Lab 7

Demo and Helpful Tips
List.h -> 2 structures

// List element: a list is a chain of these
typedef struct element
{
    int val;
    struct element* next;
} element_t;

// List header - keep track of the first and last list elements
typedef struct list
{
    element_t* head;
    element_t* tail;
} list_t;

struct list

List ptr

head

tail

val

next

3 x struct element

cell

cell

cell

val

next

val

next

val

next

head

tail

element_t

list_t
List.h -> 7 functions

// returns a pointer to a new header for an empty list, or NULL if memory allocation fails.
list_t* list_create( void );

// frees all the memory used by the list
void list_destroy( list_t* list );

// returns a pointer to a new list element containing integer i and next-pointer set to NULL, or NULL if memory allocation fails.
element_t* element_create( int i );

// Appends a new element containing integer i to the end of the list. Returns 0 on success, else 1.
int list_append( list_t* list, int i );

// Prepends a new element containing integer i to the head of the list. Returns 0 on success, else 1.
int list_prepend( list_t* list, int i );

// Returns a pointer to the ith list element, where the list head is 0, head->next is 1, etc., or NULL if i is out of range (i.e. larger than (number of list elements -1 ))
element_t* list_index( list_t* list, unsigned int i );

// Prints a list in human-readable form from the first to last elements, between curly braces.
void list_print( list_t* list );
The idea ...

- We are given 5 different buggy implementations of the functions described in List.h
- We are given a test driver: main.c
- We are to discover these bugs by adding code to our test driver such that ...
  - Our test driver returns 1 when it has discovered a bug
  - OR
  - It ‘crashes’ with a segmentation fault when it has discovered a bug
The downloaded main.c

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

#include "list.h"

int main( int argc, char* argv[] )
{
    // test the create function
    list_t* list = list_create();

    // check to see if the create function did everything it was supposed to
    if( list == NULL )
    {
        printf( "create: create failed to malloc\n" );
        return 1;
    }

    if( list->head != NULL )
    {
        printf( "create: head is not null!\n" );
        return 1;
    }

    if( list->tail != NULL )
    {
        printf( "create: tail is not null!\n" );
        return 1;
    }

    // now test all the other functions (except list_print) to see if
    // they do what they are supposed to

    // you code goes here

    return 0; // tests pass
}
```
Makefile

```
al1: t1 t2 t3 t4 t5

t1: main.c t1.c
   gcc -Wall -std=c99 -o $@ main.c t1.c

for i in {1..5}; do make t$i; done

al2: t1 t2 t3 t4 t5

t2: main.c t2.c
   gcc -Wall -std=c99 -o $@ main.c t2.c

t3: main.c t3.c
   gcc -Wall -std=c99 -o $@ main.c t3.c

t4: main.c t4.c
   gcc -Wall -std=c99 -o $@ main.c t4.c

t5: main.c t5.c
   gcc -Wall -std=c99 -o $@ main.c t5.c

clean:
   rm -f t1 t2 t3 t4 t5 *.o
```

At the command line:

$ make all
or
$ make
or
$ make t1
$ make t2
$ make t3
$ make t4
$ make t5
or
$ for i in {1..5}; do make t$i; done
Execute our `main`

- At the command line:
  
  ```bash
  $ for i in {1..5};
  > do
  > ./t$i; echo $?;
  > done
  ```

- Are we getting the results we expected?
Result of executing `main`

```
alavergn@cs-moie:~/sfuhome/cmpt-127/cmpt127-1194-alavergn/75$ make
gcc -Wall -std=c99 -o t1 main.c t1.c
gcc -Wall -std=c99 -o t2 main.c t2.c
gcc -Wall -std=c99 -o t3 main.c t3.c
gcc -Wall -std=c99 -o t4 main.c t4.c
gcc -Wall -std=c99 -o t5 main.c t5.c
alavergn@cs-moie:~/sfuhome/cmpt-127/cmpt127-1194-alavergn/75$ for i in {1..5}; do ./t$i ; echo $? ; done
 0
 0
 0
 0
 0
alavergn@cs-moie:~/sfuhome/cmpt-127/cmpt127-1194-alavergn/75
```

Hum... no bugs has been discovered!!!
How to proceed!

- Do not look at t1.c, t2.c, t3c, t4.c, t5.c – searching for the bugs! Nope!
- Instead, work only with main.c, extending it, i.e., adding code to it in order to verify that all 7 functions behave as expected
- So, what is the expected behaviour of these 7 functions?

- Hint: no bugs in implementation of list_create(), list_print() and list_destroy()
Let’s investigate `list_prepend( )`

- What are we expecting from `list_prepend( )`?
- From `List.h`:
  ```c
  // Prepends a new element containing integer i to the head of the list. Returns 0 on success, else 1.
  int list_prepend( list_t* list, int i);
  ```

- `list_prepend(list, 26)`
- We are expecting:
  ```
  list
  head
  tail
  val
  26
  ```

- Let’s add code to our `main.c` to confirm that the above expected result is indeed what we obtain from `list_prepend(list, 26)`
Modified main.c

// Testing list_prepend()
int val = 26;
int ret = list_prepend( list, val );
// list_prepend(...) returns 0 on success, else 1.
if ( ret ) {
    puts( "list_prepend() failed." );
    return 1;
}
if( list->head == NULL ) {
    puts( "list_prepend(): list->head NULL." );
    return 1;
}
if( list->tail == NULL ) {
    puts( "list_prepend(): list->tail NULL." );
    return 1;
}
if( list->head != list->tail ) {
    puts( "list_prepend(): first prepend: head != tail." );
    return 1;
}
if( list->head->next != NULL ) {
    puts( "list_prepend(): list->head->next != NULL." );
    return 1;
}
if( list->head->val != val ){
    puts( "list_prepend(): list->head->val != val." );
    return 1;
}
Result of executing `main`

```
alavergn@cs-moyie:~/sfuhome/cmpt-127/cmpt127-1194-alavergn/7$ make
gcc -Wall -std=c99 -o t1 main.c t1.c
gcc -Wall -std=c99 -o t2 main.c t2.c
gcc -Wall -std=c99 -o t3 main.c t3.c
gcc -Wall -std=c99 -o t4 main.c t4.c
gcc -Wall -std=c99 -o t5 main.c t5.c
alavergn@cs-moyie:~/sfuhome/cmpt-127/cmpt127-1194-alavergn/7$ for i in {1..5}; do ./t$i ; echo $? ; done
0
0
0
list_prepend(): list->tail NULL.
1
0
alavergn@cs-moyie:~/sfuhome/cmpt-127/cmpt127-1194-alavergn/7$
```

Bingo!!! We discovered a bug in `t4.c`
Helpful tip when testing `element_create()`

```c
// Testing element_create( )
element_t* el = malloc( sizeof( element_t ) );
assert(el);
memset( el, 0xFF, sizeof( element_t ) );
free(el);

el = element_create( 1492 );
assert(el);

if( el->next )
{
    puts( "element_create(): el->next not NULL." );
    return 1;
}

if( el->val != 1492 )
{
    puts( "element_create(): el->val not correct." );
    return 1;
}
```
Result of our testing

Bingo!!! We discovered a bug in t2.c
Continue modifying `main.c`

- ... by adding code to `main.c` that verifies the expectations you have of the behaviour of each of the functions called from `main.c`
- Once you have detected the bug in each of the 5 implementations of the `List.h` functions (`t1.c`, `t2.c`, `...`, `t5.c`), commit your `main.c` to Gitlab and move on to Lab 7 Task 6 and Task 7