl;
```

```
  }
}
int main() {
  int ch;
  cout<<"1) Insert element to queue"<<endl;
  cout<<"2) Delete element from queue"<<endl;
  cout<<"3) Display all the elements of queue"<<endl;
  cout<<"4) Exit"<<endl;
  do {
    cout<<"Enter your choice : "<<endl;
    cin>>ch;
    switch (ch) {
      case 1: Insert();
      break;
      case 2: Delete();
      break;
      case 3: Display();
      break;
      case 4: cout<<"Exit"<<endl;
      break;
```

```
        default: cout<<"Invalid choice"<<endl;
      }
   } while(ch!=4);
   return 0;
}
```

3. Implement a stack using single queue

```
#include<bits/stdc++.h>
using namespace std;
class Stack
{
      queue<int>q;
public:
      void push(int val);
      void pop();
      int top();
      bool empty();
};
```

```
void Stack::push(int val)
{

	int s = q.size();


	q.push(val);


	for (int i=0; i<s; i++)
	{

		q.push(q.front());


		q.pop();
	}
}
```

```
void Stack::pop()
{
	if (q.empty())
		cout << "No elements\n";
	else
		q.pop();
}

int Stack::top()
{
	return (q.empty())? -1 : q.front();
}

bool Stack::empty()
{
	return (q.empty());
}
```

```
int main()
{
	Stack s;
	s.push(10);
	s.push(20);
	cout << s.top() << endl;
	s.pop();
	s.push(30);
	s.pop();
	cout << s.top() << endl;
	return 0;
}
```

4. Sort a Stack

```
// C++ program to sort a stack using recursion
#include <iostream>
using namespace std;

// Stack is represented using linked list
struct stack {
```

```
	int data;
	struct stack* next;
};

// Utility function to initialize stack
void initStack(struct stack** s) { *s = NULL; }

// Utility function to check if stack is empty
int isEmpty(struct stack* s)
{
	if (s == NULL)
		return 1;
	return 0;
}

// Utility function to push an item to stack
void push(struct stack** s, int x)
{
	struct stack* p = (struct stack*)malloc(sizeof(*p));
```

```
        if (p == NULL) {
                fprintf(stderr, "Memory allocation failed.\n");
                return;
        }

        p->data = x;
        p->next = *s;
        *s = p;
}

// Utility function to remove an item from stack
int pop(struct stack** s)
{
        int x;
        struct stack* temp;

        x = (*s)->data;
        temp = *s;
        (*s) = (*s)->next;
        free(temp);
```

```
    return x;
}

// Function to find top item
int top(struct stack* s) { return (s->data); }

// Recursive function to insert an item x in sorted way
void sortedInsert(struct stack** s, int x)
{
    // Base case: Either stack is empty or newly
inserted
    // item is greater than top (more than all existing)
    if (isEmpty(*s) or x > top(*s)) {
        push(s, x);
        return;
    }

    // If top is greater, remove the top item and recur
    int temp = pop(s);
```

```
        sortedInsert(s, x);

        // Put back the top item removed earlier
        push(s, temp);
}

// Function to sort stack
void sortStack(struct stack** s)
{
        // If stack is not empty
        if (!isEmpty(*s)) {
                // Remove the top item
                int x = pop(s);

                // Sort remaining stack
                sortStack(s);

                // Push the top item back in sorted stack
                sortedInsert(s, x);
        }
```

```
}

// Utility function to print contents of stack
void printStack(struct stack* s)
{
	while (s) {
		cout << s->data << " ";
		s = s->next;
	}
	cout << "\n";
}

// Driver code
int main(void)
{
	struct stack* top;

	initStack(&top);
	push(&top, 30);
	push(&top, -5);
```

```
    push(&top, 18);
    push(&top, 14);
    push(&top, -3);

    cout << "Stack elements before sorting:\n";
    printStack(top);

    sortStack(&top);
    cout << "\n";

    cout << "Stack elements after sorting:\n";
    printStack(top);

    return 0;
}
```