

Implementing Stacks and Queues in 'C'

1. Stack implementation using an array

We have seen in the lecture class how a stack can be implemented with push and pop functions to handle just integers. In this section we shall show how to handle a situation when we have to deal with integers as well as character information.

Let us consider a factory where goods of varying quality are being produced in batches. We assume that all goods produced in a particular batch have the same quality. As the items are being produced, we want to store the information regarding the number of items produced and the quality of that batch ('H' for high quality, 'G' for good quality, 'A' for average quality etc.) on a stack using PUSH function. To deliver the items to a client, we want to use the POP function. The POP function is supposed to remove the items from the stack. We are also interested to find out if the stack is full or empty at any point in time using the functions ISFULL and ISEMPY.

We start by declaring the following structure which has two arrays, one storing the number of items being produced in every batch, while the other storing the quality of the goods in that batch. The variable *top* indicates the position of the last item in an array.

The variable MAXSTACK defines the maximum size of the stack.

```
#define MAXSTACK 20
```

```
//Here is the structure declaration:
```

```
    struct arraystack {
        int items[MAXSTACK];
        char quality[MAXSTACK];
        int top;
    };
```

Calling the stack functions from the main program

```
int main( )
{
// declare other variables....
```

```

// Declare the name of the stack structure as "factory"
struct arraystack factory ;

        //initialize stack top to -1
factory.top = -1;

//jprod is number of items in a batch
//qlty is the quality of items in the batch ('H','G' etc)
. . . . .
//read from file corresponding value of jprod and qlty
//then use the following statement to store the data on
//the stack "factory"
.....
.....
// Typical calls to push and pop functions
push( factory, jprod, qlty);

. . . . .
. . . . .
result = isEmpty(factory);

. . . . .
. . . . .
pop(factory, &item, &qout );

// print qout and item

. . . . .

//Function definitions

//Precondition: Enter a valid pointer name which should not be NULL
//Postcondition: Returns 1 if stack is empty , otherwise returns 0.
int isEmpty( struct arraystack *produce )
{
    return ( produce-> top < 0 );
}

```

//Precondition:Enter a valid pointer name which should not be NULL

//Postcondition: Returns 1 if stack is full

```
int isFull( struct arraystack *produce )
{
    return ( produce->top >= MAXSTACK - 1 );
}
```

//Precondition: Enter a valid pointer name which should not be NULL along with item number and quality

//Postcondition: Pushes item number and its quality on the stack and updates the depth of stack

```
void push( struct arraystack *produce, int x, char q )
{
    if ( produce->top >= MAXSTACK - 1 )
    {
        printf( "\n% Stack is full.\n");
    }
    else {
        produce -> top = produce -> top + 1;
        produce -> items[ produce -> top ] = x;
        produce -> quality[ produce -> top ] = q;
    }
}
```

//Precondition:Enter a valid pointer name which should not be NULL along with xx that

//stores the integer type address of the variable item code and qq stores the char type

//address of the variable quality

//Postcondition: Pops item and its quality from the stack

```
void pop( struct arraystack *produce ,int *xx, char *qq)
{
    int x = 0;
    if ( produce->top < 0 )
    {
        printf("\nStack is empty");
    }
    else {
        *xx = produce -> items[ produce ->top ];
        *qq = produce -> quality[ produce ->top ];
        produce -> top = produce -> top - 1;
    }
}
```

2. Stack implementation using a linked list

A linked list can be used to implement a stack, if we use *top* to point to the first node of the linked list. Let us use the node structure

```
struct node {
    int data;
    struct node *next;
} ;
```

The push operation can be implemented by adding a node to the front of the linked list. Here is one implementation:

```
void PUSH ( struct node **top, int d)
{
    struct node * new;
    new = (struct node *)malloc(sizeof(struct node));
    new -> data = d;
    new -> link = *top;
    *top = new;
}
```

The pop operation can be implemented similarly. One has to take care of the empty stack. If the stack is empty, the function can be made to return a large negative integer.

3. Queue implementation using a linked list

A queue can be implemented using a linked list by making use of two pointers – *Front* pointing to the first node in the list, and *rear* pointing to the last node of the list.

To enqueue an element, the element can be put in a new node and the node attached to the node marked rear. One has to take care that the very first node of the linked list would be the marked both rear and front.

To dequeue an element, the node at the front needs to be deleted. This function would be very similar to the POP operation of a stack. If the queue contained a single element, then after dequeue, both rear and front pointers have to be set to NULL.