THE UNIVERSITY OF WESTERN ONTARIO

DEPARTMENT OF COMPUTER SCIENCE
LONDON CANADA

Software Tools and Systems Programming
(Computer Science 2211b)

ASSIGNMENT 5
Due date: Friday, March 31, 2017, 11:55 PM

Assignment overview

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

- advanced data structures,
- manipulation of C pointers and C structures,
- dynamic allocation and 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 a sparse matrix structure (2D array) indexed by a pair of strings 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 pair of strings (a pair of pointers to char) which will be used later as indices of the matrix structure. The data type can be any type and in this assignment it is int.

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

```
typedef int Data_Item;
typedef char* Sub_Key;
typedef struct {Sub_Key key1; Sub_Key key2;} Key;
```

You will need a function to duplicate a sub_key (string), a function to generate a key from a pair of sub_keys (strings), a function to print a key, a function to print a data, and a function to compare two keys.

```
int key_comp(Key key1, Key key2);
```
Use strcmp( ) to do comparison. 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.

```
Sub_Key sub_key_dup(Sub_Key skey);
Key key_gen(Sub_Key key1, Sub_Key key2);
void key_print(Key key);
void data_print(Data_Item data);
```

The type definitions for binary search trees are the following:

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

The operations for binary search trees are the following.

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

```
void bs_tree_insert(BStree bst, Key key, Data_Item 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_Item data);
```

```
Data_Item *bs_tree_search(BStree bst, Key key);
```
If **key** is in **bst**, return a pointer to **key**'s associated data. If **key** is not in **bst**, return NULL.

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

```
void bs_tree_free(BStree bst);
```

Free all the dynamically allocated memory of **bst**.

A Matrix Indexed by a pair of Strings
The matrix structure will be implemented as Matrix using BStree.

The type definition in C is the following.

```
typedef BStree Matrix;
typedef Sub_Key Index;
```

The operations are the following.

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

`Data_Item *matrix_get(Matrix m, Index index1, Index index2);`
If at location (**index1, index2**) in Matrix **m**, the value is defined, then return a pointer to the associated data. Otherwise, return NULL.

`void matrix_set(Matrix m, Index index1, Index index2, Data_Item data);`
Assign **data** to Matrix **m** at location (**index1, index2**). If that location already has data, then overwrite.

`void matrix_listing(Matrix m);`
Print values in the Matrix **m** (with bs_tree_traversal()).

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

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 Data_Item, the type Sub_Key, the type Key, and declare prototypes of the functions for type Data_Item and type Key.
- In the file datatype.c, implement the functions for type Data_Item and type Key.
- In the file bs_tree.h, define the type BStree_node, the type BStree and declare prototypes of the operations on BStree.
- In the file bs_tree.c, implement the functions on BStree.
- In the file matrix.h, define the type Index 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, string pairs (a pair of strings, i.e. two strings, per line) and then calculate occurrences of each string pair read using the Matrix created.
3. print the data in the Matrix
4. free all allocated memory spaces for the Matrix and terminate.

A sample input is given below.

```
bba    aa
 aab  aab
bba  aa
aab  abb
bba   aaa
```

A sample output is given below.

```
String 1  String 2   Occurrence
aab       aab        1
aab       abb        1
bba       aa         2
bba       aaa        1
```

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" cleans all the files generated by "gcc".

4 Testing your program

You should implement BStree first and then test it to make sure it is correct before implementing Matrix.

Your program should have no segmentation fault, 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.

You should submit your assignment through OWL. Please check the assignment submission guidelines.