**Stack with struct:**

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);

}

#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]);

}

}

**Queue with struct:**

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);

}

#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;

}