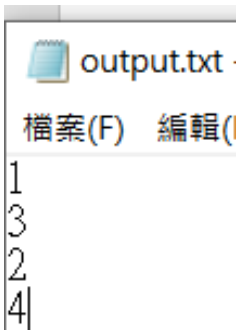
HW4

(1) result screenshot

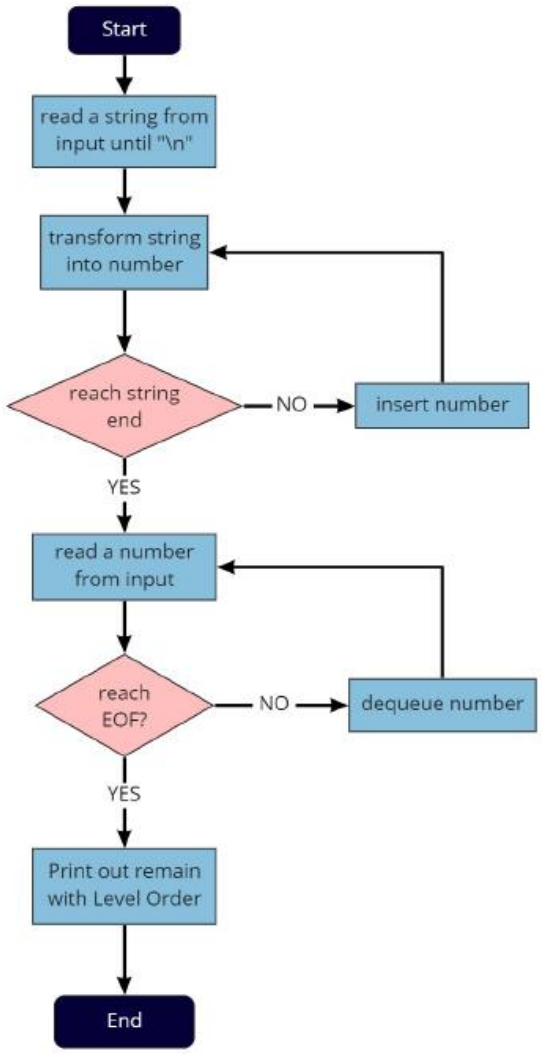
```
Jamei@DESKTOP-PGELV1B MINGW64 /C/Users/Jamei
$ gcc -std=c11 -o hw_4 hw_4.c

Jamei@DESKTOP-PGELV1B MINGW64 /C/Users/Jamei
$ ./hw_4<input0_windows.txt>output.txt
```

Screenshot of command line.



Screenshot of “output.txt”.

(2) program architecture



(3) program function

```
struct bintreeNode{
    int data;
    struct bintreeNode* left;
    struct bintreeNode* right;
};//stucture of binary tree
struct queueNode
{
    struct bintreeNode* node;
    struct queueNode* next;
};//Since we will printout with Level Order, we can use a queue to do that.
```

Parameters

In binetreeNodes

`data` - the integer used to save keys .

`left` - a pointer used to point the left subtree in the node.

`right` - pointer used to point the right subtree.

In Queue

`node` - it means the data in the queue, saved as binetreeNodes pointer.

`next` - the pointer pointed to the next data.

```
typedef struct bintreeNode bintree;
typedef bintree * treeptr;
typedef struct queueNode queue;
typedef queue * qptr;
```

type define list, I will use abbreviation in under description.

```
qptr Qhead=NULL;
qptr Qtail=NULL;
int firstline=1;
treeptr root=NULL;
```

Parameters

`Qhead, Qtail`- queue 的頭跟尾，一開始沒指向任何東西。

`firstline`- 用來確認有沒有印第一行的參數，因為在輸出時事先印換行在輸出，所以需要避免第一行換行的情況。其值為 1 表示準備印第一行，為 0 表示不是印第一行。

`root-` binary tree 的頂端。

```
int main() {
    int num=0;
char temp;
while(scanf("%c",&temp)!=EOF)
{
  if(temp=='\n')
  {
      insert(num,&root);//最後一個數後面會直接接換行，記得要讓最後一個數也 insert
      break;
  }
  if(temp==' ')
  {
      insert(num,&root);
      num=0;
  }
  else
  {
      num=num*10+(temp-'0');
```

```
  }
}
//用字元一次讀一個，再把他轉成數字加起來(ex,輸入 57，num 一開始=0，讀入 5，num=0*10+5=5；讀入 7，num=5*10+7=57)
//如果遇到空白就可以 insert，如果遇到換行就插入最後一個數然後跳出迴圈
    while(scanf("%d",&temp)!=EOF) root=delete(temp,&root);
//來到第二行，一次讀入一個數值到文件尾端，並刪除那個數
    levelOrder(root);
//以 Level order 方式印出來
    return 0;
}
```

Parameter

num -用來記錄要 insert 或 delete 的數

temp –暫時用來記錄讀入的字元

Return value

如果程式順利執行會 return 0

Function

insert –安插新的 treenode

```
void insert(int item,treeptr *current)
{
    if(*current==NULL)
    {
        treeptr newptr=getnewN(item);
        if(newptr==NULL) printf("No space to insert tree.\n");
        *current=newptr;
    }
//如果所在位置沒有指向任何東西，它就可以放新的 treenode
    if(item<(*current)->data) insert(item,&((*current)->left));
//如果所在位置有指向東西，且該東西的值比 item 大，往那個東西的 left subtree 找適當位置
    if(item>(*current)->data) insert(item,&((*current)->right));
//如果所在位置有指向東西，且該東西的值比 item 小，往那個東西的 right subtree 找適當位置
}
```

Parameter

item –要安插的數，只是一個值

current –當前所在位置指向的地方，每筆資料都是從 root 開始

newptr –暫存新產生的 treenode

Return value

這個 function 沒有 return value

getNewN –產生一個 treenode

```
treeptr getnewN(int item)
{
    treeptr newptr=malloc(sizeof(bintree));
    if(newptr!=NULL)
    {
        newptr->data=item;
        newptr->left=NULL;
```

```
        newptr->right=NULL;
    }
//如果 malloc 成功，初始化產生的 ptr
    return newptr;
}
```

Parameter

`item` –要安插的值，如果 malloc 成功， newptr 指向的 data 會是這個值

`newptr` –一個指標，指向 malloc 出來的 bintree 空間

Return value

該程式會把產生完的 newptr 回傳回去給 insert 內使用

delete –刪除不要的 treenode

```
treeptr delete(int item,treeptr *current)
{
    if((*current)==NULL)
        return NULL;
//當他發現它啥都找不到的時候，不要動回傳回去
    if(item>(*current)->data) (*current)->right=delete(item,&((*current)->right));
//如果要刪除的數比所在的大，往它的右邊找
    else if(item<(*current)->data) (*current)->left=delete(item,&((*current)->left));
//如果要刪除的比所在的小，往左邊找
    else
//要刪除的數跟所在一樣大
    {
        if((*current)->left==NULL)
        {
            treeptr temp=(*current)->right;
            free(*current);
            return temp;
        }
//如果所在的左邊為空，就把右邊的記起來，之後右邊的就會接到上一層了(如果右邊為空也會記起來，就是沒有 child 的意思，只有左邊空的話，是一個 child 的意思)
        else if((*current)->right==NULL)
        {
            treeptr temp=(*current)->left;
            free(*current);
            return temp;
        }
//如果左邊不是空的，就記著左邊，讓左邊跟上一層接起來(一個 child)
        treeptr temp=(*current)->right;
//確定有兩個 child 了，先進去右邊子樹
        while(temp&&temp->left!=NULL)//如果它跟它左邊的下面不是空的，往左邊下去
        {
            temp=temp->left;
//讓最小變成當前的左邊
        }
        (*current)->data=temp->data;
//最後把找到的最小值跟要刪除的值換
        (*current)->right=delete(temp->data,&((*current)->right));
```

```
//然後準備把最小的刪掉，因為它會從它的右邊開始找，最後找到替換的樹，然後讓它的下面往上接或是讓它指向空的
    }
    return (*current);
//在 delete 做完之後，負責把值歸還回去
}
```

Parameter

`item` –想要刪掉的數

`*current` –在裡面搜尋時的所在位置

`temp` –用來儲存要替換的位置

Return value

和 main function 的 `root=delete(temp,&root)` 有直接關聯，因為回傳的是 `treeptr`，所以將讓回傳值 assign 給 root，如果有動到 `root`(刪除的是最頂端)，那 `root` 會被重置，否則就只更動需要更動的地方，傳回 root 的東西和原本傳下去的會一樣。

levelOrder –以廣度尋訪並印出來

```
void levelOrder(treeptr begin)
{
    treeptr current=begin;
//先把一開始傳入的位置，通常是 root 放在紀錄點
    enq(current);
//enqueue 一開始的東西
    while(!IsEmpty(Qhead))
    {
        current=Qhead->node;
        deq();
        if(current->left!=NULL) enq(current->left);
        if(current->right!=NULL) enq(current->right);
    }
//如果 queue 沒空，dequeue 一個東西，並看那個東西有沒有左右 child，有的話把 child enqueue 進去
}
```

Parameter

`begin` -一開始要開始印的位置

`current` -看 dequeue 出來的東西有沒有左右 child node

Return value

這個 function 沒有 return value

enq(treeptr),deq(),getnewQ(treeptr),IsEmpty(qptr) –queue 的相關操作，因為前面作業寫過了所以在此省略