(1) Result screenshot

```
Jamei@DESKTOP-PGELV1B MINGW64 /c/Users/Jamei/Desktop
$ gcc -std=c11 -o hw5 hw5.c
```

```
Jamei@DESKTOP-PGELV1B MINGW64 ~/Desktop
$ ./hw5<input0_windows.txt>output.txt
```

Screenshot of command line.

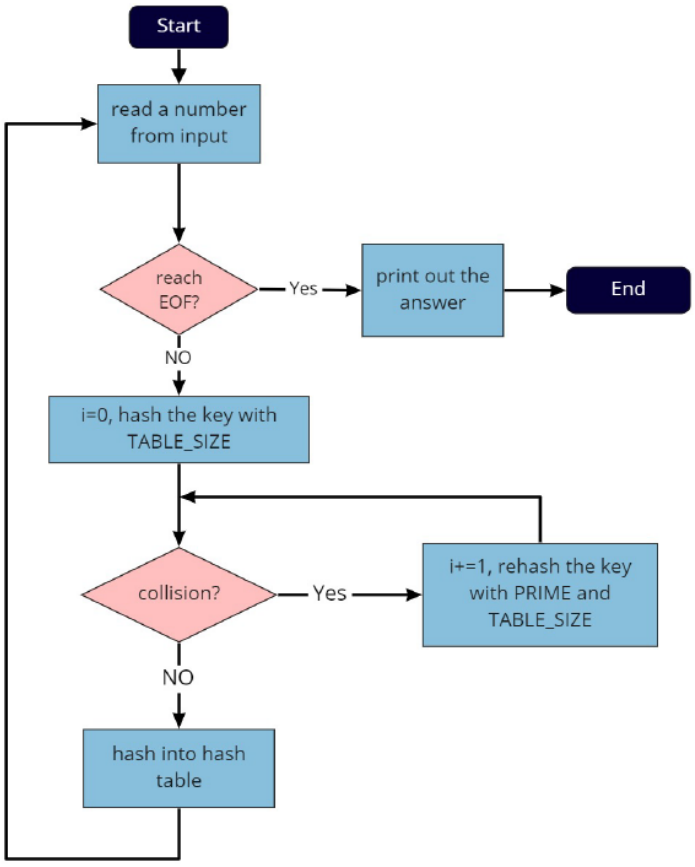
output.txt - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

```
0->77
1->98
2->2
3->37
4->56
5->31
6->45
7->85
8->8
9->70
10->10
11->24
12->64
```

Screen shot of output.txt.

(2) Program architecture



(3) Program functions

```
#define TABLE_SIZE 13
#define PRIME 7
```

Parameters

TABLE_SIZE：在題目裡面是 13，區分的空間數

PRIME：在題目裡面是 7，和 TABLE_SIZE 均可隨意更改，為了方便更改做了 MACRO define

```
int key,check[TABLE_SIZE]={0},hash[TABLE_SIZE]={0};
```

Parameters

key：準備 hash 的數

check[TABLE_SIZE]：初始值均為 0，代表沒存東西，當 hash 的 x 格存入數字時，則 check 的相對應格會變成 1。這是為了避免 key 是 0 的情況，導致無法分辨 hash 內存的值到底是初始值或是經過 hash 的

hash[TABLE_SIZE]：hash 完後的結果會存至這邊，一開始均存為 0

```
while(scanf("%d",&key)!=EOF)
//因為 input 僅有一行數字，所以可以一次讀入一數直到文件末端
{
    int temp=key%TABLE_SIZE;
//當 i=0 時，hash function 的簡化，hash2 會等於 0，只留下 hash1
    if(check[temp]==0)
//如果這個格子還沒填入任何數字，那就可以把 key hash 進去，並且 check 表的狀態會變更為已經存入數字，然後直接到一開始讀入下一個數字
    {
        hash[temp]=key;
        check[temp]=1;
        continue;
    }
    do
```

```
//當發生 collision 時，把 hash2 加進來，每次 i+1 時，其實 hash function 的結果即為原結果加上 PRIME-(key%PRIME)，最後 mod TABLE_SIZE 是當它超過 TABLE_SIZE 拾取其餘數即可。
而當 check 表內已經存過數字的話，代表 i 要再+1
    {
        temp=(temp+PRIME-(key%PRIME))%TABLE_SIZE;
    }while (check[temp]==1);
    hash[temp]=key;
    check[temp]=1;
//當終於沒有 collision 時，做法和一開始一樣
}
```

Parameters

`temp` 暫時儲存 `key` 經過 hash function 後的結果，並用來比對是否會造成 collision

```
printf("%d->%d",0,hash[0]);
//因為第一行不該換行，所以直接印出
for(int i=1;i<TABLE_SIZE;i++)
//第一個數之後，均先換行再印下一個數
{
    puts("");
    printf("%d->%d",i,hash[i]);
}
```

Return Value

當整個 main function 順利跑完，return 0 並結束程式