# Data Structure Assignment 4

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

## Result Screenshots

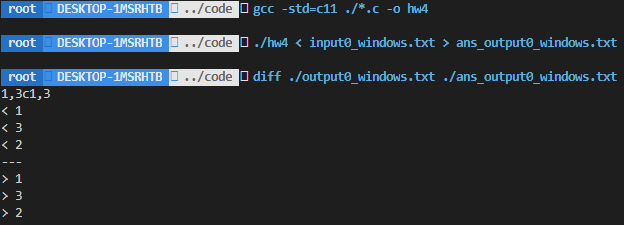


Figure 1 Screenshot of command line

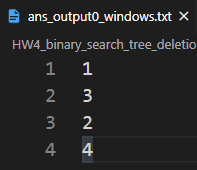


Figure 2 ans\_output0\_windows.txt

|  |  |
| --- | --- |
|  |  |

## Program Architecture

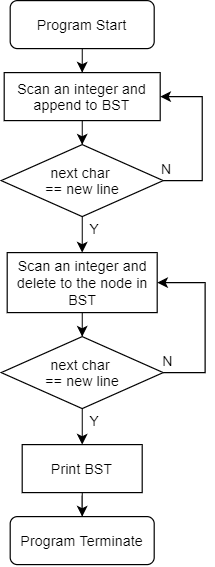


Figure 3 Flow chart of hw4

## 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)。

1. 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。

1. 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