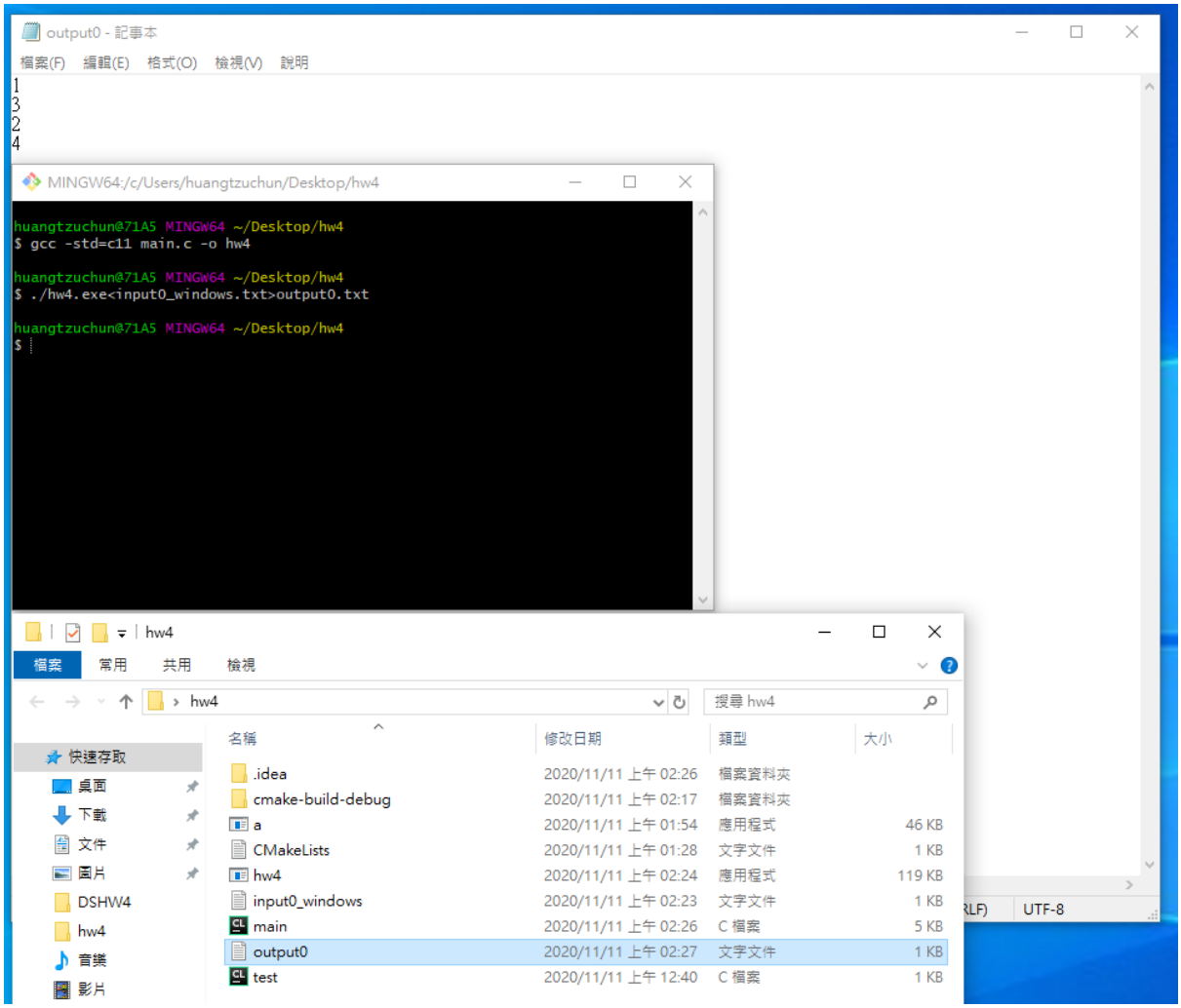
HW4

(1) result screenshot



(2) program structure

(3) program functions

`void add_node(int num)`

Usage

- Add node with num to the tree

Parameters

- `int num` :the number of new node want to add.

Return values

- No return value(void)

`Node* find_mini_node(Node* root)`

Usage:

- To find minimum in right subtree.

Parameters

- `Node* root` Node type pointer(self define), which is the start root for doing search.

Return values

- `Node*` Node type pointer(self define), which is the node with minimum value in right subtree.

`Node* delete_node(int delete_num,Node *root)`

Usage

- To delete node which has `delete_num` .

Parameters

- `int delete_num` the number want to search in tree and delete that node.
- `Node *root`

Return values

`Node*` Node type pointer(self define), return this pointer of recursive structure.

`void enqueue(Node* ptr)`

Usage

- enqueue an Node to the queue.

Parameters

`Node* ptr` the pointer point to the node you want to enqueue.

Return values

- No return value(void)

`void enqueue_children(Node* ptr)`

Usage

- enqueue an child of Node to the queue.

Parameters

- `Node* ptr` : the pointer point to the node which you want to enqueue it's child.

Return values

- No return value(void)

`void dequeue(void)`

Usage

- dequeue first node in queue.

Parameters

- No parameters `void`.

Return values

- No return value(void)

`void print_tree(void)`

Usage

- print all remain nodes in the binary search tree.

Parameters

- No parameters `void`.

Return values

- No return value(void)

---

(4) How I design this program

1. How to delete node in binary search tree?

- we will possible face three fallowing cases.

1. The node want to delete has child.

```
if (root->leftChild == NULL && root->rightChild == NULL) {
    free(root);
    root = NULL;
}
```

2. The node want to delete only has left child.

```
else if (root->leftChild == NULL) {
            Node *temp = root;
            root = root->rightChild;
            free(temp);
        }
```

3. The node want to delete only has right child.

```
else if (root->rightChild == NULL) {
    Node *temp = root;
    root = root->leftChild;
    free(temp);
}
```

4. The node want to delete has both left child and right child.

```
else {
    Node *temp = find_mini_node(root->rightChild);
    root->data = temp->data;
    root->rightChild = delete_node(temp->data,root->rightChild);
}
```

2.How to print all remain node?

```
typedef struct node Node;
struct node
{
    int data;
    Node* leftChild;
    Node* rightChild;
    Node* next;
```

```
};

Node *rootPointer = NULL, *front = NULL, *rear = NULL;
```

- 在這個自定義的Struct中加入`Node* next;` 是`void print_tree()` 中以linked list 實踐queue所需的鏈結指標。
- `void print_tree()` 主要以linked list 實踐queue，一開始把root的node加入queue（如果是空的樹則完全忽略以下步驟）。在接下來的loop中，重複從queue中移除node，再加入被移除node的子結點(left child and right child)的步驟，以達到以level order的順序印出剩餘node的目的。