COMP1511 18s2 (webcms)

Code Examples from Lectures on stacks_and_queues

stack_example.c

```
Three implementations of a Stack Abstract Data Type
https://en.wikipedia.org/wiki/Abstract_data_type


$ dcc stack_example.c  stack_list.c -o stack_example
$ a.out
3
12
12
11
10
0
$ dcc stack_example.c  stack_array.c -o stack_example
$ a.out
3
12
12
11
10
0
$ dcc stack_example.c  stack_realloc.c -o stack_example
$ a.out
3
12
12
11
10
0
```

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

int main(void) {
    stack s;

    s = stack_create();

    stack_push(s, 10);
    stack_push(s, 11);
    stack_push(s, 12);

    printf("%d\n", stack_size(s)); // prints 3

    printf("%d\n", stack_top(s)); // prints 12

    printf("%d\n", stack_pop(s)); // prints 12
    printf("%d\n", stack_pop(s)); // prints 11
    printf("%d\n", stack_pop(s)); // prints 10

    printf("%d\n", stack_size(s)); // prints 0

    return 0;
}
```

stack.h

```
/*
Stack Abstract Data Type
https://en.wikipedia.org/wiki/Abstract_data_type

Actual implementation of stack opaque to (hidden from) user
*/

typedef struct stack_internals *stack;

stack stack_create(void);           // create a new stack
void stack_free(stack s);           // free a stack
void stack_push(stack s, int item); // add new item to stack
int stack_pop(stack s);             // remove top item from stack and return it
int stack_is_empty(stack s);        // return true if stack is empty
int stack_top(stack s);             // return top item from stack but don't remove it
int stack_size(stack s);            // return number elements in stack
```

stack_array.c

```
Implementation of stack Abstract Data Type with an array
```

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

#define MAX 2048

struct stack_internals {
    int elements[MAX];
    int top;
};

// create a new stack
stack stack_create(void) {
    stack s = malloc(sizeof *s);
    if (s == NULL) {
        fprintf(stderr, "Out of memory\n");
        exit(1);
    }
    s->top = 0;
    return s;
}

// add new item to stack
void stack_push(stack s, int item) {
    if (s->top == MAX) {
        fprintf(stderr, "push() exceeds maximum stack size %d\n", MAX);
        exit(1);
    }
    s->elements[s->top] = item;
    s->top = s->top + 1;
}

// remove top item from stack and return it
int stack_pop(stack s) {
    if (stack_is_empty(s)) {
        fprintf(stderr, "pop() of empty stack\n");
        exit(1);
    }
    s->top = s->top - 1;
    return s->elements[s->top];
}
```

stack_list.c

```
Implementation of stack Abstract Data Type with a linked list
```

Implementation of stack ADT with a linked list

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

struct stack_internals {
    struct node   *top;
    int            size;
};

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

// create a new stack
stack stack_create(void) {
    stack s = malloc(sizeof *s);
    if (s == NULL) {
        fprintf(stderr, "Out of memory\n");
        exit(1);
    }
    s->top = NULL;
    s->size = 0;
    return s;
}

// add new item to stack
void stack_push(stack s, int item) {
    struct node *n = malloc(sizeof *n);
    n->data = item;
    n->next  = s->top;
    s->top = n;
    s->size = s->size + 1;
}

// remove top item from stack and return it
int stack_pop(stack s) {
    int i;
    struct node *n;

    if (stack_is_empty(s)) {
        fprintf(stderr, "pop() of empty stack\n");
```

[stack_realloc.c](#)

```
Implementation of stack Abstract data Type with  malloc/realloc
```

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

#define INITIAL_STACK_SIZE 1024

struct stack_internals {
    int *elements;
    int size;
    int top;
};

// create a new stack
stack stack_create(void) {
    stack s = malloc(sizeof *s);
    if (s == NULL) {
        fprintf(stderr, "Out of memory\n");
        exit(1);
    }
    s->size = INITIAL_STACK_SIZE;
    s->elements = malloc(s->size * sizeof s->elements[0]);
    if (s->elements == NULL) {
        fprintf(stderr, "Out of memory\n");
        exit(1);
    }
    s->top = 0;
    return s;
}

// add new item to stack
void stack_push(stack s, int item) {
    if (s->top == s->size) {
        s->size = s->size * 2;
        s->elements = realloc(s->elements, s->size * sizeof s->elements[0]);
        if (s->elements == NULL) {
            fprintf(stderr, "Out of memory\n");
            exit(1);
        }
    }
    s->elements[s->top] = item;
    s->top = s->top + 1;
}
```

brackets.c

```
Read stdin checking   { [ ( left brackets match ) ] } right brackets
Other characters ignored.


$ dcc brackets.c stack_list.c -o brackets   # or stack_array.c or stack_realloc.c
$ ./brackets
[ {{ ( ) } ]

Error: bracket mismatch '{' versus ']'
```

```c
#include <stdio.h>

#include "stack.h"


int main(void) {

    stack s = stack_create();

    int ch = getchar();
    while (ch != EOF) {

        if (ch == '{' || ch == '(' || ch == '[') {
            stack_push(s, ch);
        } else if (ch == '}' || ch == ')' || ch == ']') {
            if (stack_is_empty(s)) {
                printf("Error: unbalanced '%c'\n", ch);
                return 1;
            }

            int left_bracket = stack_pop(s);
            if (
                (ch == '}' && left_bracket != '{') ||
                (ch == ')' && left_bracket != '(') ||
                (ch == ']' && left_bracket != '[')
                ) {
                printf("Error: bracket mismatch '%c' versus '%c'\n", left_bracket, ch);
                return 1;
            }
        }

        ch = getchar();
    }

    if (!stack_is_empty(s)) {
        printf("Error: unbalanced '%c'\n", stack_pop(s));
        return 1;
    }

    return 0;
}
```

postfix.c

```
Reverse Polish notation calculator using stack ADT


https://en.wikipedia.org/wiki/Reverse_Polish_notation


$ dcc postfix.c stack_list.c -o postfix   # or stack_array.c or stack_realloc.c
$ ./postfix
15 7 1 1 + - / 3 * 2 1 1 + + -

Result: 5
```

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

#include "stack.h"

#define MAX_LINE 4096
int main(void) {

    char line[4096];
    while (fgets(line, sizeof line, stdin) != NULL) {
        stack s = stack_create();

        int i = 0;
        while (line[i] != '\0') {
            int ch = line[i];

            if (isdigit(ch)) {
                // convert chars to num push number onto stack
                // and skip over digits

                int num = atoi(&line[i]);

                stack_push(s, num);

                while (isdigit(line[i])) {
                    i = i + 1;
                }
            } else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {

                // pop 2 values from stack
                // calculate result
                // push result onto stack

                int a = stack_pop(s);
                int b = stack_pop(s);

                int result;
                if (ch == '+') {
                    result = b + a;
                } else if (ch == '-') {
                    result = b - a;
```

queue_example.c

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

#include "queue.h"

int main(void) {
    queue q;

    q = queue_create();
    queue_enqueue(q, 10);
    queue_enqueue(q, 11);
    queue_enqueue(q, 12);

    printf("%d\n", queue_size(q)); // prints 3
    printf("%d\n", queue_front(q)); // prints 10
    printf("%d\n", queue_dequeue(q)); // prints 10
    printf("%d\n", queue_dequeue(q)); // prints 11
    printf("%d\n", queue_dequeue(q)); // prints 12

    return 0;
}
```

queue.h

```
/*
Queue Abstract Data Type

https://en.wikipedia.org/wiki/Abstract_data_type

Actual implementation of queue opaque to (hidden from) user
*/

typedef struct queue_internals *queue;

queue queue_create(void);                  // create a new queue
void queue_free(queue q);                  // free a queue
void queue_enqueue(queue q, int item); // add new item to queue
int queue_dequeue(queue q);                // remove next item from queue and return it
int queue_is_empty(queue q);               // return true if queue is empty
int queue_front(queue q);                  // return next item from queue but don't remove it
int queue_size(queue q);                   // return number elements in queue
```

queue_list.c

Implementation of stack ADT with a linked list
completion left as an exercise

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

struct queue_internals {
};

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

// create a new queue
queue queue_create(void) {
    return NULL;
}

// add new item to queue
void queue_enqueue(queue queue, int item) {
}

// remove top item from queue and return it
int queue_dequeue(queue queue) {
    return 0;
}

// return true if queue is empty
int queue_is_empty(queue queue) {
    return 0;
}

// return top item from queue but don't remove it
int queue_front(queue queue) {
    return 0;
}

// return number elements in queue
int queue_size(queue queue) {
    return 0;
}

// free a queue
```