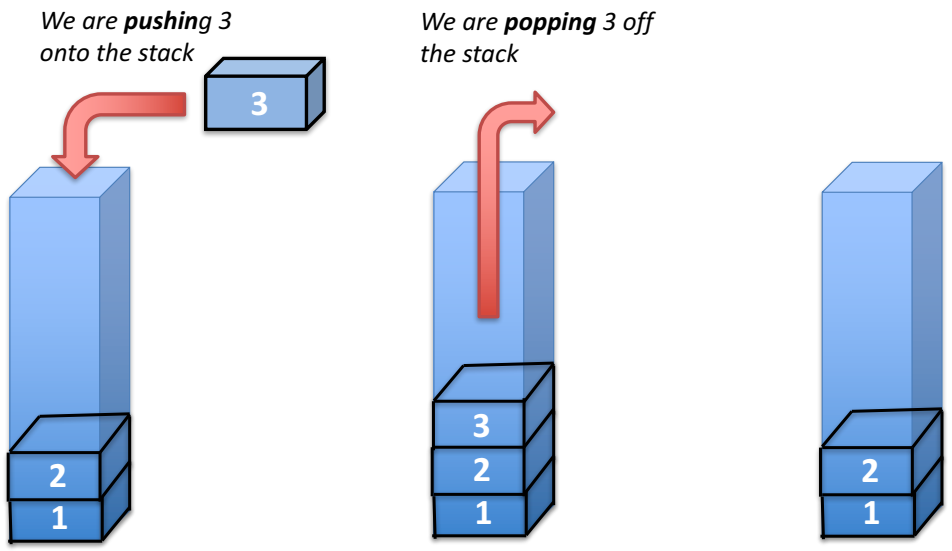
Stacks and Queues (+headerfiles, READMEs)

(see README on modules)

Stacks and Queues:

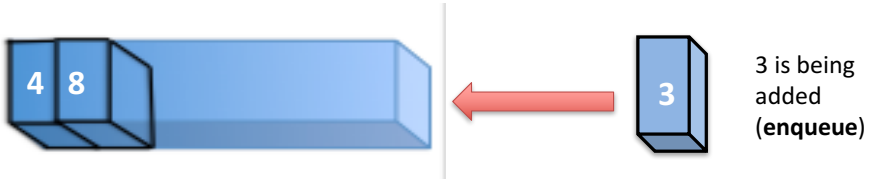
- These are data structures
- Like linked lists, they are not built into C…so we need to implement them
- Like linked lists, there are different ways to implement them

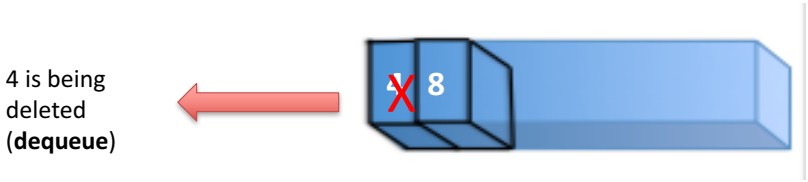
Stack:



-Insertion (push) and deletion (pop) are ONLY allowed from the top of the stack
-Last in, first out (LIFO) (since 3 was the last in, it is the first out)
-First in, last out (FILO) (since 1 was the first in, it is the last out)

Queue:





-Insertion (enqueue) is ONLY allowed from the end of the queue
-Deletion (dequeue) is ONLY allowed from the front of the queue
-First in, first out (FIFO) (since 4 was the first in, it is the first out)

- As with our other data structures, we have operations we can do to on them to achieve some goal
  - **Example: traversing (going through)**
    - We can traverse arrays by running a loop through each index of the array
    - We can traverse linked lists by running a loop until we hit NULL

Stacks and Queues

Stack Operations:

- Push
- Pop
- Peek
- Empty?
- Full?

(There can be more or less)

Queue Operations:

- Enqueue
- Dequeue
- Peek
- Empty?
- Full?

(There can be more or less)

Stacks and Queues

- WE implement these operations (you will see me do this today
- The implementation is up to the needs of the programmer
  - There are many different ways to implement
- The idea stays the same, the implementation changes

I will show you today a few different ways to implement a stack (there are other ways you could do it, just showing you a few ways):

Stack implementation 1 (using an array-very simple implementation):

```
computer$ gcc stack_array1.c
computer$ ./a.out
Element at top of the stack: 15
Elements:
15
22
11
19
15
13
Stack is full. false
Stack is empty. true
```

```c
#include <stdio.h>
#define MAXSIZE 8

int top = -1;  /*we will change this value when add or take away from the stack*/

int isempty() {
```

```
   if(top == -1)  /*true, the stack is empty (because we change the value of top from -1
if it has elements)*/
      return 1;
   else /*false, its not empty*/
      return 0;
}

int isfull(){

   if(top == MAXSIZE-1) /*true, the stack is full (can't add more elements)*/
      return 1;
   else
      return 0;
}

int peek(int stack[]) {
   return stack[top]; /*return the value at the top of the stack*/
}

/*get a value from the top of the stack*/
int pop(int stack[]) {
   int data;

   if(!isempty()) /*check first that the stack isn't empty*/
   {
      data = stack[top]; /*get value from top of stack*/
      top = top - 1;  /*decrement the value of top by 1*/
     }
   else
   {
      printf("Could not retrieve data, Stack is empty.\n");
      data=-1;

   }

   return data;
}

/*add a value to the stack*/
void push(int stack[],int data) {

   if(!isfull())  /*make sure stack isn't full first*/
   {
      top = top + 1;
      stack[top] = data;
   }
   else
   {
      printf("Could not insert data, Stack is full.\n");
   }
}

int main(int argc, char **argv) {
   /* push items on to the stack*/
```

```
   int stack[MAXSIZE];

   push(stack,13);
   push(stack,15);
   push(stack,19);
   push(stack,11);
   push(stack,22);
   push(stack,15);

   printf("Element at top of the stack: %d\n", peek(stack));
   printf("Elements: \n");

   /* print stack data-while stack isn't empty, keep loop going*/
   while(!isempty())
  {
      int data = pop(stack);
      printf("%d\n",data);
   }

     /*stack is empty after loop is done executing*/

     /*using the ternary operator*/
   printf("Stack is full. %s\n" , isfull(stack) ? "true" : "false"); /*isfull() returns
0, so prints out false*/
   printf("Stack is empty. %s\n" , isempty(stack) ? "true" :"false"); /*isempty() returns
1, so prints out true*/

   return 0;
}
```

Stack implementation 2 (using an array and struct):

computer$ gcc stack2.c
computer$ ./a.out

-Stack Options
---
1. Push
2. Pop
3. Display
4. Exit

Enter your choice:
1
Enter the element to be pushed: 33
Do you want to continue? y or n
y
---
1. Push

```
2. Pop
3. Display
4. Exit
----------
Enter your choice:
1
Enter the element to be pushed: 45
Do you want to continue? y or n
y
----------
1. Push
2. Pop
3. Display
4. Exit
----------
Enter your choice:
1
Enter the element to be pushed: 99
Do you want to continue? y or n
y
----------
1. Push
2. Pop
3. Display
4. Exit
----------
Enter your choice:
3

The status of the stack is
99
45
33

Do you want to continue? y or n
y
----------
1. Push
2. Pop
3. Display
4. Exit
----------
Enter your choice:
2
Popped element is = 99
Do you want to continue? y or n
y
----------
1. Push
2. Pop
3. Display
4. Exit
```

Enter your choice:
3

The status of the stack is
45
33

Do you want to continue? y or n
n

```c
#include <stdio.h>
#define MAXSIZE 5

struct stack /*containing all stack stuff (stack itself and top variable) in a struct*/
{
    int stk[MAXSIZE];
    int top;
};
typedef struct stack Stack;

void push (Stack *s)
{
    int num;
    if (s->top == (MAXSIZE - 1))
    {
        printf ("Stack is Full\n");
        return;
    }
    else
    {
        printf ("Enter the element to be pushed: ");
        scanf ("%d", &num);
        s->top = s->top + 1;
        s->stk[s->top] = num;
    }
    return;
}
/*  Function to delete an element from the stack */
int pop (Stack *s)
{
    int num;
    if (s->top == - 1)
    {
        printf ("Stack is Empty\n");
        return (s->top);
    }
    else
    {
        num = s->stk[s->top];
        printf ("Popped element is = %d\n", s->stk[s->top]);
        s->top = s->top - 1;
    }
    return(num);
```

```c
}

/*  Function to display the status of the stack */
void display (Stack *s)
{
    int i;
    if (s->top == -1)
    {
        printf ("Stack is empty\n");
        return;
    }
    else
    {
        printf ("\n The status of the stack is: \n");
        for (i = s->top; i >= 0; i--) /*start at current top value-keep going until print
last index 0*/
        {
            printf ("%d\n", s->stk[i]);
        }
    }
    printf ("\n");
}

int main (int argc, char **argv)
{
    int choice;
    Stack stack1;
    Stack *s=&stack1; /*need a pointer or else can't change values in function*/
    char option='a';
    stack1.top = -1; /*set top value to -1*/

    printf ("\n-Stack Options \n");
    while (option!='n')
    {
        printf ("-----------\n");
        printf ("1. Push\n");
        printf ("2. Pop \n");
        printf ("3. Display\n");
        printf ("4. Exit \n");
        printf ("-----------\n");

        printf ("Enter your choice:\n");
        scanf    ("%d", &choice);

        if(choice==1)
        {
            push(s);
        }

        else if(choice==2)
        {
            pop(s);
        }
```

```
        else if(choice==3)
        {
            display(s);
        }

        else /*exit*/
        {
            break;
        }

        printf ("Do you want to continue? y or n\n");
        scanf (" %c", &option);
    }
}
```

Stack implementation 3 (using a linked list):

computer$ gcc stack3.c
computer$ ./stack
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack
Top element is 20

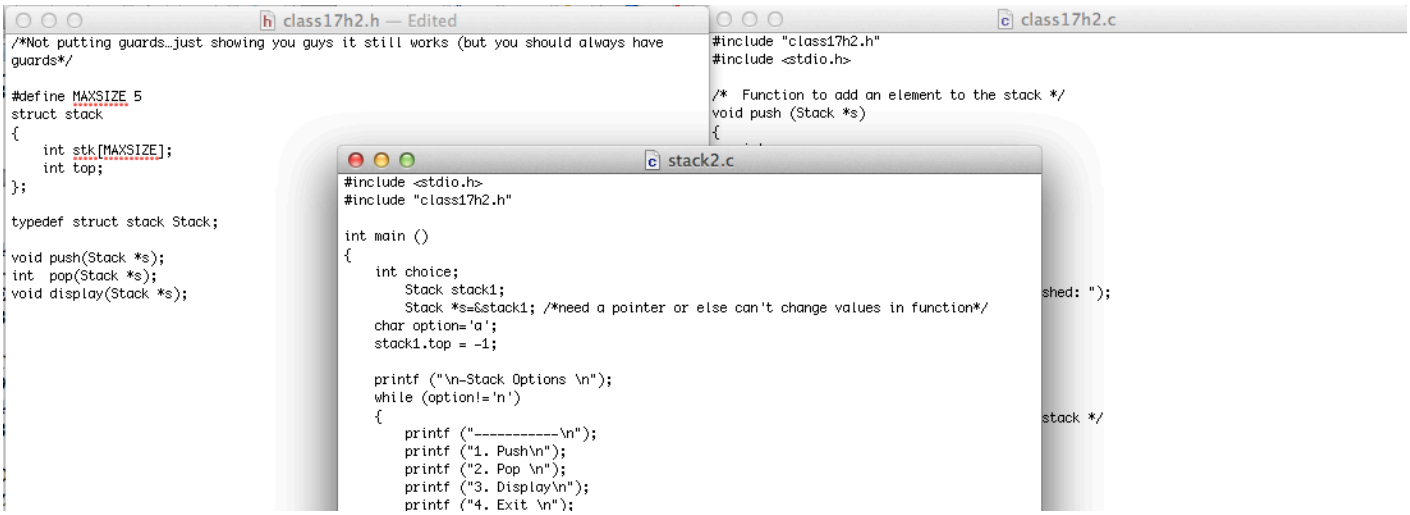
```
#include <stdio.h>
#include <stdlib.h>

struct stack_node
{
    int data;
    struct stack_node* next;
};


int isEmpty(struct stack_node *root)
{
    return !root;
}

void push(struct stack_node** current, int data)
{
    struct stack_node* stack_node = malloc(sizeof(struct stack_node)); /*should make sure
not null*/
    stack_node->data = data;
    stack_node->next = *current;
```

```
    *current = stack_node;
    printf("%d pushed to stack\n", data);
}

int pop(struct stack_node** current)
{
    if (isEmpty(*current))
    {
        return -1;
    }
    struct stack_node* temp = *current;
    *current = (*current)->next;
    int popped = temp->data;
    free(temp);

    return popped;
}

int peek(struct stack_node* root)
{
    if (isEmpty(root))
    {
        return -1;
    }
    return root->data;
}

int main(int argc, char **argv)
{
    struct stack_node* root = NULL;

    push(&root, 10);
    push(&root, 20);
    push(&root, 30);

    printf("%d popped from stack\n", pop(&root));
    printf("Top element is %d\n", peek(root));

    return 0;
}
```

Headerfiles

Up until now, all of the functions or structs we've been creating have been kept in the same file. We've been using other functions, like printf, that are contained in other files by using #include for some header files.

Let's start doing the same thing-we'll start keeping our functions in other files and including headerfiles to access them. For example, let's put all of our stack array operations in a header file:



We will now have 3 different files: the file with our main (nothing else) then a header file (.h) and the definition file (.c)..notice they have the same name

computer$ gcc stack2.c class17h2.c -o stack
computer$ ./stack

stack_array_main.c (notice we have our main alone in this file, and including a header)

```
#include <stdio.h>
#include "class17h2.h"  /* https://docs.microsoft.com/en-us/cpp/preprocessor/hash-include-directive-c-cpp?view=msvc-170  see this for difference between " " and < >*/

int main (int argc, char **argv)
{
   int choice;
   Stack stack1;
   Stack *s=&stack1; /*need a pointer or else can't change values in function*/
   char option='a';
   stack1.top = -1; /*set top value to -1*/

   printf ("\n-Stack Options \n");
   while (option!='n')
   {
      printf ("-----------\n");
      printf ("1. Push\n");
      printf ("2. Pop \n");
      printf ("3. Display\n");
      printf ("4. Exit \n");
      printf ("-----------\n");

      printf ("Enter your choice:\n");
      scanf   ("%d", &choice);

            if(choice==1)
            {
                  push(s);
            }

            else if(choice==2)
```

```
            {
                    pop(s);
            }

            else if(choice==3)
            {
                    display(s);
            }

            else /*exit*/
            {
                    break;
            }

      printf ("Do you want to continue? y or n\n");
      getchar();
      scanf ("%c", &option);
   }
}
```

class17h2.h (header file-notice only function declarations, not definitions)

```
#ifndef CLASS17_H3   /*see below for explanation of header guards*/
#define CLASS17_H3

#define MAXSIZE 5
struct stack /*containing all stack stuff (stack itself and top variable) in a struct*/
{
   int stk[MAXSIZE];
   int top;
};
typedef struct stack Stack;

void push(Stack *s);
int  pop(Stack *s);
void display(Stack *s);

#endif
```

class17h2.c (defining what is in header file)

```
#include "class17h2.h"
#include <stdio.h>

/*  Function to add an element to the stack */
void push (Stack *s)
{
   int num;
   if (s->top == (MAXSIZE - 1))
   {
      printf ("Stack is Full\n");
      return;
```

```
    }
    else
    {
        printf ("Enter the element to be pushed: ");
        scanf ("%d", &num);
        s->top = s->top + 1;
        s->stk[s->top] = num;
    }
    return;
}
/*  Function to delete an element from the stack */
int pop (Stack *s)
{
    int num;
    if (s->top == - 1)
    {
        printf ("Stack is Empty\n");
        return (s->top);
    }
    else
    {
        num = s->stk[s->top];
        printf ("Popped element is = %d\n", s->stk[s->top]);
        s->top = s->top - 1;
    }
    return(num);
}

/*  Function to display the status of the stack */
void display (Stack *s)
{
    int i;
    if (s->top == -1)
    {
        printf ("Stack is empty\n");
        return;
    }
    else
    {
        printf ("\n The status of the stack is: \n");
        for (i = s->top; i >= 0; i--) /*start at current top value-keep going until print last index 0*/
        {
            printf ("%d\n", s->stk[i]);
        }
    }
    printf ("\n");
}
```

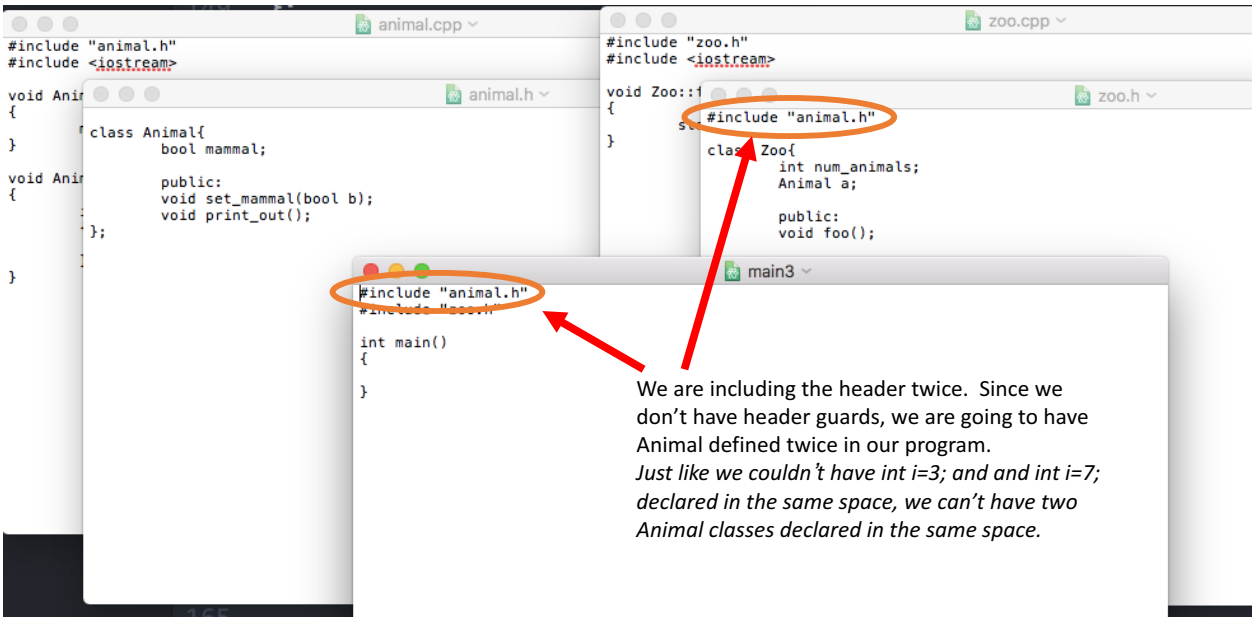
Header guards:
If you notice in the .h header file above, I have the following at the beginning and end of my file:

#ifndef CLASS17_H3
#define CLASS17_H3

#endif

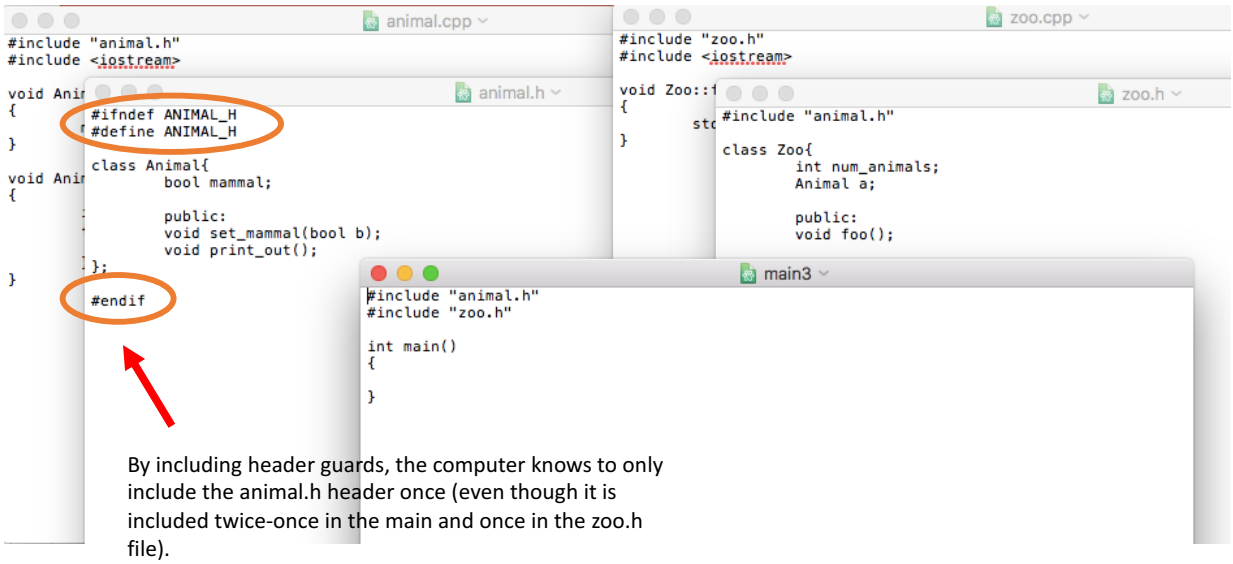
What are these? And why will our code sometimes work even if we don't have them? These are our header guards (technically, we are using preprocessor directives as header guards: https://docs.microsoft.com/en-us/cpp/preprocessor/preprocessor-directives?view=msvc-170)

When making your header files (.h) you should always include header guards (see .h example below). By including them, we ensure we do not define something twice.



Errors when you don't use header guards (this is in C++, but the concept is the same):

```
computer$ g++ -std=c++14 main3.cpp animal.cpp zoo.cpp
In file included from main3.cpp:2:
In file included from ./zoo.h:1:
./animal.h:2:7: error: redefinition of 'Animal'
class Animal{
      ^
main3.cpp:1:10:       './animal.h' included multiple times, additional include site here
#include "animal.h"
         ^
./zoo.h:1:10:       './animal.h' included multiple times, additional include site here
#include "animal.h"
         ^
./animal.h:2:7:       unguarded header; consider using #ifdef guards or #pragma once
class Animal{
      ^
1 error generated.
```



How does this work? The header guards give a specific "name" to the header file ("name" is ANIMAL_H). If we have already included it once and try to include it again, we check first to make sure the "name" ANIMAL_H hasn't already been included before.

For example, if the first time we try to include *animal.h* is in the zoo.h file, we go ahead and include the class Animal in our whole program. If we try to include *animal.h* again (in the *main3.cpp* file), we check first that our "name" ANIMAL_H hasn't already been used-since it has been used, we know not to include it a second time.

Abstraction:
By writing our code in this way, we can achieve a form of abstraction. For example, we have written a function called pop(). When the user uses this function, they don't know or care how it's implemented-they are just using it in their code. You can go and redefine it another way in the .c file with the defintions, but this won't affect the code that was already using the function…all the programs the user had written before with pop(), where they include the header file holding the declaration of pop() will work the same.

Notice below, are we not dealing with the implementations of the functions (push, pop etc. kept in the headers we created above). We are just using them and we can focus on the problem.

Real Problem 1: Stack-Array

Write a program that allows the user to enter in book prices and then print the prices out to screen.

- I will use a stack to hold my info
- I will use an array implementation (variation with struct)

computer$ gcc books.c class17h2.c
books.c: In function 'main': /*getting an error since im doing int i=0 in my for loop*/

books.c:17: error: ‘for’ loop initial declaration used outside C99 mode
computer$ gcc **-std=c99** books.c class17h2.c /*changing the “dialect” to C99 using –std=99*/
computer$ ./a.out
/*Since I’m using the functions from last time, notice I’m really having to stretch them to make them work for the program I’m trying to make. This should show you the importance of making your functions as general as possible. For your next HW you will need to modify the stack/queue functions to have them fit your needs (or just make completely new ones if you would prefer)- otherwise you will end up with this same issue*/

How many books would you like to enter? 4
Book Price-Enter the element to be pushed: 7
Book Price-Enter the element to be pushed: 3
Book Price-Enter the element to be pushed: 4
Book Price-Enter the element to be pushed: 4

All books (in dollars):
The status of the stack is
4
4
3
7

```
#include <stdio.h>
#include "class17h2.h"

int main(void)
{

 Stack prices; /*makes a struct (kept in the header)*/
 prices.top=-1; /*empty stack signaled by -1*/
 int p=10;

 while(MAXSIZE<p)
 {
        printf("\nHow many books would you like to enter? ");
        scanf("%d", &p);
 }

 for(int i=0;i<p;i++)
 {
        printf("Book Price-");
        push (&prices);
 }

 printf("\nAll books (in dollars):");
 display(&prices);
}
```

Real Problem 2: Stack-Linked List

Five brothers own a deli shop in New Jersey called Jersey Subs. The number of sandwiches each brother sells a day is kept in a text file. Create a program that calculates the total number of sandwiches sold.

- I will use a stack to hold my info
- I will use a linked list implementation

```
computer$ gcc jersey_subs.c class17h3.c
computer$ ./a.out sandwich.txt /*format of sandwich.txt: name, num_sandwiches*/

Sandwiches Vinny sold: 21  --21 pushed to stack
Sandwiches Giani sold: 25  --25 pushed to stack
Sandwiches Pauly sold: 21  --21 pushed to stack
Sandwiches Mikey sold: 22  --22 pushed to stack
Sandwiches Ronni sold: 33  --33 pushed to stack

Popping from the stack:
33...22...21...25...21...

Total sandwiches sold: 122

-------
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "class17h3.h"


int main(int argc, char **argv)
{
 struct StackNode* root=NULL;
 FILE *fp=fopen(argv[1],"r");
 char *line=malloc(30);
 char *token;
 int n=0, total=0;

 /*read file info into stack*/

 printf("\n");
 while(fgets(line,30,fp)!=NULL)
 {
        token=strtok(line, ",");
        printf("Sandwiches %s sold:", token);
        token=strtok(NULL, "\n");
        n=atoi(token);
        printf(" %d  --", n);
        push(&root, n);
 }
```

```
 printf("\nPopping from the stack:\n");
 while(!isEmpty(root))
 {
        n=pop(&root);
        printf("%d…",n);
        total+=n;
 }

 printf("\n\nTotal sandwiches sold: %d\n", total);
}
```