Project 1

Own Your Headers

1.1 Intrepid Implementation

In this section, you will be required to implement the following data structures in separate files. The supported functions along with their complexities are listed below:

1.1.1 Doubly Linked List

- `DLLcreateNode(x, n)` Creates a new node that contains the value `x` and points to `n`. → $O(1)$
- `DLLaddFirst(h, x)`: Adds a new node containing value `x` to the front of the linked list → $O(1)$
- `DLLaddLast(h, x)`: Adds a new node containing value `x` to the end of the linked list → $O(1)$
- `DLLfind(h, x)`: Finds the node containing `x`. → $O(n)$
- `DLLinsertAfter(h, k, v)`: Finds the node containing `k` and insert a new node containing value `v` after it. → $O(n)$
- `DLLinsertBefore(h, k, v)`: Finds the node containing `k` and insert a new node containing value `v` before it. → $O(n)$
- `DLLdelete(h, k)`: Finds the node containing `k` and deletes it. → $O(n)$
- `DLLsort(h)`: Sorts the linked list in ascending order.
- `DLLtraverse(h)`: Traverse the linked list sequentially and print the data. → $O(1)$

Here, `h` denotes the pointer to the head of the linked list.

1.1.2 Max Heap

- `MXHmaxHeapify(A)`: Corrects a single violation of max-heap property in a subtrees root. → $O(\log n)$
- `MXHbuildMaxHeap(X)`: Converts an array `X` into a max-heap. → $O(n)$
- `MXHheapSize(A)`: Returns the size of the heap. → $O(1)$
- `MXHheapPush(A, x)`: Insert the value `x` into the max-heap. → $O(\log n)$
- `MXHheapPop(A)`: Extract the max item from the max-heap. → $O(\log n)$
- `MXHheapSort(X)`: Sorts the array `X` using heap sort. → $O(n \log n)$
- `MXHtraverse(X)`: Traverse the heap array sequentially and print the keys → $O(n)$

Here, `A` denotes the array used to store heap.

1.1.3 Binary Search Tree

- `BSTinsert(R, v)`: Inserts a node with value `v`. → $O(h)$
- `BSTfind(R, v)`: Checks if `v` exists in BST. → $O(h)$
- `BSTfindMin(R)`: Finds the minimum value stored in BST. → $O(h)$
- `BSTfindMax(R)`: Finds the maximum value stored in BST. → $O(h)$
- `BSTnextLarger(R, x)`: Finds the successor of `x`. → $O(h)$
- `BSTdelete(R, x)`: Deletes the node containing the value `x`. → $O(h)$
- `BSTrank(R, x)`: Returns how many nodes contain value $\leq x$. → $O(h)$
- `BSTbfs(R)`: Traverse the whole list in Breadth-First Order and print the keys → $O(n)$
- `BSTdfs(R)`: Traverse the whole list in Depth-First Order and print the keys → $O(n)$

1.1.4 Testing Your Code

Each file should contain a `main` function demonstrating a simple example using all the functions implemented.

1.1.5 Making Your Code Submission Ready

Each file should be named `T1XXXZZ.c` where `XXX` will be replaced by `DLL` for Doubly Linked List implementation, `MXH` for Max Heap implementation and `BST` for Binary Search Tree implementation. `ZZ` will be replaced by the last two digits of your student ID.

1.2 Hellish Headers

As we all know, files with the extension `.h` are called header files in C. These header files contain function declarations which can be used in C programs. For this task, you need to create your own header files which can be used in any other C programs. Create 3 separate header files for each of the data structures you implemented. The name of your header file should look like the following: `XXXheap.h`, where `XXX` is replaced by your initial. For example: if I were to create a header file for my heap implementation, I'd name it `MBHheap.h`.

The accompanying C file will contain the actual implementation. They'll be named like: `XXXheap.c`, where `XXX` is replaced by your initial. For example: if I were to create a C file for my heap implementation, I'd name it `MBHheap.c`.

For your convenience, I have implemented singly linked list and created a header file that supports the following operations:

- `SLLcreateNode(x, n)` Creates a new node that contains the value `x` and points to `n`.
- `SLLaddFirst(h, x)`: Adds a new node containing value `x` to the front of the linked list $\rightarrow O(n)$
- `SLLaddLast(h, x)`: Adds a new node containing value `x` to the end of the linked list $\rightarrow O(1)$
- `SLLinsertAfter(h, k, v)`: Finds the node containing `k` and insert a new node containing value `v` after it. $\rightarrow O(n)$
- `SLLinsertBefore(h, k, v)`: Finds the node containing `k` and insert a new node containing value `v` before it. $\rightarrow O(n)$
- `SLLdelete(h, k)`: Finds the node containing `k` and deletes it. $\rightarrow O(n)$
- `SLLtraverse(h)`: Traverse the list from first to last. $\rightarrow O(n)$

Check the `MBHsinglylinkedlist.h` and `MBHsinglylinkedlist.c` for further clarification. Notice how I modified the return types of the functions to handle multiple header files simultaneously. The first few lines in the header file ensures that the same library isn't added multiple times. The `T2test01.c` file contains a simple demonstration of the header file.

1.2.1 Running My Code

To run `T2test01.c` file, you need to install a C compiler and set the environment variable in your computer.

Code::Blocks Installation

For this project, we'll be using Code::Blocks to compile and run our codes. Code::Blocks is a free C, C++ and Fortran IDE built to meet the most demanding needs of its users. It is designed to be very extensible and fully configurable. Check if your PC has Code::Blocks installed. If not, download it from the internet/ftp and install. If there are multiple options, download the following one:

```
codeblocks-17.12mingw-setup.exe
```

Remember where you installed the program. It might be useful later.

Environment Variable

An environment variable is a dynamic "object" on a computer, containing an editable value, which may be used by one or more software programs in Windows. Environment variables help programs know what directory to install files in, where to store temporary files, and where to find user profile settings. They help shape the environment in which programs run on your computer. To compile a C program, your PC needs to know where to find the compiler. It uses environment variable to do so. Check if the correct variable is set for your PC. Open your folder, while pressing shift, right-click on an empty space of the folder. If you do it properly, you will see a pop-up menu. There you will find an option `Open PowerShell Window here` or `Open command window here`. Click on the option and enter the command:

```
mingw32-gcc -v
```

If the environment variables are set properly, you will see a message showing the gcc version. If not, you will see `gcc: command not found`. In that case, you will have to set the PATH variable of the Environment Variables using the following steps:

1. Go to the installation path of Code::Blocks > MinGW > bin. Copy the whole path.
2. Right-Click on This PC
3. Go to Properties > Advanced system settings > Environment Variables...
4. In System variables section, find Variable named `Path`. Double-Click on it. A window named `Edit environment variable` will be opened.*
5. Click on New and paste the path you copied in Step 1
6. Close the windows by clicking OK
7. Close and Reopen your PowerShell /Command Prompt

*In some older versions of Windows, a different window named `Edit System Variable` opens. In that case, skip step 5, go to the end of the `Variable value:` field, put a semicolon (;) and paste the path.

Running The Code

Put `MBHsll.h`, `MBHsll.c` and `L4T2test01.c` in the same folder. While pressing shift, right-click on an empty space of the folder to open `Powershell` or `Command prompt`. Run the following command:

```
mingw32-gcc -o test.exe MBHsll.c T2test01.c
```

If done correctly, an executable file named `test.exe` will be created. Run that file using `./test.exe` (for PowerShell) or `test.exe` (for Command Prompt).

1.2.2 Testing Your Code

Each header should be included in a single file containing a `main` function demonstrating a simple example using for each data structure using all the functions implemented. To run the file, you need to run a command similar to the following:

```
mingw32-gcc -o test.exe MBHdll.c MBHmxh.c MBHbst.c T2test01.c
```

This will create an executable file named `test.exe`. Run that file using the command specified before.

1.2.3 Making Your Code Submission Ready

Create a file named `T2testZZ.c` where ZZ will be replaced by the last two digits of your student ID. Include the three header files and demonstrate their work with simple examples as shown in `T2test01.c` using a `main` function.

The header files should be named as mentioned before.

1.3 Submitting Your Code

Submit all 10 (3 from task 1 + 7 from task 2) files in Google Classroom.