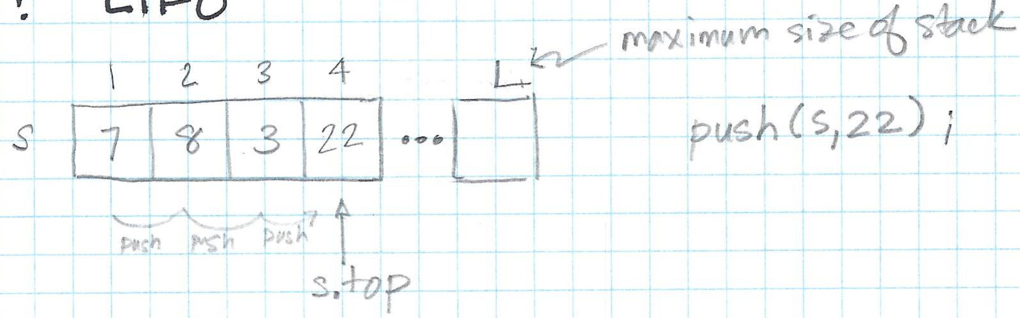
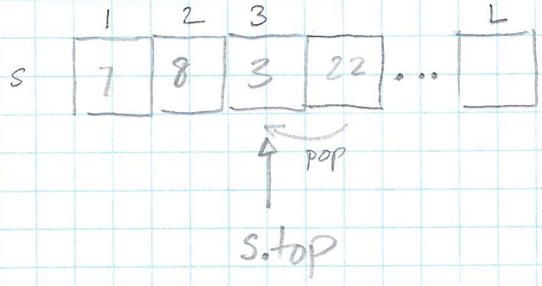
BASIC DATA STRUCTURES

- material here from Introduction to Algorithms, Third Edition, Cormen et al.

stack : LIFO

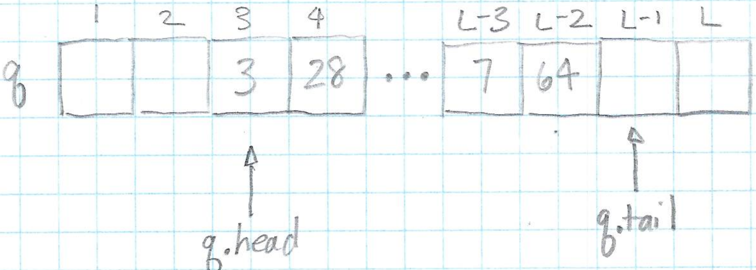


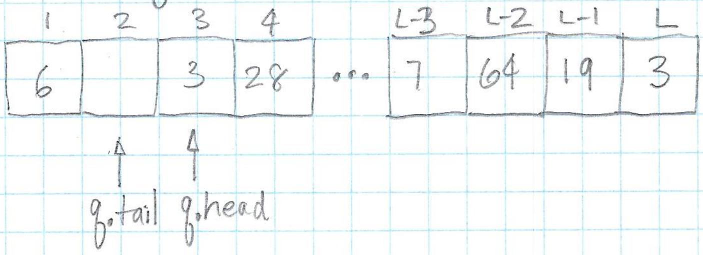
push(s,22);



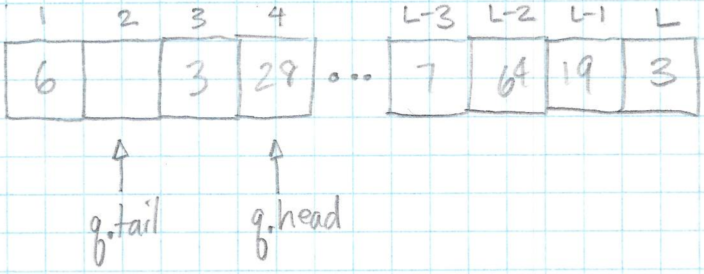
pop (s);
(returns 22)

queue : FIFO





enqueue(q,19);
enqueue (q,3);
enqueue (q,6);



dequeue(q);
(returns 3)

- stacks & queues are frequently used in embedded system software.

- these are basic data structures that can be realized using a number of techniques like arrays or pointers.

Algorithms : thinking through w/ pseudo code

Stacks

```
i)   stack_empty (s):
         if  s.top == 0
             return TRUE
         else return FALSE

ii)  push (s, x):
         s.top = s.top + 1
         s[s.top] = x

iii) pop (s):
     if  stack_empty (s)
         error "underflow"
     else
         s.top = s.top - 1
       return s[s.top + 1]
```

Queues

```
i)  enqueue (q, x):
       q[q.tail] = x
       if  q.tail == L
           q.tail = 1
       else
           q.tail = q.tail + 1
```

```
ii) dequeue (q):
       x = q[q.head]
       if q.head == L
          q.head = 1
       else
          q.head = q.head + 1
       return x
```

Implementation

- let's realize a stack along with empty-querying, pushing and popping in C

- in this example, our stack will store integers (int type)

```
#include <stdlib.h>

#define   L        1024   /* the number of integers
#define   TRUE     1U        we will store */
#define   FALSE    0U

int       s[L]   ;    /* stack declaration as
                         a global variable */

size_t    s_top = 0;  /* the stack pointer s.top */

typedef   unsigned short int   bool_t;


bool_t  stack_empty (void)
{
        if (s_top == 0)
        {
            return TRUE;
        }
        else
        {
            return FALSE;
        }
}

void    push (int x)
{

        ++s_top;

        s[s_top - 1] = x;

        return;
}
```

```
int  pop (void)
{
    if  stack-empty ()
    {
        printf ("underflow error \n");
        exit (EXIT_FAILURE);
    }
    else
    {
        return  s [s_top--];
    }
}

int  main ()
{
    /* inclass lab: write a program to illustrate
       use of the stack */

    int  loadarr[10] = {52, -29, 36, 1154, 72,
                        Ø  68, 44, 33, 59};

    /* load stack */

    size_t  i;

    for  (i=0; i!=10, ++i)
    {
        push (loadarr [i]);
    }

    /* pop stack */
    int  x;
    while  (stack-empty() == FALSE)
    {
        x = pop ();
        printf (" %d \n", x);
    }

    return Ø;
}
```

- keeping the stack "together" using a struct

```
struct   s_struct
{

      int   data[L];
     size_t top;
};
typedef struct s_struct stack_t;

stack_t   s;    /* declaration */
/* now, access via */
s.data[i]  }  or, if using
s.top++    }
                  stack_t  *sp; :

                  (sp->data)[i];
                  (sp->top)++;
```

using struct pointers is necessary in C if we wish to make changes to passed parameters in function calls

- please study the basic data structures starter code in the Git repository

- Homework: write a program, queue.c, which implements a queue along with the queue functions enqueue() and dequeue(), in the style of stack.c.