THE UNIVERSITY OF WESTERN ONTARIO

DEPARTMENT OF COMPUTER SCIENCE
LONDON CANADA

Software Tools and Systems Programming
(Computer Science 2211a)

ASSIGNMENT 5
Due date: Thursday, November 28, 2019, 11:55 PM

Assignment overview

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

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

In this assignment, you are to write a C program to implement a dynamic memory allocation and deallocation system 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. Avoid segmentation fault and memory leak.

1 Preliminaries

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

Data types

This assignment need the following data types. The key type and data type are both integers which will be used in the binary search trees and the linked lists.

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

```
typedef int Key;
typedef int Data;
```

You will need the following functions for data types.

```
typedef struct {
  Key key;
  Data data;
  int left, right;
} Node;
typedef struct List_node {
    Key key;
    Data data;
    struct List_node *next;
} List_node;

int key_comp(Key key1, Key key2);
int data_comp(Data data1, Data data2);
void key_print(Key key);
void data_print(Data data);
void print_node(Node node);
void print_list_node(List_node *node);
```

`void print_node(Node node);`
Print **node.key** and **node.data**.

`void print_list_node(List_node *node);`
Print **node->key** and **node->data**.

Binary search tree

This will be implemented with arrays and structures, the same as in assignment 4. The only difference with assignment 4 is that the key type is an integer. This data structure will be used to manage free memory blocks in the dynamic memory allocation and deallocation system.

The type definitions for binary search trees are the following:

```
typedef struct {
    Node *tree_nodes;
    unsigned int *free_nodes;
    int size, top, root;
} BStree_struct;
typedef BStree_struct* BStree;
```

The operations for binary search trees are the following.

`BStree bstree_ini(int size);`
Allocate memory and set up initial values for type BStree.

`void bstree_insert(BStree bst, Key key, Data data);`
Insert **data** with **key** into **bst**. If **key** is in **bst**, then do nothing.

`void bstree_traversal(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**.

The above four functions are also in assignment 4. Code will be given to you.

`void bstree_delete(BStree bst, Key key);`
Delete the node in **bst** with its key equals to **key**. If no such node in **bst**, do nothing.

The above function is not in assignment 4, the code will be given to you.

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

`Key *bstree_data_search(BStree bst, Data data);`
Return the address of the key of the first node, in order of the keys in **bst**, such that the data of the node is greater than or equals to **data** If there is no such node, return NULL.

The above two functions are not in assignment 4. You will write the code.

You may want to use a helper function for insertion to create an integer to a new tree node from key and data.
`static int new_node(BStree bst, Key key, Data data);`

Binary search tree insertion, traversal, free, search, data search, and deletion should be implemented using recursion. You will need to use additional helper functions for the recursion. Binary search tree insertion, or new node(), should check the array bound.

A dynamic memory allocation and deallocation system to simulate system malloc() and free() functions

The dynamic memory allocation and deallocation system will be implemented using an unsigned char array and a BStree.

The binary search tree will be used to store all the free memory blocks in the unsigned char array where key is the starting index of the free block and the data is the size of the free block. Immediately after the initialization the binary search tree contains one node with key equals to **zero** and data equals to **size**, the size of the unsigned char array.

When a user requests **n** bytes, we search the binary search tree to identify a node where the size of the corresponding block is large enough, i.e. $\geq$ **n+4**. We then delete the node from the binary search tree. From the starting index of the block, we use 4 bytes to store **n** and will return the address of the fifth byte to the user. If the block still has memory left after we used **n+4** bytes, insert a new node into the binary search tree. This way, we do not need another data structure to store all the used blocks.

As an example, suppose that the unsigned char array is called **memory** and the binary search tree is called **bst**. Now suppose that we have a request of 20 bytes, and the first large enough block in **memory** is at index 50 with 105 bytes. This means a node in **bst**

with **key=50** and **data=105**. We first delete this node from the **bst**, and then we use **memory[50-53]** to store integer value 20 and will return **&memory[54]** to the user. Since we only used 24 bytes of a block of 105 bytes, we will have **81** bytes available and the starting index of the free block is 50+4+20=**74**. Therefore we insert a node to the **bst** with **key=74** and **data=81**.

When a free memory request with a pointer **ptr** is given, with appropriate pointer casting, we can use the four bytes starting from **ptr-4** to retrieve the size of the memory allocated. This information can be used to determine a free block and this free block can then be inserted into the binary search tree.

The operations for the dynamic memory allocation and deallocation system are the following:

```
void mem_ini(unsigned int size);
```
Initialize an unsigned char array of **size** elements and initialize a binary search tree.

```
void mem_free();
```
Free memory used for the array and the binary search tree.

```
void *simu_malloc(unsigned int size);
```
Allocate **size** bytes and return a pointer to the first byte allocated.

```
void simu_free(void *ptr);
```
Put the allocated memory pointed by **ptr** back to be free memory.

```
void mem_print(void);
```
Print all the free memory blocks.

A list structure to test the dynamic memory allocation and deallocation system

In this implementation, we will use **simu_malloc()** and **simu_free()** instead of **malloc()** and **free()**.
The type definitions for list structure are the following:

```
typedef List_node *List;
```

The operations for list structure are the following.

```
List list_ini(void);
```
Return a pointer to a dynamically allocated and initialized List.

```
Data *list_search(List list, key key);
```
If **key** is in **lsit**, return the address of **key**'s associated data. If **key** is not in **list**, return NULL.

```
void list_add(List list, Key key, Data data);
```
Add **key** with **data** into the front of **list**. If **key** is in **lsit**, then do nothing.

`void list_delete(List list, Key, key);`
Delete the node in **list** with its key equals to **key**. If no such node in **list**, do nothing.

`void list_print(List list);`
Linearly traversal the **list** and print each node's key and data.

`void list_free(List list);`
Free all the dynamically allocated memory of **list**.

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, the type Key, the type Node, the type list_node, and declare prototypes of the functions for these types.
- In the file datatype.c, implement the functions for type Data and type Key.
- In the file bstree.h, define the type BStree and declare prototypes of the operations of BStree.
- In the file bstree.c, implement the functions of BStree.
- In the file memory.h, declare prototypes of the operations on the dynamic memory allocation and deallocation system.
- In the file memory.c, implement the functions of the dynamic memory allocation and deallocation system.
- In the file list.h, define the type List and declare prototypes of the operations of List.
- In the file list.c, implement the functions of List.
- In the file main.c, your program will
    1. read in the size of the dynamic memory allocation and deallocation system.
    2. initialize the dynamic memory allocation and deallocation system.
    3. initialize a linked list.
    4. read from stdin, or redirect from a file, integers (one integer per line) and then calculate occurrences of each integer read using the linked list created.
    5. print the key and data in the list
    6. free all allocated memory spaces for the linked list.
    7. free all allocated memory spaces for the dynamic memory allocation and deallocation system.

A sample input is given below.

```
1000
2
1
3
1
4
3
2
3
3
```

A sample output is given below.

```
integer   Occurrence
  3         4
  2         2
  4         1
  1         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 "mymemory" 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, Data, Node, and List_node. Test these functions to make sure they are correct.
Then implement BStree and test it to make sure it is correct before implementing the memory allocation and deallocation system.
Then implement dynamic memory allocation and deallocation system and test it before using it for List.
Then implement List and test it.
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 with several test cases 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.