Implementation of stack using an array

In this section we shall show how stacks can be implemented using arrays. Consider a situation where goods of varying quality are being produced in a factory. Every batch of goods has 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.

Let us consider the following structure where we have two arrays serving as stacks for "items" and "quality". The variable *top* indicates the position of the last item in an array. The variable MAXSTACK defines the maximum size of the stack.
We use the 'arrow' symbol to indicate component of a structure that is being referenced through a pointer. Thus if *produce is a pointer to the structure in any function call then we can write

```
produce-> top  to denote   (*produce ).top
```

Let MAXSTACK denote the maximum size of the stack.

```
#define MAXSTACK 20
```

//Here is the structure declaration:

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

//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;
           }
}
```

Use of the stack functions from the main program

```
int main( )
{

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)
. . . . . .
. . . . . ..
push( factory, jprod, qlty);

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

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


. . . . . .
```