THE UNIVERSITY OF WESTERN ONTARIO

DEPARTMENT OF COMPUTER SCIENCE
LONDON CANADA

Software Tools and Systems Programming
(Computer Science 2211a)

ASSIGNMENT 5
Due date: Tuesday, December 5, 2023, 11:55 PM

Assignment overview

Objectives. The purpose of this assignment is to get experience with

- advanced data structures and ADT,
- manipulation of C pointers and C structures,
- dynamic allocation and deallocation of memory,
- organizing code in multiple files,
- writing Makefile to compile your code.

In this assignment, you are to write a C program to implement an associative matrix structure (2D array) indexed by a (string, integer) pair using binary search trees.

Assignment basic requirements. The code should be well documented and logically organized. The comments should be proper. Your code should be tested carefully before submitting it, especially for boundary cases, such as empty data-structures. Avoid segmentation fault and memory leak.

1 Preliminaries

In this assignment, you will implement the following data structures.

Binary search tree

This will be implemented with pointers and structures. The key type is a (string, integer) pair which will be used later as indices of the matrix structure. The data type can be any type and in this assignment it is pointer to float.

The type definitions for key and data in C are the following.

```
typedef float* Data;
typedef char* Key1;
typedef int Key2;
typedef struct {Key1 key1; Key2 key2;} Key_struct;
typedef Key_struct* Key;
```

For the Key type, you will need a function to initialize a key with dynamic memory allocation, a function to set a key value from a string and an integer, a function to compare two keys, a function to print a key, and a function to free memory used by a key. You may need a function to duplicate a string with dynamic memory allocation.

For Data, you will need a function to initialize a data with dynamic memory allocation, a function to set a data value from a float, a function to print a data, and a function to free memory used by a data.

`Key key_ini();`
Create and initialize a key with dynamic memory allocation.

`void key_set(Key key, Key1 key1, Key2 key2);`
Set, i.e. dynamic allocation memory and copy string and set integer, **key −> key1** and **key −> key2** with **key1** and **key2**.

`int key_comp(Key key1, Key key2);`
If **key1 < key2**, then return value < 0, if **key1 = key2**, then return value = 0, and if **key1 > key2**, then return value > 0. If **key1 −> key1** and **key2 −> key1** are different, then they determine the comparison result. If **key1 −> key1** and **key2 −> key1** are the same, then **key1 −> key2** and **key2 −> key2** determine the comparison result. Note: use strcmp( ) to compare **key1 −> key1** and **key2 −> key1**

`void key_print1(Key key);`
Print a key, **key −> key1** and then **key−> key2**.

`void key_print2(Key key);`
Print a key, **key −> key2** and then **key −> key1**.

`void key_free(Key key);`
Free memory allocated for key.

`Data data_ini();`
Create and initialize a data with dynamic memory allocation.

`void data_set(Data data, float intdata);`
Assign value of data with intdata.

`void data_print(Data data);`
Print a data.

`void data_free(Data data);`
Free memory allocated for data.

The type definitions for binary search trees are the following:

```
typedef struct BStree_node {
  Key key;
  Data data;
  struct BStree_node *left, *right;
} BStree_node;
typedef BStree_node** BStree;
```

The operations for binary search trees are the following.

`BStree bstree_ini(void);`
Allocate memory of type BStree_node*, set the value to NULL, and return a pointer to the allocated memory.

`void bstree_insert(BStree bst, Key key, Data data);`
Insert **data** with **key** into **bst**. If **key** is in **bst**, then do nothing.
You may want to use a helper function for insertion to create a pointer to a tree node from key and data.

```
BStree_node *new_node(Key key, Data data);
```

`Data bstree_search(BStree bst, Key key);`
If **key** is in **bst**, return **key**'s associated data. If **key** is not in **bst**, return NULL.

`void bstree_traverse(BStree bst);`
In order traversal of **bst** and print each node's key and data.

`void bstree_free(BStree bst);`
Free all the dynamically allocated memory of **bst**.

Binary search tree insertion, traverse and free should be implemented with recursion.

In the implementation of binary search trees, you can use any helper function you need.

A Matrix Indexed by a pair of a string and an integer

The matrix structure will be implemented as Matrix using BStree.

The type definition in C is the following.

```
typedef BStree Matrix;
typedef Key1 Index1;
typedef Key2 Index2;
typedef float Value;
```

The operations are the following.

`Matrix matrix_construction(void);`
Matrix construction using bstree_ini();

`unsigned char matrix_index_in(Matrix m, Index1 index1, Index2 index2);`
If location (**index1, index2**) is defined in Matrix **m**, then return 1. Otherwise, return 0.

`const Value *matrix_get(Matrix m, Index1 index1, Index2 index2);`
If location (**index1, index2**) is defined in Matrix **m**, then return a pointer to the associated value. Otherwise, return NULL.

`void matrix_set(Matrix m, Index1 index1, Index2 index2, Value value);`
Assign **value** to Matrix **m** at location (**index1, index2**). If that location already has value, then overwrite.

`void matrix_list(Matrix m);`
Print indices and values in the Matrix **m** (with bstree_traverse()).

`void matrix_destruction(Matrix m);`
Free allocated space (with bstree_free()).

When implementing Matrix, you are free to use helper functions.

2 Organizing the code into multiple files
For this assignment you are to organize the code in the following way:

- In the file datatype.h, define the type Key, the type Data, and declare prototypes of the functions for type Key and type Data.
- In the file datatype.c, implement the functions for type Key and type Data.
- In the file bstree.h, define the type BStree_node, the type BStree and declare prototypes of the operations on BStree.
- In the file bstree.c, implement the functions on BStree.
- In the file matrix.h, define the type Index, the type Value, and the type Matrix and declare prototypes of the operations on Matrix.
- In the file matrix.c, implement the functions on Matrix.
- In the file main.c, your program will
  1. create a new Matrix.
  2. read from stdin, or redirect from a file, sequence of integer string pair (i.e. an integer and then a string, possibly with spaces, per line) and then calculate occurrences of each integer string pair read using the Matrix created.
     - when input from keyboard, use *Ctrl+D* to terminate input.
     - use the return value of **scanf()** to determine the end of input.
  3. print the data in the Matrix
  4. free all allocated memory spaces for the Matrix and terminate.

- Make sure that header files are put into the right .h and .c files.

A sample input is given below.

```
70  Oxford Street
8  3 Valleys Place
101    Western Road
74  3rd Street
4    4 Oaks Crescent
10 Abbott Street
70  Oxford Street
100 1st Street
204 Beech Drive
10 Abbott Street
36 Beechbank Crescent
36 Beechmount Crescent
70  Oxford Street
46 Oxford Street
101    Western Road
```

A sample output is given below.

```
Number    Street            Occurrence
 100    1st Street              1
   8    3 Valleys Place         1
  74    3rd Street              1
   4    4 Oaks Crescent         1
  10    Abbott Street           2
 204    Beech Drive             1
  36    Beechbank Crescent      1
  36    Beechmount Crescent     1
  46    Oxford Street           1
  70    Oxford Street           3
 101    Western Road            2
```

3 Creating a Makefile to compile the source code

You are asked to create a Makefile to compile your source code. When "make" is typed, an executable program called "mymatrix" is generated. Typing "make clean" delete all the files generated by "gcc".

4 Testing your program

You should first implement functions related to type Key and Data. Test these functions to make sure they are correct.

Then implement BStree and test it to make sure it is correct before implementing Matrix.

Then implement Matrix and test it before implementing the main program.

Create your own test input files and redirect test input files as stdin of your program to test your program.

Your program should have no segmentation fault and no memory leak. Your program should print all the elements correctly.

You should test your program by running it on Gaul. Capture the screen of your testing by using script command.