CS 137: Assignment #6

Due on Friday, Nov 11, 2022, at 11:59 PM

Submit all programs using the Marmoset Submission and Testing Server located at
https://marmoset.student.cs.uwaterloo.ca/

Victoria Sakhnini

Fall 2022

Notes:

- Use the examples to guide your formatting for your output. Remember to terminate your output with a newline character.
- For this assignment, you may use any content covered until the end of Module 9.
- <math.h> is not allowed.
- Your solution may not pass marmoset tests for some of the tests due to the wrong implementation or memory leak. Therefore, I advise that even if you pass your own tests and are very positive about your implementation, use Valgrind to detect memory leaks before submitting your solutions. GCC won't detect memory leaks, nor maybe your local compiler. Valgrind is also useful for detecting uninitialized variables and accessing invalid ranges in an array, and more.

Problem 1

In C, you can also have a pointer to a pointer. The syntax for this is given by `int ** ptr` name. Create

a C file named `split.c` that consists of a function

```
int ** split(int a[], int n, int p, int *len1, int *len2)
```

that returns a pointer that points to an array allocated on the heap containing two pointers, each of which points to an integer array; the first with all elements less than or equal to p of size stored where the pointer len1 is pointing at (n1 in main), and the second consisting of all elements greater than p of size stored where the pointer len2 is pointing at (n2 in main). Ensure that you are allocating the memory usage with `malloc` and that you are only allocating exactly enough memory for the two arrays.

You may assume that everything will fit in memory and that a is a non-empty array. Therefore, you are to submit this file containing only your implemented function (that is, you must delete the test cases portion). However, **you must keep the required included libraries.**

The sample code is below.

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

int **split(int a[], int n, int p, int *len1, int *len2);

int main(void)
{
    int a[] = {9,3,2,6,-1,3,6,6,7,8,5,2,3,4,1,0};
    int n = sizeof(a)/sizeof(a[0]);
    int n1,n2;
    int **ans = split(a,n,5,&n1, &n2);
    assert(n1==10);
    assert(n2==6);
    printf("First Array\n");
    for (int i=0; i<n1; i++){
      printf("%d\n",ans[0][i]);
    }
    printf("Second Array\n");
    for (int i=0; i<n2; i++){
      printf("%d\n",ans[1][i]);
    }
    free(ans[1]);
    free(ans[0]);
    free(ans);

    ans = split(a,n,-10,&n1, &n2);
    assert(n1==0);
    assert(n2==n);
    printf("First Array\n");
    for (int i=0; i<n1; i++){
      printf("%d\n",ans[0][i]);
    }
    printf("Second Array\n");
    for (int i=0; i<n2; i++){
      printf("%d\n",ans[1][i]);
    }
```

```
38.    free(ans[1]);
39.    free(ans[0]);
40.    free(ans);
41.
42.    return 0;
43.  }
```

Sample output for the above code

```
First Array
3
2
-1
3
5
2
3
4
1
0
Second Array
9
6
6
6
7
8
First Array
Second Array
9
3
2
6
-1
3
6
6
7
8
5
2
3
4
1
0
```

Problem 2

Create the file `vlinteger.h` which contains the following definitions and declarations:

```
struct linteger {
     int length;
      int *arr;    // array of digits to represent a very long positive integer, most left digit is in index 0.
};
struct linteger *vlintegerCreate(void);
```

// free the memory

```
void vlintegerDestroy(struct linteger *v);
```

// reads digits of a very long integer and stores them in `t1`.

```
void vlintegerRead(struct linteger *t1);
```

//prints the length then prints the digits with no spaces in between ended by `\n`.

//check the exact format and the additional text printed in the sample executions

```
void vlintegerPrint(struct linteger *t1);
```

// returns the addition of `t1` and `t2`. No leading zeros to the left should be kept in the array.

```
struct linteger * vlintegerAdd(struct linteger *t1, struct linteger *t2);
```

// returns the multiplication of `t1` and `t2`. No leading zeros to the left should be kept in the array.

```
struct linteger * vlintegerMult(struct linteger *t1, struct linteger *t2);
```

Implement all the functions above in the file `vlinteger.c`

The function `vlintegerCreate` is implemented for you.

```
struct linteger *vlintegerCreate(void) {
      struct linteger *t1 = malloc(sizeof(struct linteger));
      t1->arr = NULL;
      t1->length = 0; // no values in t1.
      return t1;}
```

Submit `vlinteger.zip`, **which contains** the files `vlinteger.c`, and `vlinteger.h`.

A sample program testing the function, along with its output, is given below.

// Make sure to check that there is no memory leak in any function that uses memory allocation and has // to free that memory before returning.

```
1. #include "vlinteger.h"
2.
3. int main(void)
4. {
5.          struct linteger *int1 = vlintegerCreate();
6.          printf("Enter the digits separated by a space: \n");
7.          vlintegerRead(int1);
8.          vlintegerPrint(int1);
9.          assert(int1->arr[0] !=0);
10.         char c;
11.         scanf("%c", &c);
12.         struct linteger *int2 = vlintegerCreate();
13.         printf("Enter the digits separated by a space: \n");
14.         vlintegerRead(int2);
15.         vlintegerPrint(int2);
16.         assert(int2->arr[0] !=0);
17.         scanf("%c", &c);
18.         struct linteger *add = vlintegerAdd(int1, int2);
19.         printf("addition\n");
20.         vlintegerPrint(add);
21.         assert(add->arr[0] !=0);
22.         scanf("%c", &c);
23.         struct linteger *mult = vlintegerMult(int1, int2);
24.         printf("multiplication\n");
25.         vlintegerPrint(mult);
26.         assert(mult->arr[0] !=0);
27.         vlintegerDestroy(int1);
28.         vlintegerDestroy(int2);
29.         vlintegerDestroy(add);
30.         vlintegerDestroy(mult);
31. }
```

Sample1 execution:

```
Enter the digits separated by a space:
7 8 9 4 5 6 ?
length=6
789456
Enter the digits separated by a space:
3 2 1 9 9 9 2 *
length=7
3219992
addition
length=7
4009448
multiplication
length=13
2542042004352
```

Sample2 execution:

```
Enter the digits separated by a space:
2 3 4 5 7 6 6 6 5 4 3 2 6 7 8 9 3 4 5 6 7 8 9 1 2 3 4 5 6 ?
length=29
23457666543267893456789123456
Enter the digits separated by a space:
9 9 9 9 9 9 8 7 6 5 4 4 4 4 4 4 4 4 4 3 2 1 1 1 1 8 9 7 6 5 4 3 9 8 7 6 9 8 8 8 8
6 5 4 3 2 9 8 7 *
length=49
99999987654444444443211118976543987698888654329 87
addition
length=49
99999987654444444443445695641976666633456545564 43
multiplication
length=78
234576636472886383203935452094139265817130425070115792801599674059469637843072
```