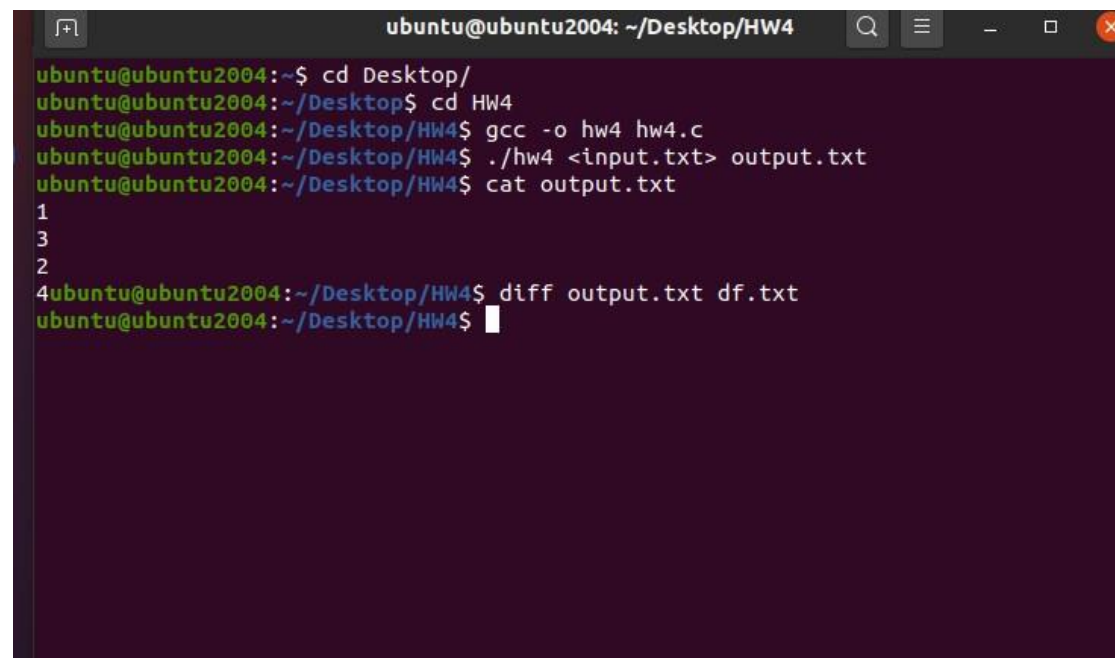


HW4

Result screenshot



```
ubuntu@ubuntu2004: ~/Desktop/HW4
ubuntu@ubuntu2004:~$ cd Desktop/
ubuntu@ubuntu2004:~/Desktop$ cd HW4
ubuntu@ubuntu2004:~/Desktop/HW4$ gcc -o hw4 hw4.c
ubuntu@ubuntu2004:~/Desktop/HW4$ ./hw4 <input.txt> output.txt
ubuntu@ubuntu2004:~/Desktop/HW4$ cat output.txt
1
3
2
4
ubuntu@ubuntu2004:~/Desktop/HW4$ diff output.txt df.txt
ubuntu@ubuntu2004:~/Desktop/HW4$
```

Program Architecture

先用函式讀入上下兩列的測資，分別存成兩個陣列，再利用 **linked list** 去建立二元樹，為了能 **trace back**，我不僅有設置左右節點，也有設置父節點。接著使用我自己寫的 **deletenode** 函式去刪除節點(此處經由我自己花大量時間去畫二元樹，我發現此功能很複雜，有許多種可能性要考慮，下面會詳細解釋)。接著再使用二元樹的 **linked list** 走訪的方式，寫一個 **preorder** 副程式去印出整棵樹。就完成了。

這次程式我為了 **debug**，大約編譯測試了有 1000 次，有許多 **debug** 時用到的 **printf** 函式，**preordertest** 副程式等，我都有將其註解掉了。

Program function

bt create_tree(bt,int);

bt 是這樣來的:

struct tree

```
{
    int data;
    struct tree *left;
    struct tree *right;
    struct tree *father;
};
```

typedef struct tree node;

```
typedef node *bt;
```

這個是傳入 **root** 位址與欲 **insert** 進入二元樹的數字，用 **Linked list** 去建立二元樹的函式。

```
void preorder(bt)
```

這是傳入 **root** 位址，印出整棵二元樹的副程式。

```
bt position(bt,int)
```

這是傳入 **root** 位址與欲刪除的節點數字，然後去走訪，讓指標指在欲刪除的節點上面，並回傳該節點位址的函式。

```
void deletenode(bt)
```

這是傳入剛剛 **position** 函式找到之位址，並將之刪除的函式，這裡面有許多狀況要特別考慮，底下會詳細說明。

How you design your program

這次的程式裡面有許多我為了 **debug** 而設置的變數與函式，我都將他們註解掉，或是沒有用到了。

其他部分(讀入資料~建立完二元樹)我在一開始就基本上都想出來也沒錯，重點是 **deledtnode** 讓我 **debug** 花了 4 天，這邊詳細說明 **deletenode** 的各種狀況。首先分為欲刪除之節點是否為為樹的 **root** 的情況。若為樹的 **root**，則去判斷其左右子樹是否為空，再將 **root** 的位置移至左子樹開頭或是右子樹開頭，將新的 **root** 的 **father** 指向 **NULL**，並 **free** 掉原本的位址。

若欲刪除的節點不為 **root**，則一樣去判斷其左右子樹是否為空，並將欲刪除的節點的父節點繞過欲刪除節點直接指向其子節點，並 **free** 掉該位址，這邊有個重點，**free** 完之後，一定要再次將每個原本指向的位址被 **free** 掉的地方重新指向 **NULL**，才不會產生問題，以下是我的 **deletenode** 函式。

```
void deletenode(bt ptr1)
```

```
{
    case6++;

    if(ptr1->father==NULL)
    {
        if(ptr1->left==NULL&&ptr1->right==NULL)
        {
            //return 0;
```

```

}else if(ptr1->left!=NULL&&ptr1->right==NULL)
{
    bt tmp=ptr1;
    ptr1=ptr1->left;
    free(tmp);
    ptr1->father=NULL;
    root=ptr1;
}else if(ptr1->left==NULL&&ptr1->right!=NULL)
{
    bt tmp=ptr1;
    ptr1=ptr1->right;
    free(tmp);
    ptr1->father=NULL;
    root=ptr1;
}else if(ptr1->left!=NULL&&ptr1->right!=NULL)
{
    bt tempptr=ptr1;
    bt tempptrleft=ptr1->left;

    case4++;
    //printf("%d ",ptr1->data);
    ptr1=ptr1->right;

    //printf("%d ",ptr1->data);
    while(ptr1->left!=NULL)
    {
        judge=1;
        ptr1=ptr1->left;
        //printf("%d\n",ptr1->data);
    }

    //printf("father:%d ",ptr1->data);

    //printf("\n");
    if(judge==0)
    {

```

```

        tempptrleft->father=ptr1;
        ptr1->left=tempptrleft;
        free(ptr1->father);
        ptr1->father=NULL;
        root=ptr1;
    }else if(judge==1)
    {
        tempptr->data=ptr1->data;
        ptr1=ptr1->father;
        free(ptr1->left);
        ptr1->left=NULL;
        root=tempptr;
    }
}
}else
{
    if(ptr1->left==NULL && ptr1->right==NULL)
    {
        case1++;
        if(ptr1->data<ptr1->father->data)
        {
            bt ptr2=ptr1;
            ptr1=ptr1->father;
            free(ptr2);
            ptr1->left=NULL;
        }else
        {
            bt ptr2=ptr1;
            ptr1=ptr1->father;
            free(ptr2);
            ptr1->right=NULL;
        }
    }

    }else if(ptr1->left==NULL && ptr1->right!=NULL)
    {

```

```

case2++;
bt temp=ptr1;
if(temp->data<temp->father->data)
{
    temp->father->left=temp->right;
}else
{
    temp->father->right=temp->right;
}
free(temp);
}else if(ptr1->right==NULL && ptr1->left!=NULL)
{
    case3++;
    bt temp=ptr1;
    if(temp->data<temp->father->data)
    {
        temp->father->left=temp->left;
    }else
    {
        temp->father->right=temp->left;
    }
    free(temp);
}else
{

```

```

    bt tempptr=ptr1;
    bt tempptrleft=tempptr->left;
    case4++;
    //printf("%d ",ptr1->data);
    ptr1=ptr1->right;
    //printf("%d ",ptr1->data);
    while(ptr1->left!=NULL)
    {
        judge=1;
        ptr1=ptr1->left;
        //printf("%d\n",ptr1->data);
    }

```

```

//printf("father:%d ",ptr1->data);

//printf("\n");
if(judge==0)
{
    ptr1->father=tempptr->father;
    tempptr->father->right=ptr1;
    tempptrleft->father=ptr1;
    ptr1->left=tempptrleft;
    free(tempptr);
}else if(judge==1)
{
    tempptr->data=ptr1->data;
    //printf("data: %d\n",tempptr->data);

    ptr1=ptr1->father;
    free(ptr1->left);
    ptr1->left=NULL;
}
//printf("%d\n",ptr1->left->data);

//printf("%d %d %d %d %d\n",case6,case1,case2,case3,case4);
}
}
}

```