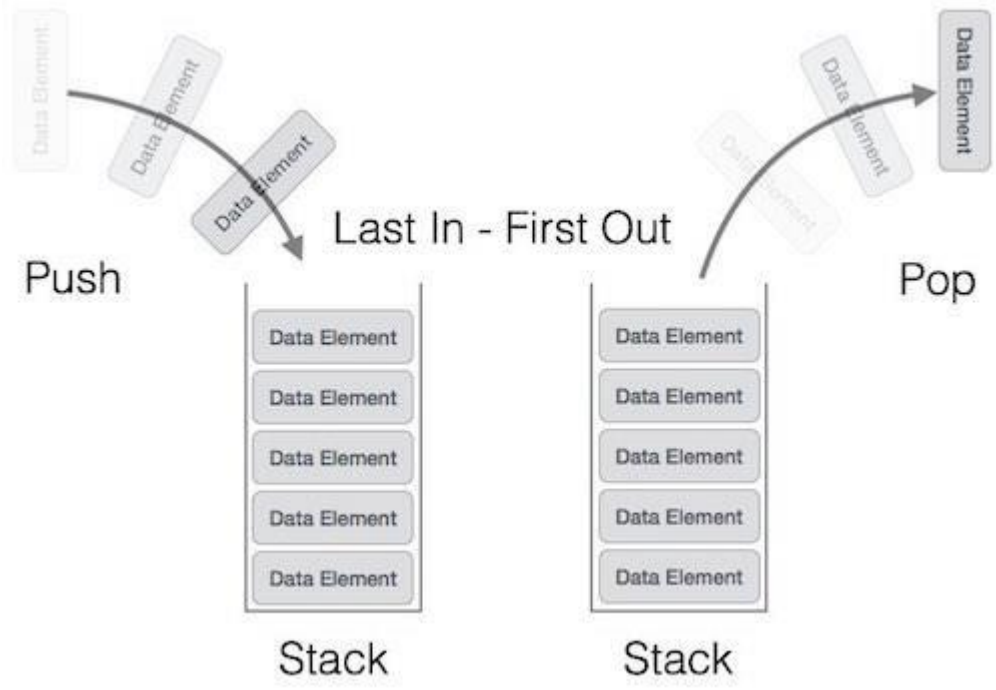
Assignment 4: StackADT and its application

Create an ADT for the stack data structure with the following functions. stackADT will have the integer array, top and size.
createStack(top) – initialize size and top with -1

a. push(top,data) – push data into the stack if the stack is not full. Print a message when stack isfull
b. pop(top) – decrements the top by 1
c. peek(top)– returns the element at top, if stack is not empty, otherwise returns -1
d. isEmpty(top) – returns 1 if stack empty, otherwise returns 0
e. isFull(top) – returns 1 if stack full, otherwise returns 0

DESIGN:



stackADT.h

```
#include <stdio.h>
#include <stdlib.h>
struct stack {
int top;
int data[100];
int size;
};
```

```
void init(struct stack *s, int limit) {
s->size = limit;
s->top = -1;
}
int isFull(struct stack *s) {
return s->top == s->size - 1;
}
int isEmpty(struct stack *s) {
return s->top == -1;
}
void push(struct stack *s, int d) {
if (isFull(s)) {
printf("Stack is full. %d.\n", d);
} else {
s->data[++s->top] = d;
printf("\nPushed: %d", d);
}
}
void pop(struct stack *s) {
if (isEmpty(s)) {
printf("Stack is empty.\n");
} else {
printf("Popped: %d\n", s->data[s->top]);
s->top--;
}
}
int peek(struct stack *s) {
if (isEmpty(s)) {
printf("Stack is empty.\n");
return -1;
} else {

return s->data[s->top];
}
}
```

Main.h

```
#include<stdio.h>
#include<stdlib.h>
#include"stackADT"
void main()
{
```

```
  struct stack*s;
  s=(struct stack*)malloc(sizeof(struct stack));
  printf("\ntop element:%d",peek(s));
  push(s,2);
  push(s,4);
  push(s,6);
  push(s,8);
  pop(s);
  printf("\ntop element :%d",peek(s));
  printf("\ntop element:%d",peek(s));
  pop(s);
  pop(s);


}
```

Output:

```
top element:-1
Pushed: 2

Pushed: 4

Pushed: 6

Pushed: 8
Popped: 8

top element :8
top element:8Popped: 8
Popped: 8
```

APPLICATION:

1. Evaluate the infix to postfix expression using Stack
Example: (2+3)*(4+5)
Ans: 23+45+*

ALGORITHM :
Input: char *infix, char *postfix
Output:void
Variable: Character,integer

```
        int i, j;
        i = j = 0;
        WHILE(infix[i] != '\0')
        char token = infix[i];
        if (token >= '0' && token <= '9')
        postfix[j++] = token; // If the token is a digit, add it to the output
        else if (token == '(')
        push(&s, token); // If the token is '(', push it onto the stack
        else if (token == ')')
      WHILE (peek(&s) != '(')
                postfix[j++] = peek(&s); // Pop and add operators from the stack to the
output until '(' is encountered
                pop(&s);
        pop(&s); // Pop and discard the '(‘
        else if (isOperator(token)
    WHILE (!isEmpty(&s) && getPrecedence(peek(&s)) >= getPrecedence(token))
                postfix[j++] = peek(&s);
                pop(&s)
                push(&s, token); // Push the current operator onto the stack
                i++;
        WHILE(!isEmpty(&s))
           postfix[j++] = peek(&s); // Pop and add remaining operators on the stack to
           pop(&s);
           postfix[j] = '\0'; // Null-terminate the postfix expression
```

Main.h

```
#include<stdio.h>
#include<stdlib.h>
#include "stackADT"
int main()
{
        char infix[] = "(2+3)*(4+5)";
        char postfix[100];

        infixToPostfix(infix, postfix);

        printf("Infix Expression: %s\n", infix);
        printf("Postfix Expression: %s\n", postfix);
```

```
        return 0;
}
```

“stackADT.h”

```
#include<stdio.h>
#include<stdlib.h>

struct stack
{
        int size;
        int top;
        int data[100];
};

void init(struct stack *s,int limit)
{
        s->size = limit;
        s->top = -1;
}

void push(struct stack *s, int x)
{
        if (!isFull(s))
        {
        s->data[++(s->top)] = x;
        printf("\nPushed %d\n",x);
        }
        else
        {
        printf("\nStack is full\n");
        }
}

int isFull(struct stack *s)
{
        if (s->top < s->size-1)
        {
        return 0;
        }
        else
        {
        return 1;
        }
```

```
}

void pop(struct stack *s)
{
        if (!isEmpty(s))
        {
        s->top --;
        }
        else
        {
        printf("\nStack is empty\n");
        }
}

int isEmpty (struct stack *s)
{
        if (s->top == -1)
        {
        return 1;
        }
        else
        {
        return 0;
        }
}

int peek(struct stack *s)
{
        if (!isEmpty(s))
        {
        return s->data[s->top];
        }
        else
        {
        return -1;
        }
}

int isOperator(char c)
{
        return (c == '+' || c == '-' || c == '*' || c == '/');
}

int getPrecedence(char c)
```

```
{
        if (c == '+' || c == '-')
        return 1;
        if (c == '*' || c == '/')
        return 2;
        return 0;
}

void infixToPostfix(char *infix, char *postfix)
{
        struct stack s;
        init(&s, strlen(infix));

        int i, j;
        i = j = 0;

        while (infix[i] != '\0')
        {
        char token = infix[i];

        if (token >= '0' && token <= '9')
        {
        postfix[j++] = token;
        }
        else if (token == '(')
        {
        push(&s, token);
        }
        else if (token == ')')
        {
        while (peek(&s) != '(')
        {
                postfix[j++] = peek(&s);
                pop(&s);
        }
        pop(&s); // Pop the '('
        }
        else if (isOperator(token))
        {
        while (!isEmpty(&s) && getPrecedence(peek(&s)) >= getPrecedence(token))
        {
                postfix[j++] = peek(&s);
                pop(&s);
        }
```

```
        push(&s, token);
        }
        i++;
        }

        while (!isEmpty(&s))
        {
        postfix[j++] = peek(&s);
        pop(&s);
        }

        postfix[j] = '\0';
}

void decimalToBinary(int decimal)
{
        struct stack s;
        init(&s, 32);

        while (decimal > 0)
        {
        push(&s, decimal % 2);
        decimal /= 2;
        }

        printf("Binary Representation: ");
        while (!isEmpty(&s))
        {
        printf("%d", peek(&s));
        pop(&s);
        }
        printf("\n");
}
```

Output;

```
Infix Expression: (2+3)*(4+5)
Postfix Expression: 23+45+*
```

2.Convert the given decimal number into binary using stack

Example: 14

Ans: 1110

ALGORITHM:
Input:int decimal
Output:void
Variable:integer

WHILE(decimal >0)
    push(s,decimal%2)
    decimal/=2;
END WHILE

printf(“binary”)
WHILE(!isempty(s))
    printf(peek(s));
    pop(s);

Main.h

```
#include "stackADT"
#include <stdio.h>
#include <stdlib.h>
int main() {
struct node *s1;
s1 = (struct stack *)malloc(sizeof(struct stack));
Binary(14);
return 0;
}
```

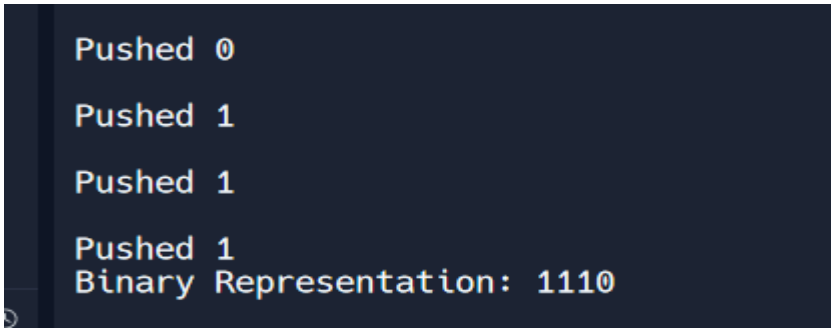
stackADT.h

```
#include<stdio.h>
#include<stdio.h>

struct stack
{
        int size;
        int top;
        int data[100];

};

void init(struct stack *s,int limit)
{
```

```
        s->size = limit;
        s->top = -1;
}

void push(struct stack *s, int x)
{
        if (!isFull(s))
        {
        s->data[++(s->top)] = x;
        printf("\nPushed %d\n",x);
        }
        else
        {
        printf("\nStack is full\n");
        }
}

int isFull(struct stack *s)
{
        if (s->top < s->size-1)
        {
        return 0;
        }
        else
        {
        return 1;
        }
}

void pop(struct stack *s)
{
        if (!isEmpty(s))
        {
        s->top --;
        }
        else
        {
        printf("\nStack is empty\n");
        }
}

int isEmpty (struct stack *s)
{
        if (s->top == -1)
```

```
        {
        return 1;
        }
        else
        {
        return 0;
        }
}

int peek(struct stack *s)
{
        if (!isEmpty(s))
        {
        return s->data[s->top];
        }
        else
        {
        return -1;
        }
}



void Binary(int decimal)
{
        struct stack s;
        init(&s, 32);

        while (decimal > 0)
        {
        push(&s, decimal % 2);
        decimal /= 2;
        }

        printf("Binary Representation: ");
        while (!isEmpty(&s))
        {
        printf("%d", peek(&s));
        pop(&s);
        }
        printf("\n");
}
```

Output:



3.Find next greater element using stack

Main.h

```
int main() {
	int arr[] = {11,13,21,3};
	int n = sizeof(arr) / sizeof(arr[0]);
	GreaterEle(arr, n);
	return 0;
}
```

stackADT.h

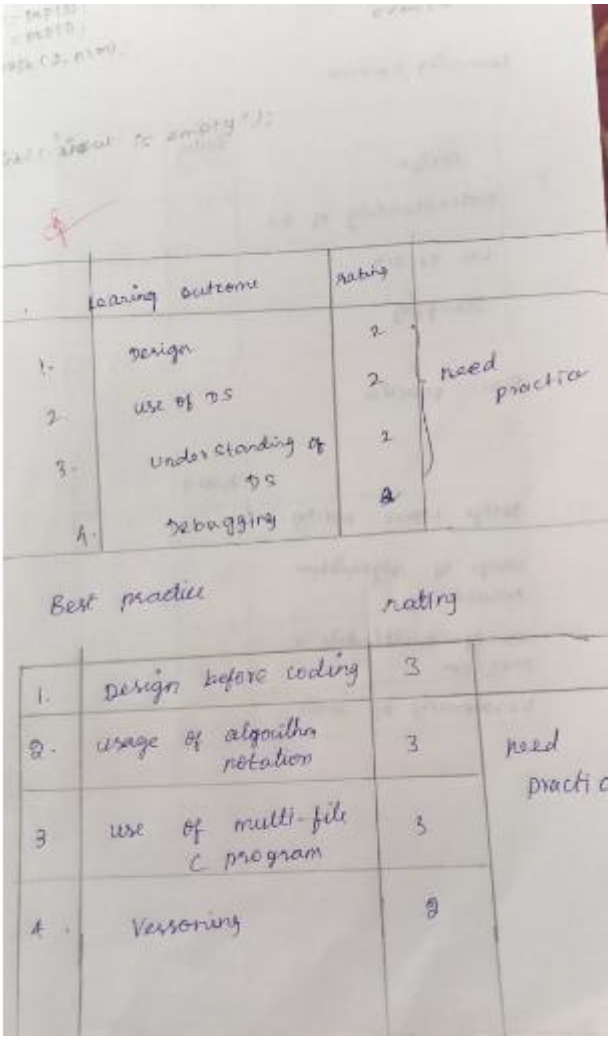
```
#include <stdio.h>
#include <stdlib.h>

struct Stack {
	int top;
	int capacity;
	int* array;
};

struct Stack* createStack(int capacity) {
	struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
	stack->top = -1;
	stack->capacity = capacity;
	stack->array = (int*)malloc(stack->capacity * sizeof(int));
	return stack;
```

```
}

int isEmpty(struct Stack* stack) {
        return (stack->top == -1);
}

void push(struct Stack* stack, int item) {
        stack->array[++stack->top] = item;
}

int pop(struct Stack* stack) {
        if (isEmpty(stack))
        return -1;
        return stack->array[stack->top--];
}

void GreaterEle(int arr[], int n) {
        struct Stack* stack = createStack(n);
        int* result = (int*)malloc(n * sizeof(int));

        for (int i = 0; i < n; i++) {
        while (!isEmpty(stack) && arr[i] > arr[stack->array[stack->top]]) {
        int prevIndex = pop(stack);
        result[prevIndex] = arr[i];
        }
        push(stack, i);
        }

        while (!isEmpty(stack)) {
        int prevIndex = pop(stack);
        result[prevIndex] = -1;
        }

        printf("Original Array: ");
        for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
        }
        printf("\nNext Greater Elements: ");
        for (int i = 0; i < n; i++) {
        printf("%d ", result[i]);
        }
}
```

Output:

```
Original Array: 11 13 21 3
Next Greater Elements: 13 21 -1 -1
```

LEARNING OUTCOMES;



	Learning outcome	rating	
1.	design	2	need practice
2.	use of DS	2	
3.	understanding of DS	2	
4.	debugging	2	

Best practice		rating	
1.	Design before coding	3	need practi
2.	usage of algorithm notation	3	
3	use of multi-file C program	3	
4.	Versioning	3	