Data Structure

Task 3

إسلام محمد عطية محمد

سكشن 1

Stack with struct:

```
#include <iostream>
using namespace std;

enum Error_Code {
    underflow = -1,
    success = 0,
    overflow = 1
};

#define Max_Stack 100
#define StackEntry int


struct Stack {
    int top;
    StackEntry entry[Max_Stack];
};

void createStack(Stack *ps){
    ps->top = 0;
}

Error_Code push(Stack *ps, StackEntry e){
    if(ps->top==Max_Stack) return overflow;
    ps->entry[ps->top ++] = e;
    return success;
}

Error_Code pop(Stack *ps, StackEntry &e){
    if(ps->top==0) return underflow;
    e = ps->entry[--ps->top];
    return success;
}

void travarse(Stack *ps, void (*pf)
(StackEntry)){
    for (int i=ps->top-1; i>=0; --i){
        (*pf)(ps->entry[i]);
    }
}
```

```
Error_Code stackTop(Stack *ps, StackEntry
&e){
    if(ps->top==0) return underflow;
    e = ps->entry[ps->top-1];
    return success;
}

bool stackEmpty(Stack *ps){
    return !ps->top;
}

bool stackFull(Stack *ps){
    return ps->top == Max_Stack;
}

int stackSize(Stack *ps){
    return ps->top;
}

void clearStack(Stack *ps){
    ps->top = 0;
}

void display(StackEntry e){
    cout << "Item: " << e << endl;
}

int main()
{
        Stack s;
        createStack(&s);
        push(&s, 1);
        push(&s, 2);
        push(&s, 3);
        push(&s, 4);

        travarse(&s, &display);
}
```

Queue with struct:

```
#include <iostream>
using namespace std;

enum Error_Code {
    underflow = -1,
    success = 0,
    overflow = 1
};

#define Max_Queue 100
#define QueueEntry int

struct Queue
{
    int front, rear, size;
    QueueEntry entry[Max_Queue];
};

void createQueue(Queue *pq){
    pq->front = 0;
    pq->rear = -1;
    pq->size = 0;
}

Error_Code Append(Queue *pq,
QueueEntry e){
    if (pq->size==Max_Queue) return
overflow;
    pq->rear = (pq->rear+1) % Max_Queue;
    pq->entry[pq->rear] = e;
    pq->size ++;
    return success;
}

Error_Code Serve(Queue *pq, QueueEntry
&e){
    if(pq->size==0) return underflow;
    e = pq->entry[pq->front];
    pq->front = (pq->front-1) % Max_Queue;
    pq->size -- ;
    return success;
}
```

```
void travarse(Queue *pq, void (*pf)
(QueueEntry)){
    int pos = pq->front;
    for (int i=0; i!=pq->size; ++i){
        (*pf)(pq->entry[pos]);
        pos = (pos+1) % Max_Queue;
    }
}

int queueSize(Queue *pq){
    return pq->size;
}

void ClearQueue(Queue *pq){
    pq->front=0;
    pq->rear=-1;
    pq->size=0;
}

bool QueueEmpty(Queue *pq){
    return !pq->size;
}

bool QueueFull(Queue *pq){
    return (pq->size == Max_Queue);
}

void display(QueueEntry e){
    cout << "Item: " << e << endl;
}

int main()
{
    Queue q;
    QueueEntry ee;
    createQueue(&q);
    Append(&q, 1);
    Append(&q, 2);
    Append(&q, 3);
    Append(&q, 4);

    travarse( &q, &display);
}
```