COMP1911 22T2 (https://webcms3.cse.unsw.edu.au/COMP1911/22T2) **Code Examples from Lectures on 14-9_stacksAndQueues** Introduction to Programming (https://webcms3.cse.unsw.edu.au/COMP1911/22T2)

stackDemoOneFile.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/14-9_stacksAndQueues/code/stackDemoOneFile.c)

An example of implementing and using a stack of ints

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

#define MAX_SIZE 100

typedef struct stack * Stack;
struct stack{
    int items[MAX_SIZE];
    int size;
};

Stack stackCreate(void);
void stackPush(Stack s, int item);
int stackTop(Stack s);
int stackPop(Stack s);
int stackSize(Stack s);
void destroyStack(Stack s);

int main(int argc, char * argv[]){
    Stack s;
    s = stackCreate();
    stackPush(s, 10);
    stackPush(s, 11);
    stackPush(s, 12);
    printf("%d\n", stackSize(s)); // prints 3
    printf("%d\n", stackTop(s)); // prints 12
    printf("%d\n", stackPop(s)); // prints 12
    printf("%d\n", stackPop(s)); // prints 11
    printf("%d\n", stackPop(s)); // prints 10

    return EXIT_SUCCESS;
}


Stack stackCreate(void){
    Stack s = malloc(sizeof(struct stack));
    s->size = 0;
    return s;
}
void stackPush(Stack s, int item){
    if(s->size == MAX_SIZE){
        fprintf(stderr,"Stack is full: Can't push\n");
        stackDestroy(s);
        exit(EXIT_FAILURE);
    }
    s->items[s->size] = item;
    s->size++;
}
int stackTop(Stack s){
     if(s->size == 0){
        fprintf(stderr,"Stack is empty: No top\n");
        stackDestroy(s);
        exit(EXIT_FAILURE);
     }
     return s->items[s->size-1];

}

int stackPop(Stack s){
     if(s->size == 0){
        fprintf(stderr,"Stack is empty: Can't pop\n");
        stackDestroy(s);
        exit(EXIT_FAILURE);
     }
     int i =  s->items[s->size-1];
     s->size--;
     return i;
}

int stackSize(Stack s){
    return s->size;
}

void destroyStack(Stack s){
    free(s);
}
```

stackDemo.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/14-9_stacksAndQueues/code/stackDemo.c)

An example of splitting a program into more than 1 file. The main function is in this file, all stack related prototyes and typedefs are in

Stack.h and all code for the stack is in Stack.c

To compile: dcc -o stackDemo stackDemo.c Stack.c

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


int main(int argc, char * argv[]){
    Stack s;
    s = stackCreate();
    stackPush(s, 10);
    stackPush(s, 11);
    stackPush(s, 12);
    printf("%d\n", stackSize(s)); // prints 3
    printf("%d\n", stackTop(s)); // prints 12
    printf("%d\n", stackPop(s)); // prints 12
    printf("%d\n", stackPop(s)); // prints 11
    printf("%d\n", stackPop(s)); // prints 10
    printf("%d\n", stackPop(s)); // SHould fail
    stackDestroy(s);
    return EXIT_SUCCESS;
}
```

Stack.h (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/14-9_stacksAndQueues/code/Stack.h)

define MAX_SIZE 100

```
typedef struct stack Stack;

Stack *stackCreate(void);
void stackPush(Stack *s, int item);
int stackTop(Stack *s);
int stackPop(Stack *s);
int stackSize(Stack *s);
void stackDestroy(Stack *s);
```

Stack.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/14-9_stacksAndQueues/code/Stack.c)

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

struct stack {
    int items[MAX_SIZE];
    int size;
};


Stack *stackCreate(void) {
    Stack *s = malloc(sizeof (struct stack));
    if(s == NULL) {
        fprintf(stderr,"Insufficient Memory\n");
        exit(EXIT_FAILURE);
    }
    s->size = 0;
    return s;
}

void stackPush(Stack *s, int item) {
    int index = s->size;
    if(s->size < MAX_SIZE) {
        s->items[index] = item;
        s->size++;
    } else {
        fprintf(stderr,"Stack full\n");
        stackDestroy(s);
        exit(EXIT_FAILURE);
    }
}

void stackDestroy(Stack *s) {
    free(s);
}

int stackSize(Stack *s) {
    return s->size;
}

int stackTop(Stack *s) {
   if(s->size == 0) {
       fprintf(stderr,"Stack empty\n");
       stackDestroy(s);
       exit(EXIT_FAILURE);
   }
   int topIndex = s->size -1;
   int topItem = s->items[topIndex];
   return topItem;
}

int stackPop(Stack *s) {
   if(s->size == 0) {
       fprintf(stderr,"Stack empty\n");
       stackDestroy(s);
       exit(EXIT_FAILURE);
   }
   int topIndex = s->size -1;
   int topItem = s->items[topIndex];
   s->size--;
   return topItem;
}
```

postfixCalculator.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/14-9_stacksAndQueues/code/postfixCalculator.c)

Implements a postfix calculator (see lecture notes to see what that is) using a Stack

To compile: dcc -o postfixCalculator postfixCalculator.c Stack.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include "Stack.h"

#define MAX_LEN 100
#define FALSE 0
#define TRUE 1

int isOperator(char c);

int main(int argc, char * argv[]){
    char expression[MAX_LEN];
    Stack s = stackCreate();

    if(fgets(expression,MAX_LEN,stdin) != NULL){
        int i = 0;
        while(expression[i] != '\0' && expression[i] != '\n'){
            if(isdigit(expression[i])){
                int num = atoi(&expression[i]);
                while(isdigit(expression[i])){
                    i++;
                }
                stackPush(s,num);
            } else if(isOperator(expression[i])){
                if(stackSize(s) < 2){
                    fprintf(stderr,"Illegal Expression 1\n");
                    stackDestroy(s);
                    return EXIT_FAILURE;
                }
                int num1 = stackPop(s);
                int num2 = stackPop(s);
                int result = 0;
                if(expression[i] == '+'){
                    result = num1+num2;
                } else if (expression[i] == '*'){
                    result = num1*num2;
                } else if (expression[i] == '-'){
                    result = num2-num1;
                } else if (expression[i] == '/'){
                    result = num2/num1;
                } else {
                    fprintf(stderr,"Illegal Expression 2\n");
                    stackDestroy(s);
                    return EXIT_FAILURE;
                }

                stackPush(s,result);
                i++;
            } else {
                i++;
            }
        }
        if(stackSize(s) == 1){
            int answer = stackPop(s);
            printf("%d\n",answer);
        } else {
            fprintf(stderr,"Illegal Expression 3\n");
            stackDestroy(s);
            return EXIT_FAILURE;
        }
    }

    stackDestroy(s);
    return EXIT_SUCCESS;
}

int isOperator(char c){
    if(c == '+' || c == '-' || c == '*' || c == '/'){
        return TRUE;
    } else {
        return FALSE;
    }
}
```

for.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/14-9_stacksAndQueues/code/for.c)

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

int main(void) {
    int sum = 1;
    for (int j = 0; j < 10; j++) {
        printf("%d ", sum);
        sum *= 2;
    }
    printf("\n");
}
```

intQueue.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/14-9_stacksAndQueues/code/intQueue.c)

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

#define QUEUE_SIZE 5

typedef struct queueStruct Queue;

struct queueStruct {
    int queue[QUEUE_SIZE];
    int top;
    int bot;
};

void enqueue(Queue *q, int n);
int dequeue(Queue *q);
Queue *newQueue(void);
void freeQueue(Queue *q);

int main(int argc, char* argv[]) {
    int i = 0;
    int temp;

    Queue *q = newQueue();

    while (i < QUEUE_SIZE) {
        scanf("%d", &temp);
        enqueue(q, temp);
        i++;
    }

    printf("All read in! Time to print\n");

    i = 0;
    while (i < QUEUE_SIZE) {
        printf("%d\n", dequeue(q));
        i++;
    }

    freeQueue(q);
    q = NULL;

}

void enqueue(Queue *q, int n) {
    int top = q->top;
    q->queue[top] = n;
    q->top++;
}

int dequeue(Queue *q) {
    int bot = q->bot;
    q->bot++;
    return q->queue[bot];
}

Queue *newQueue() {
    Queue *q = malloc(sizeof(Queue));
    q->top = 0;
    q->bot = 0;
    return q;
}

void freeQueue(Queue *q) {
    free(q);
}
```

intStack.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/14-9_stacksAndQueues/code/intStack.c)

```
#include "intStack.h"

struct stackStruct {
    int stack[STACK_SIZE];
    int top;
};

void stackPush(Stack *s, int n) {
    int top = s->top;
    s->stack[top] = n;
    s->top++;
}

int stackPop(Stack *s) {
    s->top--;
    return s->stack[s->top];
}

Stack *stackCreate() {
    Stack *s = malloc(sizeof(Stack));
    s->top = 0;
    return s;
}

void stackDestroy(Stack *s) {
    free(s);
}

int stackSize(Stack *s) {
    return s->top;
}
```

testQueue.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/14-9_stacksAndQueues/code/testQueue.c)

```
#include <stdio.h>

#define QUEUE_SIZE 5
#define MAX_STRING_LENGTH 100

//void enqueue(char *s);
//char *dequeue(void);

int main(int argc, char* argv[]) {
    int i = 0;

    char queue[QUEUE_SIZE][MAX_STRING_LENGTH];

    while (i < QUEUE_SIZE) {
        scanf("%s", queue[i]);
        i++;
    }

    printf("All read in! Time to print\n");

    i = 0;
    while (i < QUEUE_SIZE) {
        printf("%s\n", queue[i]);
        i++;
    }

}
```

useStack.c (https://cgi.cse.unsw.edu.au/~cs1911/22T2/lec/14-9_stacksAndQueues/code/useStack.c)

```
#include "intStack.h"

int main(int argc, char* argv[]) {
    int i = 0;
    int temp;

    Stack *s = stackCreate();

    while (i < STACK_SIZE) {
        scanf("%d", &temp);
        stackPush(s, temp);
        i++;
    }

    printf("All read in! Time to print\n");

    i = 0;
    while (i < STACK_SIZE) {
        printf("%d\n", stackPop(s));
        i++;
    }

    stackDestroy(s);
    s = NULL;

}
```