DATA STRUCTURE

DAY 3 – 26/07/24

1.Array implementation of stack

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 100
typedef struct {
    int top;
    int capacity;
    int* array;
} Stack;
 Stack* createStack(int capacity) {
    Stack* stack = (Stack*)malloc(sizeof(Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (int*)malloc(capacity * sizeof(int));
    return stack;
}
 bool isEmpty(Stack* stack) {
    return stack->top == -1;
}
bool isFull(Stack* stack) {
    return stack->top == stack->capacity - 1;
}
 void push(Stack* stack, int item) {
    if (isFull(stack)) {
        printf("Stack overflow\n");
        return;
    }
    stack->array[++stack->top] = item;
}
 int pop(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow\n");
        exit(EXIT_FAILURE);
    }
    return stack->array[stack->top--];
}
 int peek(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        exit(EXIT_FAILURE);
    }
    return stack->array[stack->top];
}
 int size(Stack* stack) {
    return stack->top + 1;
}
 void printStack(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        return;
    }
    for (int i = 0; i <= stack->top; i++) {
        printf("%d ", stack->array[i]);
    }
    printf("\n");
```

```
}
 void freeStack(Stack* stack) {
    free(stack->array);
    free(stack);
}
 int main() {
    Stack* s = createStack(MAX);
    push(s, 10);
    push(s, 20);
    push(s, 30);
    printStack(s);
    printf("Popped: %d\n", pop(s));
    printf("Top: %d\n", peek(s));
    printf("Size: %d\n", size(s));
    printStack(s);
    freeStack(s);
    return 0;
}
```

OUTPUT:

```
Stack: 10 20 30
Popped:  30
Peek:  20
Size:  2
Printed stack: 10 20
```

2.Linked list of stack

PROGRAM:

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

```
#include <stdbool.h>


#define MAX 100
 typedef struct {
    int top;
    int capacity;
    int* array;
} Stack;
 Stack* createStack(int capacity) {
    Stack* stack = (Stack*)malloc(sizeof(Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (int*)malloc(capacity * sizeof(int));
    return stack;
}
 bool isEmpty(Stack* stack) {
    return stack->top == -1;
}
 bool isFull(Stack* stack) {
    return stack->top == stack->capacity - 1;
}
 void push(Stack* stack, int item) {
    if (isFull(stack)) {
        printf("Stack overflow\n");
        return;
    }
    stack->array[++stack->top] = item;
}
int pop(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow\n");
        exit(EXIT_FAILURE);
    }
    return stack->array[stack->top--];
}
 int peek(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        exit(EXIT_FAILURE);
    }
    return stack->array[stack->top];
}
 int size(Stack* stack) {
    return stack->top + 1;
}
 void printStack(Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        return;
    }
    for (int i = stack->top; i >= 0; i--) {
        printf("%d ", stack->array[i]);
    }
    printf("\n");
}
 void freeStack(Stack* stack) {
    free(stack->array);
    free(stack);
}
int main() {
    Stack* s = createStack(MAX);
    push(s, 10);
    push(s, 20);
    push(s, 30);
    printStack(s);
    printf("Popped: %d\n", pop(s));
    printf("Top: %d\n", peek(s));
    printf("Size: %d\n", size(s));
    printStack(s);
    freeStack(s);
    return 0;
}
```

OUTPUT:

```
Stack: 30 20 10
Popped: 30
Top: 20
Size: 2
Printed stack: 20 10
```