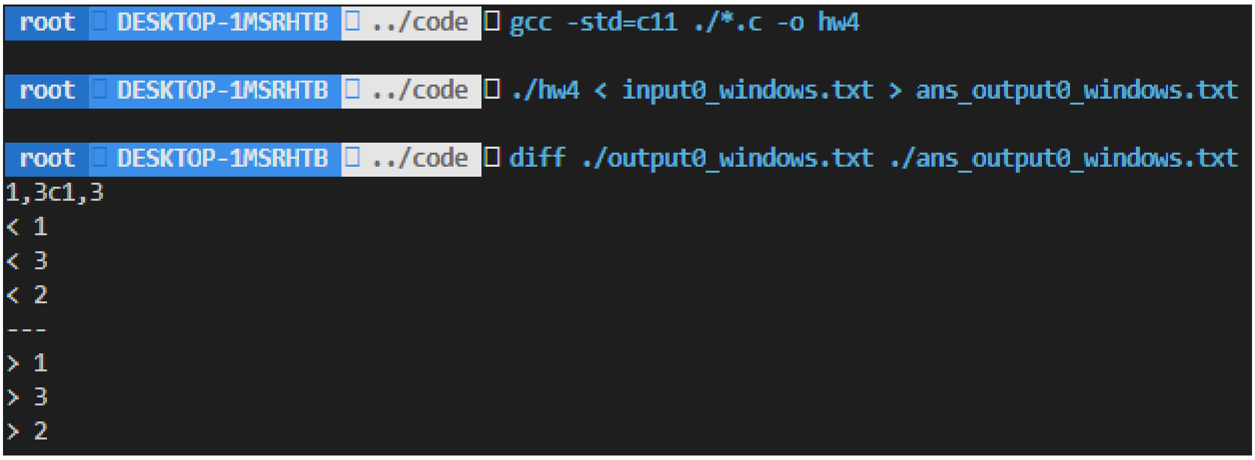
Data Structure Assignment 4

ID: E14066282	**Name:** 溫梓傑	**Department:** ME 110

○ Result Screenshots

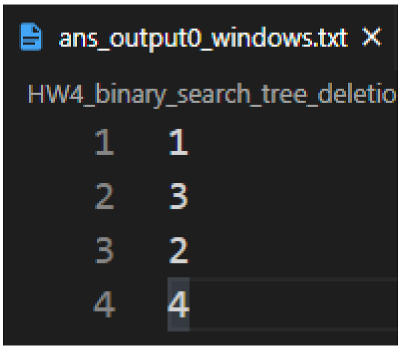


```
root □ DESKTOP-1MSRHTB □ ../code □ gcc -std=c11 ./*.c -o hw4

root □ DESKTOP-1MSRHTB □ ../code □ ./hw4 < input0_windows.txt > ans_output0_windows.txt

root □ DESKTOP-1MSRHTB □ ../code □ diff ./output0_windows.txt ./ans_output0_windows.txt
1,3c1,3
< 1
< 3
< 2
---
> 1
> 3
> 2
```

Figure 1 Screenshot of command line



```
1
3
2
4
```

Figure 2 ans_output0_windows.txt

○ Program Architecture

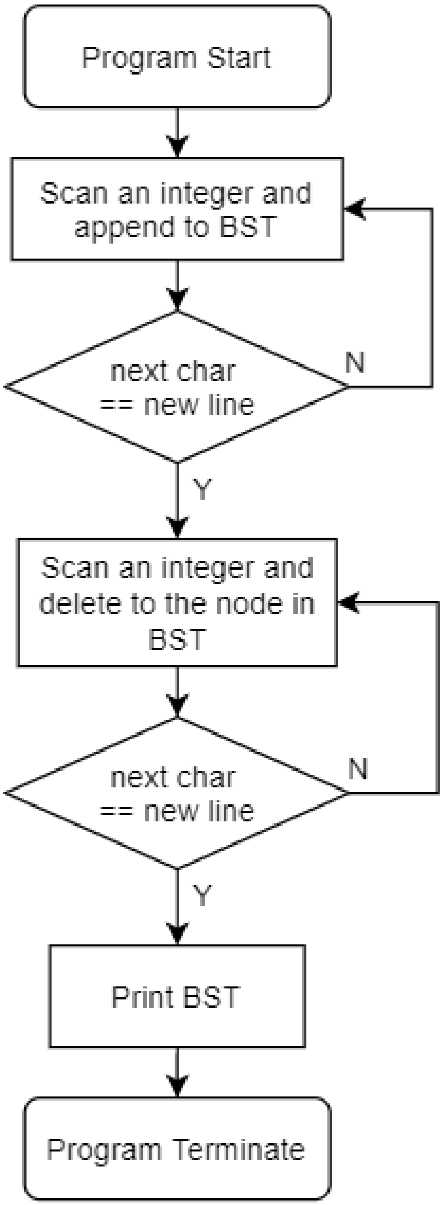


Figure 3 Flow chart of hw4

- ○ **Program Functions**

Double_LL.h

```
LinkList *create_ll();
```

Constructs a link list.

Parameters

None.

Return Value

Returns the new pointer of the link list.

- If construction fails, returns NULL.

```
Node *create_node(Node_tr *np_tr);
```

Constructs a node.

Parameters

```
np_tr
```

The element that would be initialized in the constructed node.

Return Value

Returns the new pointer of the node.

- If construction fails, returns NULL.

```
void push_node(LinkList *lp, Node *np);
```

Inserts the node on the back of the link list.

Parameters

```
lp
```

The pointer of the link list.

```
np
```

The pointer of the node.

Return Value

None.

```
Node_tr* pop_node(LinkList *lp);
```

Removes the node on the back of the link list.

Parameters

`lp`

The pointer of the link list.

Return Value

Returns the back element before removal.

- If the link list is empty, program terminates.

```
void push_front_node(LinkList *lp, Node *np)
```

Inserts the node at the front of the link list.

Parameters

`lp`

The pointer of the link list.

`np`

The pointer of the node.

Return Value

None.

```
Node_tr* pop_front_node(LinkList *lp);
```

Removes the node at the front of the link list.

Parameters

`lp`

The pointer of the link list.

Return Value

Returns the front element before removal.

- If the link list is empty, program terminates.

```
void free_LL(LinkList *lp);
```

Free all nodes in the link list.

Parameters

`lp`

The pointer of the link list.

Return Value

None.

Binary_Tree.h

```
Node_tr *create_node_tr(int key)
```

Constructs a tree node.

Parameters

`key`

The element that would be initialized in the constructed tree node.

Return Value

Returns the new pointer of the node.

- If construction fails, returns NULL.

```
void append_search_tree(Node_tr *root, int val)
```

Parameters

`root`

The root node of the BST (binary search tree).

`val`

The value of key to be appended to the BST.

Return Value

None.

```
Node_tr *findMin(Node_tr *root)
```

Returns the leftmost node in the BST.

Parameters

`root`

The root node of the BST (binary search tree).

Return Value

Returns the leftmost node in the BST.

- If root is NULL, then it returns NULL.

```
void print_tree(Node_tr *root)
```

Print the BST in level order.

Parameters

`root`

The root node of the BST (binary search tree).

Return Value

None.

○ Program Design

Print the BST in level order

在`void print_tree(Node_tr *root)` 中，由於需要先暫存資料，再讀出並進行讀取，剛好符合 FIFO，因此使用了 Queue 的資料結構來進行 level order 的存取。

Delete a key in the BST

在`Node_tr *delete_node_tr(Node_tr *root, int val)` 中，使用了遞迴的概念來撰寫：

程式第一步

先找出 key node，利用遞迴方式尋找，在呼叫函數本身後，函數會回傳新的節點指標來更新`root->left`或是`root->right`，原因是這兩個 pointer 值(刪除目標節點的 parent 成員指標)，必須要更新。可能會產生以下兩個結果：

1. 找到 NULL，即 key 不存在此二元搜尋樹(BST)中，程式會沿著呼叫順序，一路解開 function stack，最後程式停止。
2. 找到對應的 key，程式進入第二步。

程式第二步

首先將刪除節點的問題歸為三類：

1. 刪除節點為 leaf node

   動作：刪除該節點，回傳 NULL(用來更新前一層 stack 的 left 或是 right)。

2. 刪除節點只有一個 child node

   動作：把該 child node 搬至原本刪除的節點，並回傳 child node pointer。

3. 刪除節點有兩個 children node

動作：將此問題分成第 1 點或是第 2 點，進行遞迴呼叫，由於題目要求「以刪除節點的 right subtree 的最小值進行取代」，因此實作成

`root->right = delete_node_tr(root->right, temp->key);`。

Operating System

Ubuntu 20.04.1 LTS (Focal Fossa)

Compiler

gcc (Ubuntu 9.3.0-10ubuntu2) 9.3.0

Compile

```
gcc -std=c11 ./*.c -o hw4
```

Run

```
./hw4 < input.txt > output.txt
```