

開發平台：Windows7

開發環境：DEV - C++ 4.9.9.2

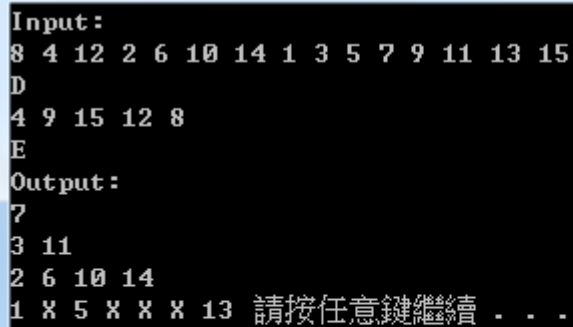
BST_deletion 程式功能：

輸入建樹的資料〈以空白鍵或回車鍵隔開資料〉

無論有沒有要刪除節點都需先輸入 D

若有要刪除節點：輸入要刪除節點的資料，最後輸入 E 結束

若無：直接輸入 E 結束



```
Input:
8 4 12 2 6 10 14 1 3 5 7 9 11 13 15
D
4 9 15 12 8
E
Output:
7
3 11
2 6 10 14
1 x 5 x x x 13 請按任意鍵繼續 . . .
```

BST_deletion 程式架構：

typedef struct treenode *TreePointer; [//refer to textbook: linked representation](#)

struct treenode

{

int data;

TreePointer leftchild;

TreePointer rightchild;

};

TreePointer TreeNodeSearch() [//refer to textbook: search tree node](#)

void InsertTreeNode() [//refer to textbook: insert tree node](#)

void TreePrint() [//refer to textbook: level order traversal](#)

void TreeNodeDelete() & void DeleteProcess():

```

void TreeNodeDelete(TreePointer *root, char *input, int *total) //root 樹的根節點
{
    int i, data;
    TreePointer head, delnode; //head 紀錄要刪除節點的parent位置, delnode 紀錄要刪除節點的位置

    while(1)
    {
        scanf("%s", input); //讀取要刪除節點的值
        if(input[0] >= 48 && input[0] <= 57)
        {
            data = 0;
            (*total)--;
            for(i = 0 ; i < strlen(input) ; i++) data = data*10 + input[i]-48;
            if((*root)->data == data) DeleteProcess(root, root); //如果刪除的節點為root, 將parent視為root傳入DeleteProcess
            else //刪除非root的節點
            {
                delnode = *root;
                while( delnode->data != data) //往下搜尋要刪除的節點
                {
                    head = delnode; //紀錄下一個節點的parent
                    if(data < delnode->data) delnode = delnode->leftchild; //往下一個節點移動, 左或右
                    if(data > delnode->data) delnode = delnode->rightchild;
                }
                DeleteProcess(&head, &delnode); //將要刪除的節點及節parent傳入DeleteProcess
            }
            free(delnode); //釋放刪除的節點回記憶體
        }
        else if(input[0] == 'E') return; //讀到結束字元'E', 結束刪除
    }
}

void DeleteProcess(TreePointer *head, TreePointer *delnode)
{
    TreePointer temp, subhead = *head; //temp 為取代原本delnode位置的節點, subhead 為temp的parent位置

    if( !((*delnode)->leftchild) ) //如果刪除節點的左子樹是空的, delnode的位置直接由右子樹的root取代
    {
        if(*head == *delnode) (*head) = (*delnode)->rightchild; //處理delnode為root的情況
        else //處理非root的情況
        {
            if((*head)->data < (*delnode)->data) (*head)->rightchild = (*delnode)->rightchild; //delnode在其parent右邊的情況
            else (*head)->leftchild = (*delnode)->rightchild; //delnode在其parent左邊的情況
        }
    }
    else if( !((*delnode)->rightchild) ) //如果刪除節點的右子樹是空的, delnode的位置直接由左子樹的root取代
    {
        if(*head == *delnode) (*head) = (*delnode)->leftchild; //處理delnode為root的情況
        else //處理非root的情況
        {
            if((*head)->data < (*delnode)->data) (*head)->rightchild = (*delnode)->leftchild;
            else (*head)->leftchild = (*delnode)->leftchild;
        }
    }
    else //左, 右子樹都非空

    else //左, 右子樹都非空
    {
        temp = (*delnode)->leftchild; //從左子樹找最大值的節點, 並記錄其位置於temp
        while( temp->rightchild != NULL ) //從結束條件知道temp的右子樹root為NULL
        {
            subhead = temp; //紀錄temp的parent位置
            temp = temp->rightchild;
        }
        if(subhead == *head) temp->rightchild = (*delnode)->rightchild; //delnode的左子樹root為最大值, 左子樹root的右邊接上delnode的右子樹
        else
        {
            if(temp->leftchild) subhead->rightchild = temp->leftchild; //temp的左邊還有東西, 接到subhead的右邊
            else subhead->rightchild = NULL; //左邊沒東西了, subhead的右邊直接設為NULL
            temp->leftchild = (*delnode)->leftchild; //temp取代delnode連接上delnode的左, 右子樹
            temp->rightchild = (*delnode)->rightchild;
        }
        if(*head == *delnode) *head = temp; //delnode為root則root以temp取代
        else //delnode非root則delnode的parent接上temp
        {
            if((*head)->data < (*delnode)->data) (*head)->rightchild = temp;
            else (*head)->leftchild = temp;
        }
    }
}

```

Equivalence_class 程式功能：

輸入集合的資料〈以空白鍵或回車鍵隔開資料〉

無論有沒有等價類都需先輸入 S

若有等價類：輸入等價類的等式，最後輸入 E 結束

若無：直接輸入 E 結束

```
Input :
0 1 2 3 4 5 6 7 8 9
S
0=9
2=5
9=5
1=6
3=4
7=1
4=8
3=6
E
Output :
<<0><2,9><5>>
<<1><3,6,7><4,8>>
請按任意鍵繼續 - - -
```

Equivalence_class 程式架構：

```
#define Max_Set_Size 1024
typedef struct{                                //refer to textbook: representation of disjoint set
    int value;
    int parent;
}Treenode;
Treenode set[Max_Set_Size];                    //record element in set
int total = 0;                                //record how many elements are in set
void Union(int i, int j)                       //refer to textbook: disjoint union
int Find(int i)                                //refer to textbook
```

Find 函數回傳所在集合的 element's index

Union 傳入 2 個 elements 所在集合的 index，比較 roots 的值大小，值較大的接在小的下方

void equivalence_construct()

```
void equivalence_construct(char *input)
{
    int i,data1,data2;

    while(1)
    {
        scanf("%s",input); //讀取等價類的等式
        if(input[0] >= 48 && input[0]<= 57)
        {
            data1 = data2 = 0;
            for(i = 0 ; input[i] != '=' ; i++) data1 = data1*10 + input[i]-48; //讀取等式左邊的數值
            for(i = i+1 ; i < strlen(input) ; i++) data2 = data2*10 + input[i]-48; //讀取等式右邊的數值
            Union( Find(data1), Find(data2) ); //找到等式兩邊的元素所在集合然後Union
        }
        else if(input[0] == 'E') return; //讀到結束字元'E',結束
        else
        {
            printf("Error! %s is invalid input.",input);
            exit(EXIT_FAILURE);
        }
    }
}
```

void TreePrint()

```
void TreePrint()
{
    int i,j,rootnumber = 0,count,currentlayer,roots[total],layer[total]; //roots紀錄元素所在集合的root, layer紀錄在樹中的level
    int *r = malloc((rootnumber+1)*sizeof(int));

    for(i = 0 ; i < total ; i++) if(set[i].parent < 0) //紀錄總共有哪些root在r[]中
    {
        r[rootnumber] = i;
        r = realloc(r,((++rootnumber)+1)*sizeof(int));
    }

    for(i = 0 ; i < total ; i++) //找每個元素在樹中的root與layer,過程與Find類似
    {
        layer[i] = 1;
        for(j = i ; set[j].parent >= 0 ; j = set[j].parent) layer[i]++;
        roots[i] = j;
    }

    for(i = 0 ; i < rootnumber ; i++) //印出r[]中第i個roots的集合
    {
        currentlayer = 1;
        count = 0;
        printf("(");
        while(count + set[r[i]].parent < 0) //印完r[i]集合內所有元素則結束
        {
            printf("(");
            for(j = 0 ; j < total ; j++) if(layer[j] == currentlayer && roots[j] == r[i])
            {
                printf("%d,",set[j].value); //print element in current layer & counting printed element
                count++;
            }
            currentlayer++;
            printf("\b");
        }
        printf(")\n");
    }
}
```