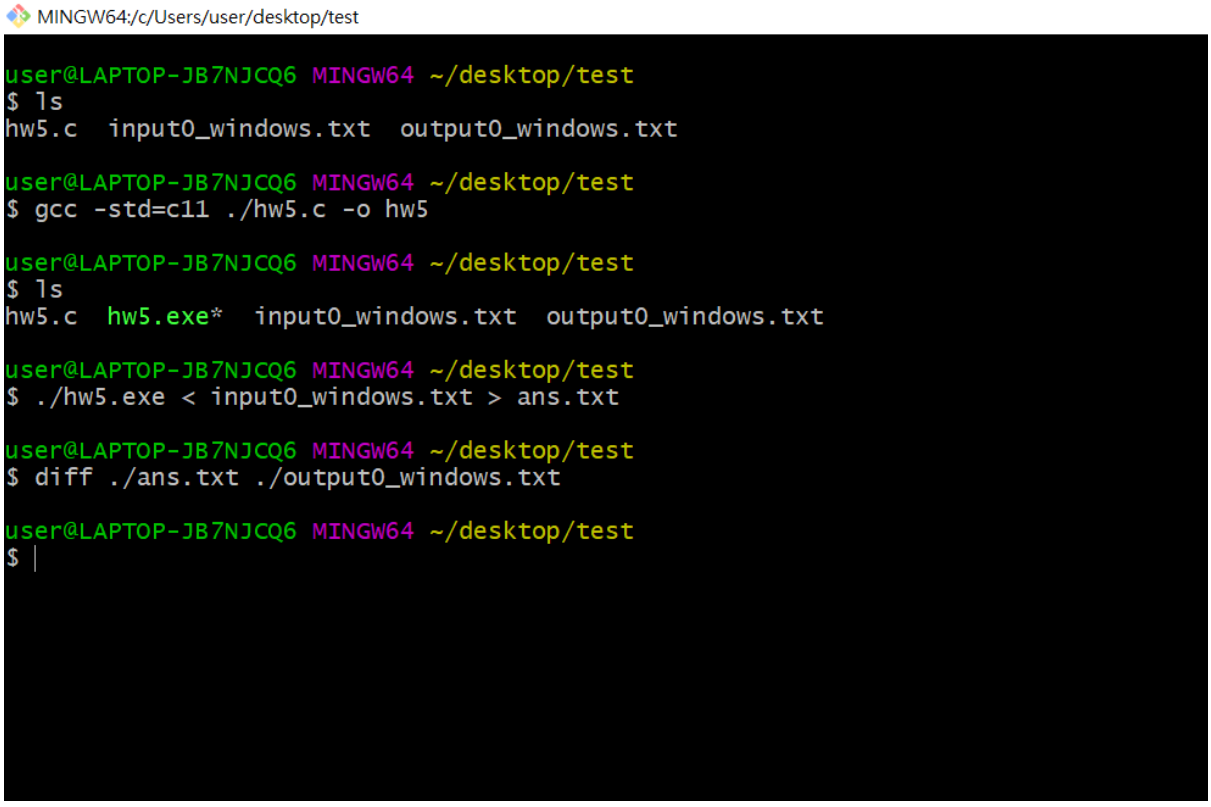
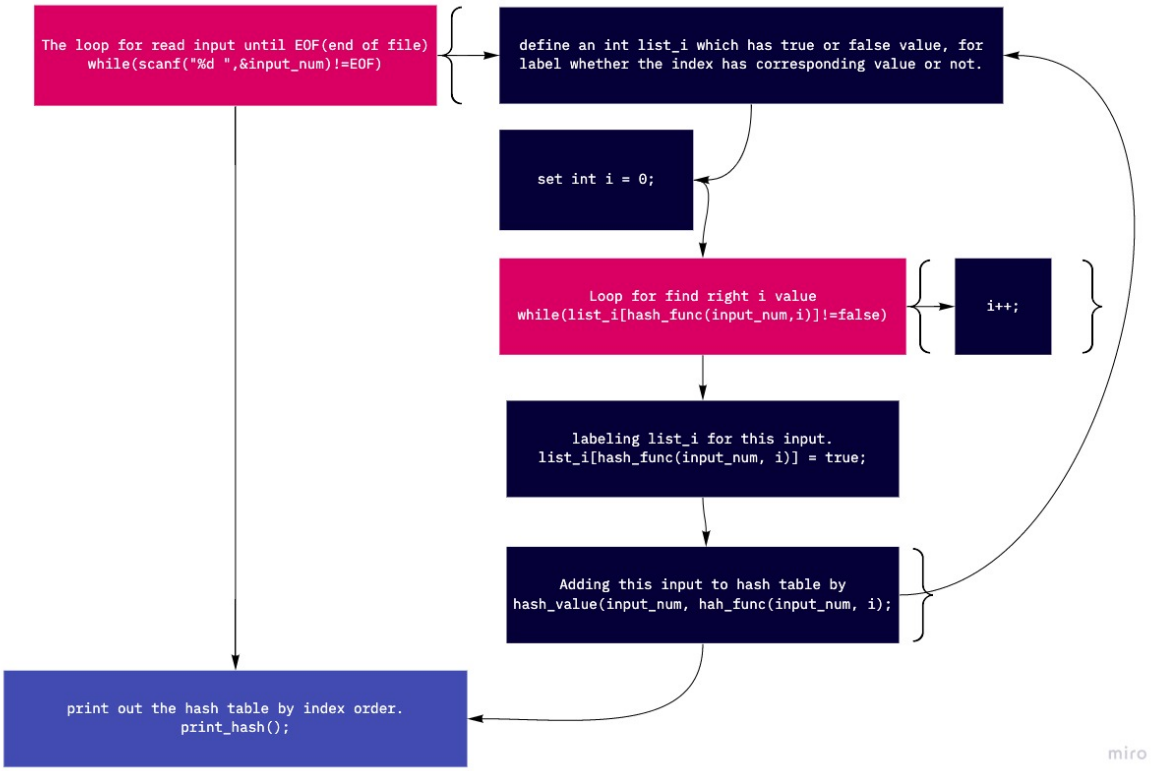
E14084117_黃子峻

資訊系 二乙

(1) result screenshot



(2) program structure



(3) program functions

`int hash1(int hash_num)`

> Part of hash function : `hash1(key) = key%TABLE_SIZE`

- parameters:
  - `int hash_num` is the number want to process by hash1() function.
- output:
  - an `int` = `hash_num%TABLE_SIZE`, TABLE_SIZE = 13 in this case.

`int hash2(int hash_num)`

> Part of hash function : `hash2(key) = PRIME - (key%PRIME)`

- parameters
  - `int hash_num` is the number want to process by hash2() function.
- output:

- an `int` = `PRIME - (hash_num % PRIME)`, PRIME = 7 in this case.

`int hash_func(int hash_num,int i)`

> Hash function : `(hash1(key) + i*hash2(key))%TABLE_SIZE`

- parameters
  - `int hash_num` is the number want to process by hash() function.
  - `int i` is set to 0 in the beginning. When a collision occurs during hashing a key, recalculate its hash value by i = i+1(not in this function). Repeat this process until no collision occurs (i.e., an empty bucket is found), and then put the key in that bucket.
- output
  - an `int` = `(hash1(hash_num) + i*hash2(hash_num))%TABLE_SIZE` ,TABLE_SIZE = 13 in this case.

`void hash_value(int hash_num , int index)`

> Adding a new number to the hash table.

- parameters
  - `int hash_num` : an `int` number get by input.
  - `int index` : an `int` index of adding value in the hash table, which get by running `hash_func()` .
- output:
  - no output value (void).

`void print_hash(void)`

> Print out the hash table by index order.

- parameters
  - no input value (void).
- output

- no output value (void).

---

(4) Self define structure

```
typedef struct hash_node hash_node;
struct hash_node
{
    int index;
    int hash_num;
    hash_node* next;
};
hash_node* front = NULL;
```

- This structure is an link list application for connecting the hash table data (`hash_num`) by index order.
- It make more convenience when print out the hash table by `index` order.
- `hash_node* front` is a pointer point to the front of this link list.

---

(5)How to dealing with i in hash value?

- In the beginning, i is set to 0. When a collision occurs during hashing a key, recalculate its hash value by i = i+1. If collision still occurs, then recalculate its hash value again by setting i = i+1. Repeat this process until no collision occurs (i.e., an empty bucket is found), and then put the key in that bucket.
- I create an `int list_i = {0};` to mark whether the index has an value or not(mark 0 for no value and 1 has value), and I include `<stdbool.h>` to convert 0 to false and 1 for true.
- by using `while(list_i[hash_func(input_num,i)]!=false)i++;` the loop will running till the right i value is found.(found index has no value.)