Programming Practice

2018-11-08

Week 10

Notice

Last week HW

- Due date has been extended.

- Due to problems in the server, some submission records for last week disappeared.
- Submit last week & this week's HW by 11/13 14:00pm.

- Everyone submit last week HW to the submission page, not email.
- No late policy applied to last week HW.

Attach code as a file

- When sending email to TA,
  please attach code as a file (ex. `yourCode.c`)
  - Not screenshot.
  - Also, please don't drag and copy/paste (-> includes line numbers).

- Easier to detect bugs.
- Quicker response.

For this week

- Please use either Lab computer(Linux) or Martini server(Putty/SSH).
- Not on local Mac.

Practice Lecture

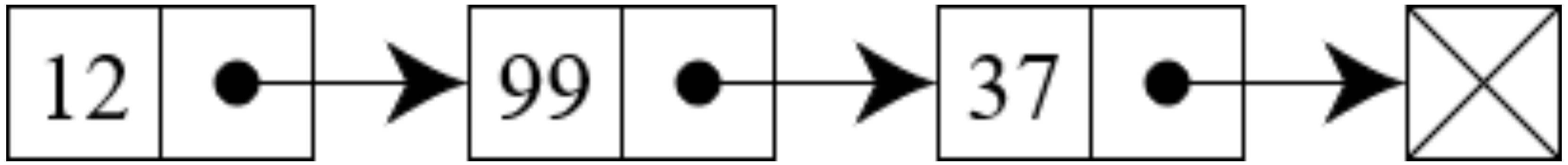
Linked List

- This week, we'll practice creating Linked List & basic functions.

- Inserting an element
- Searching an element
- Deleting an element

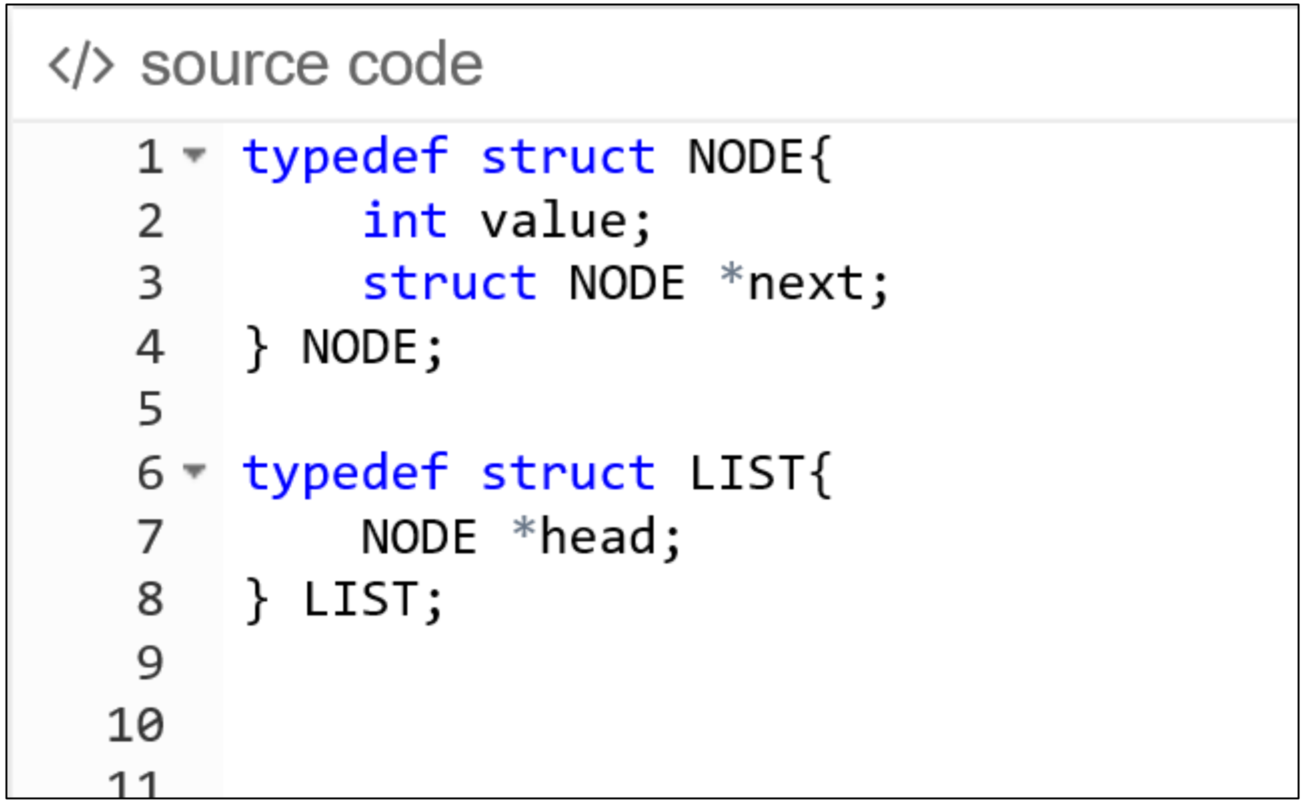
Linked List

- `struct NODE`
- `struct LIST`



struct NODE & struct LIST

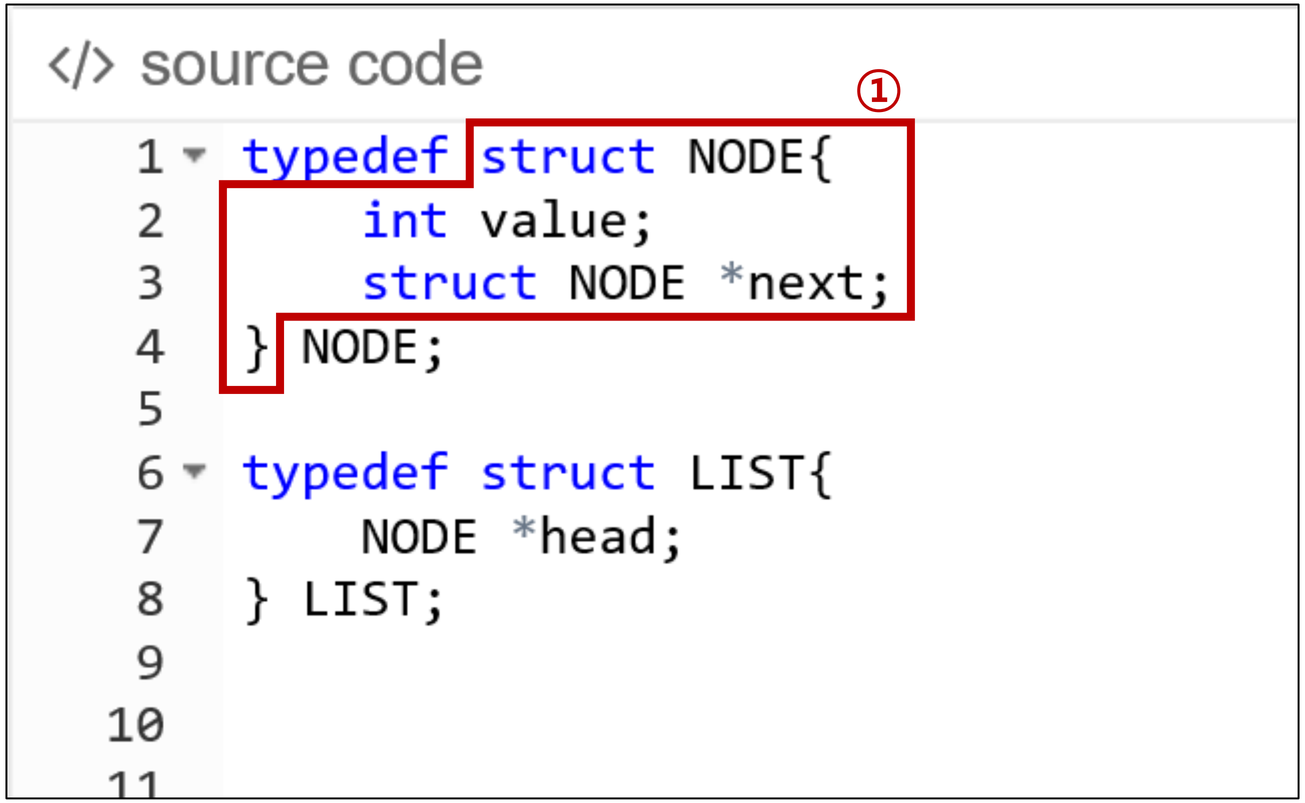
- Will be provided:



```c
typedef struct NODE{
    int value;
    struct NODE *next;
} NODE;

typedef struct LIST{
    NODE *head;
} LIST;
```

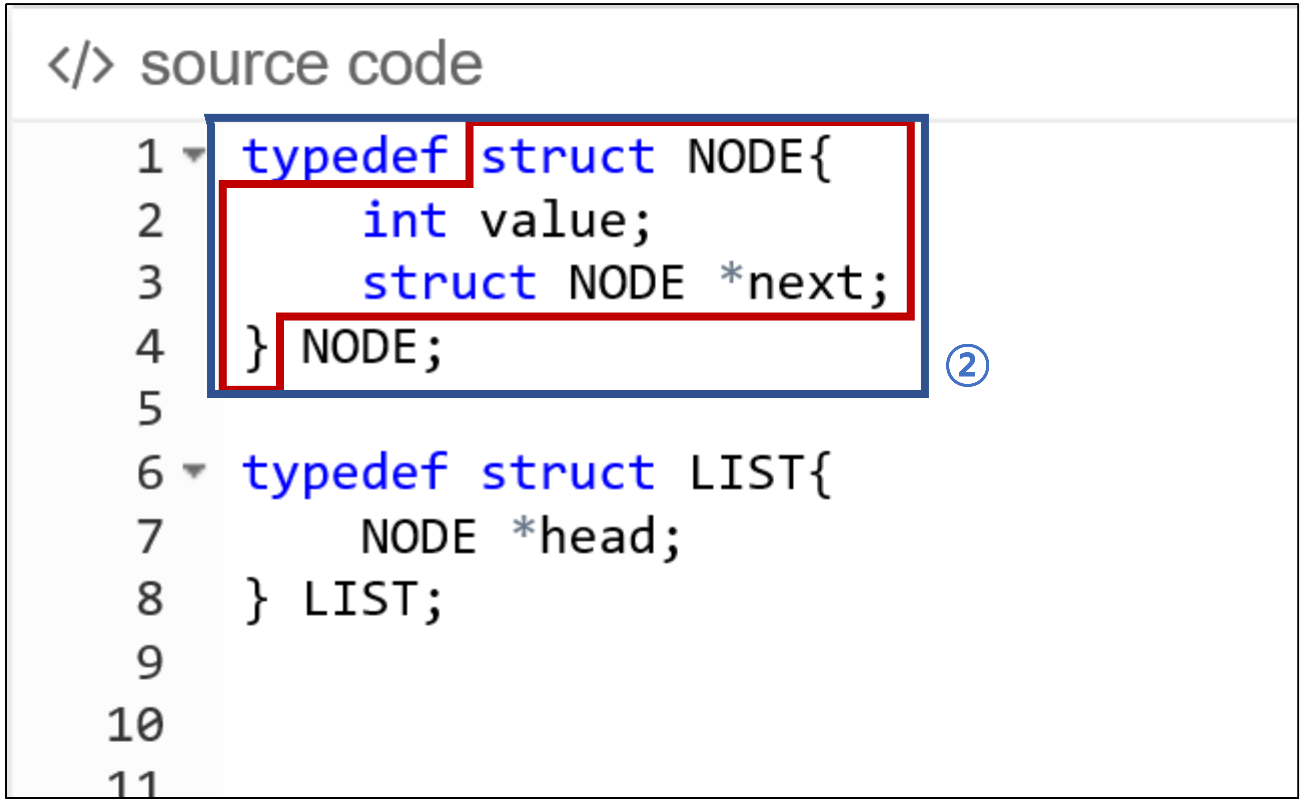
struct NODE & struct LIST



```
typedef struct NODE{
    int value;
    struct NODE *next;
} NODE;

typedef struct LIST{
    NODE *head;
} LIST;
```

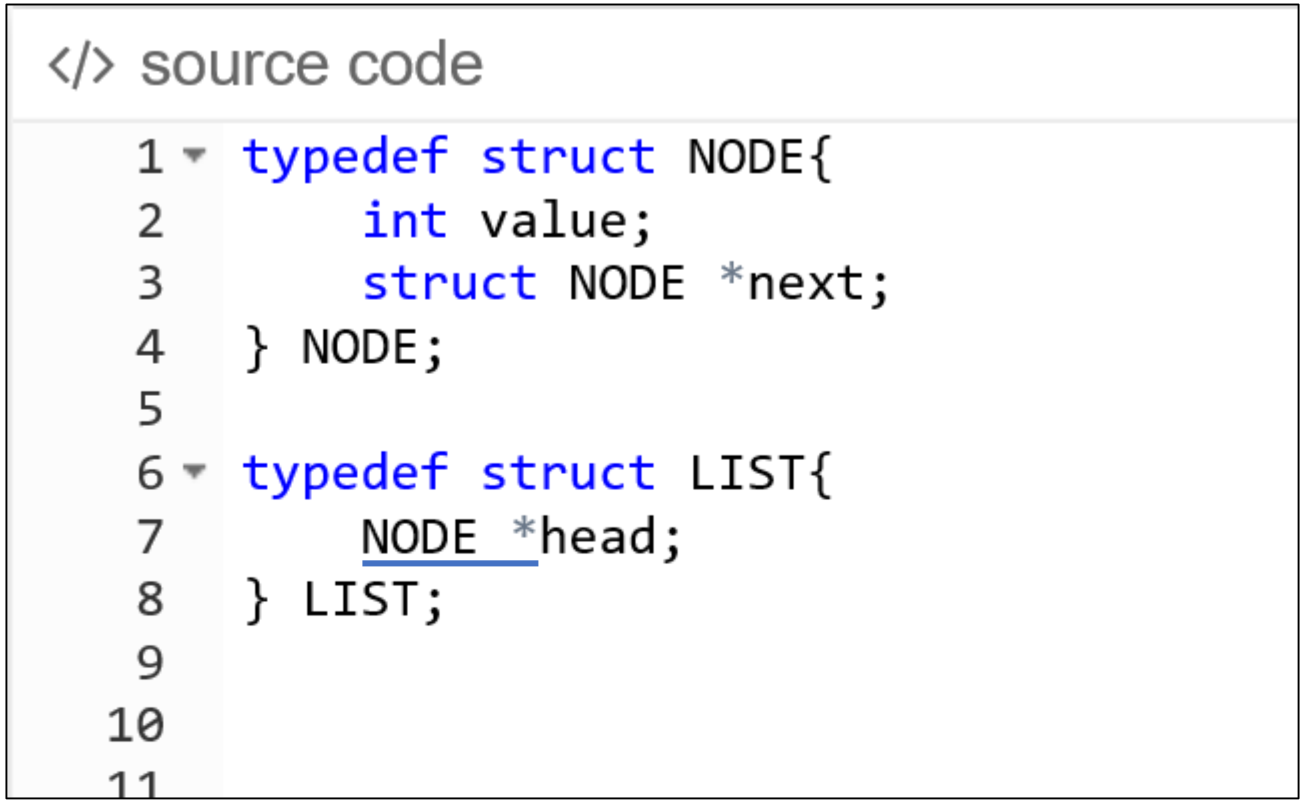
struct NODE & struct LIST



```c
typedef struct NODE{
	int value;
	struct NODE *next;
} NODE;

typedef struct LIST{
	NODE *head;
} LIST;
```

struct NODE & struct LIST



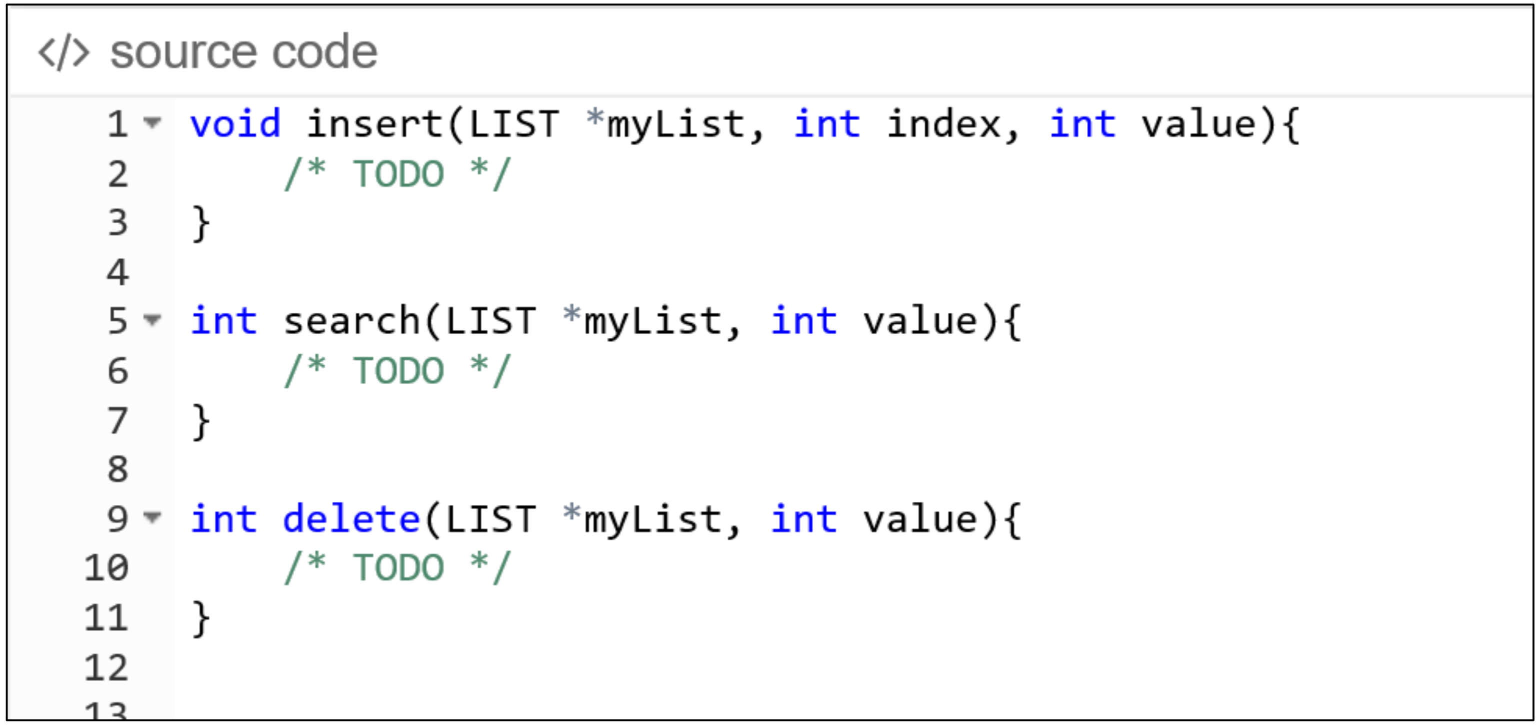
</> source code

```c
typedef struct NODE{
    int value;
    struct NODE *next;
} NODE;

typedef struct LIST{
    NODE *head;
} LIST;
```

Your job is to write

- `insert`, `search`, and `delete` functions



</> source code

```c
void insert(LIST *myList, int index, int value){
    /* TODO */
}

int search(LIST *myList, int value){
    /* TODO */
}

int delete(LIST *myList, int value){
    /* TODO */
}

```

*** For this week, all index will be 0-based.**

Homework Problems

1. Linked List Insert
2. Linked List Search
3. Linked List Delete

For this week's HW

- Submit code containing just the function (insert/search/delete).
- Don't include `main()` in your submissions.

- This applies to all three problems!

Problem. **1**

Linked List Insert

Submit your code file as `insert.c`

Description

Insert a *value*, into the position specified by *index*, to the list *myList*.

(This means you have to create a new node when this function is called.)

Assume that *index* will always be given as $0 \leq$ *index* $\leq$ (*current length of myList*).

(*index*==0 means insert to very front. *index*==(*current length of myList*) means insert to very end.)

Function prototype:

```
void insert(LIST *myList, int index, int value);
```

Arguments:

`LIST *myList` : Points to the list to do the inserting operation to.

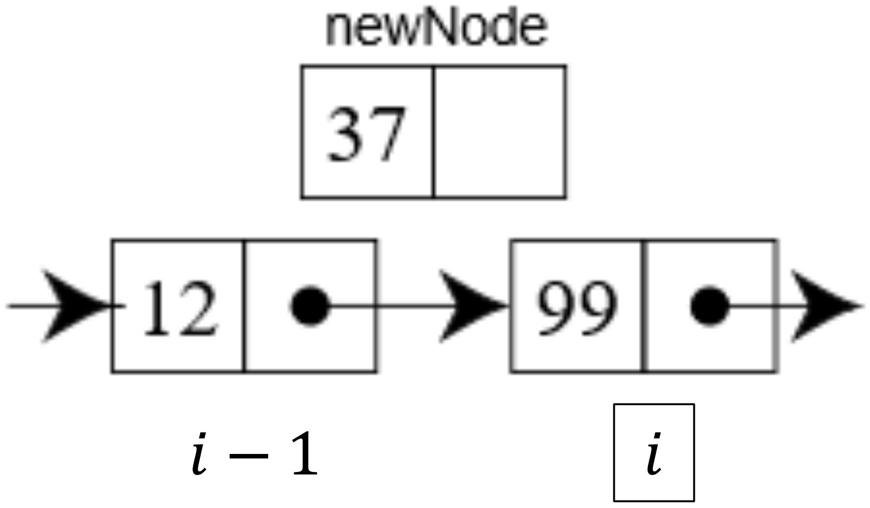
`int index` : The position to insert the new node into.

`int value` : The value for the new node to hold.

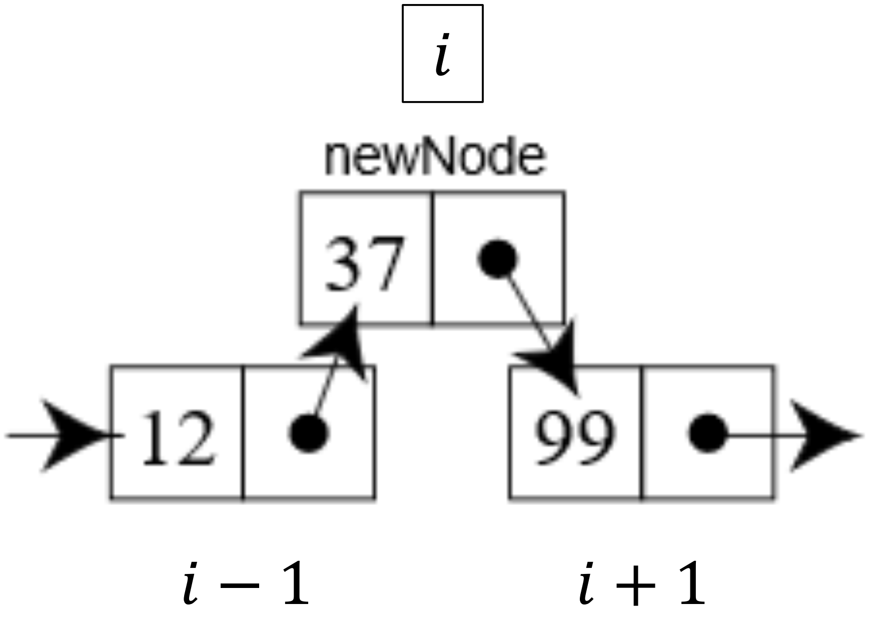
* No return for this function

Inserting to index i

Before



After



Problem. **2**

Linked List Search

Submit your code file as `search.c`

Description

Search if a *value* exists inside the list *myList*, and return the index if it is found.

Function prototype:

```
int search(LIST *myList, int value);
```

Arguments:

`LIST *myList` : Points to the list on which to do the search operation.

`int value` : The value to search for.

Return:

If found, return the (0-based) index of **the first occurrence** of *value*. If not found, return -1.

Problem. **3**

Linked List Delete

Submit your code file as **`delete.c`**

Description

Find and remove **the first occurrence** of a node holding *value,* inside the list *myList*.

Function prototype:

```
int delete(LIST *myList, int value);
```

Arguments:

`LIST *myList` : Points to the list on which to do the delete operation.

`int value` : The value to look for, in order to delete the node that is holding it.

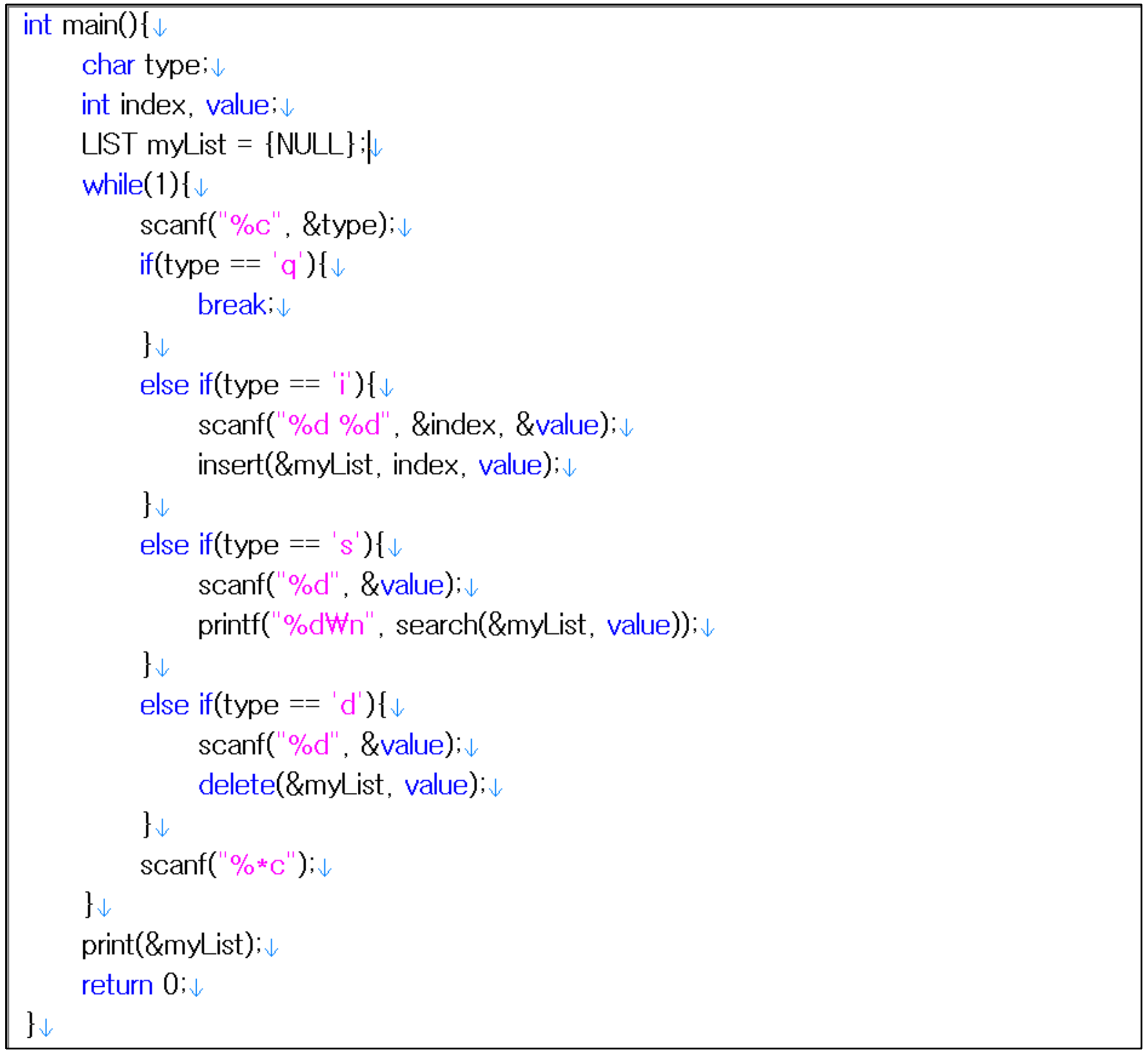
Return:

If successful (= a node with *value* was found and deleted), return 0.

If unsuccessful (= node with *value* does not exist), return -1.

main()

- Will also be provided:



```
int main(){
    char type;
    int index, value;
    LIST myList = {NULL};
    while(1){
        scanf("%c", &type);
        if(type == 'q'){
            break;
        }
        else if(type == 'i'){
            scanf("%d %d", &index, &value);
            insert(&myList, index, value);
        }
        else if(type == 's'){
            scanf("%d", &value);
            printf("%d₩n", search(&myList, value));
        }
        else if(type == 'd'){
            scanf("%d", &value);
            delete(&myList, value);
        }
        scanf("%*c");
    }
    print(&myList);
    return 0;
}
```

How this main() works

Continuously receives input, and does the requested operation.

There are 3 types of request. **`i`**`nsert`, **`s`**`earch`, **`d`**`elete`

Formats are as follows.

```
i (index) (value)
s (value)
d (value)
```

Example)	
`i 0 7`	$\longrightarrow$ Insert 7 at index 0
`i 1 4`	$\longrightarrow$ Insert 4 at index 1
`s 6`	$\longrightarrow$ Search 6
`d 4`	$\longrightarrow$ Delete 4
`q`	$\longrightarrow$ Finish with ‘**q**’

index for insert operation must be in the correct range $\left(0 \leq index \leq (current\ length)\right)$.

Skeleton Code

File Name	Description
main.c	Reads input & prints output. It will call your functions such as insert, delete.
list.h	Header file. Your functions should fit with the prototypes declared in this file.
insert.c	TODO :: Implement **insert** function.
delete.c	TODO :: Implement **delete** function.
search.c	TODO :: Implement **search** function.
TA_obj/	Directory that contains object files of TA's implementation.
Makefile	Compile macros are defined here.
testin.txt	Example test input. You may change test input if you want.

Makefile

- Build automation tool. Makes compile & running programs easier.

- You don’t have to understand it.
- It will help you to compile and test your program.

- You can simply open this file with `vim Makefile`
- If you are not familiar with this, Do NOT modify it.

How to use

- Three commands, each to test each function.

```
> make itest
> make stest
> make dtest
```

Test command

```
> make itest
```

: Compiles your **`insert.c`** with TA's search, delete files.

Check your output file: **`myout.txt`**

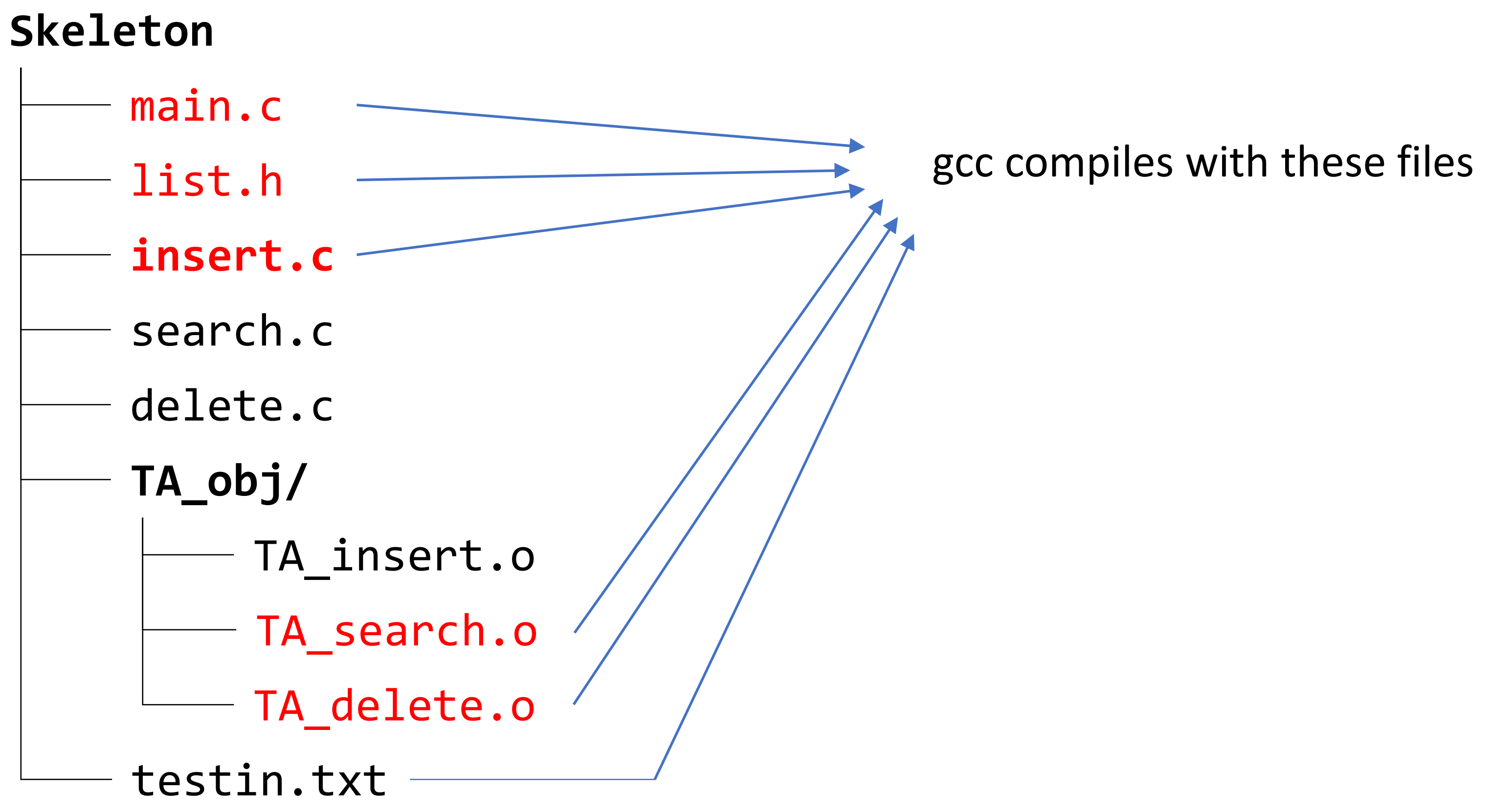
"It will execute following commands automatically."

```
> gcc –o reference main.c TA_insert.o TA_search.o TA_delete.o
> ./reference < testin.txt > refout.txt
> gcc -o runinsert main.c insert.c TA_search.o TA_delete.o
> ./runinsert < testin.txt > myout.txt
> diff -Z --report-identical-files refout.txt myout.txt
```

→ Compile reference program
→ Generate reference output
→ Compile your program
→ Generate your output
→ Compare two outputs

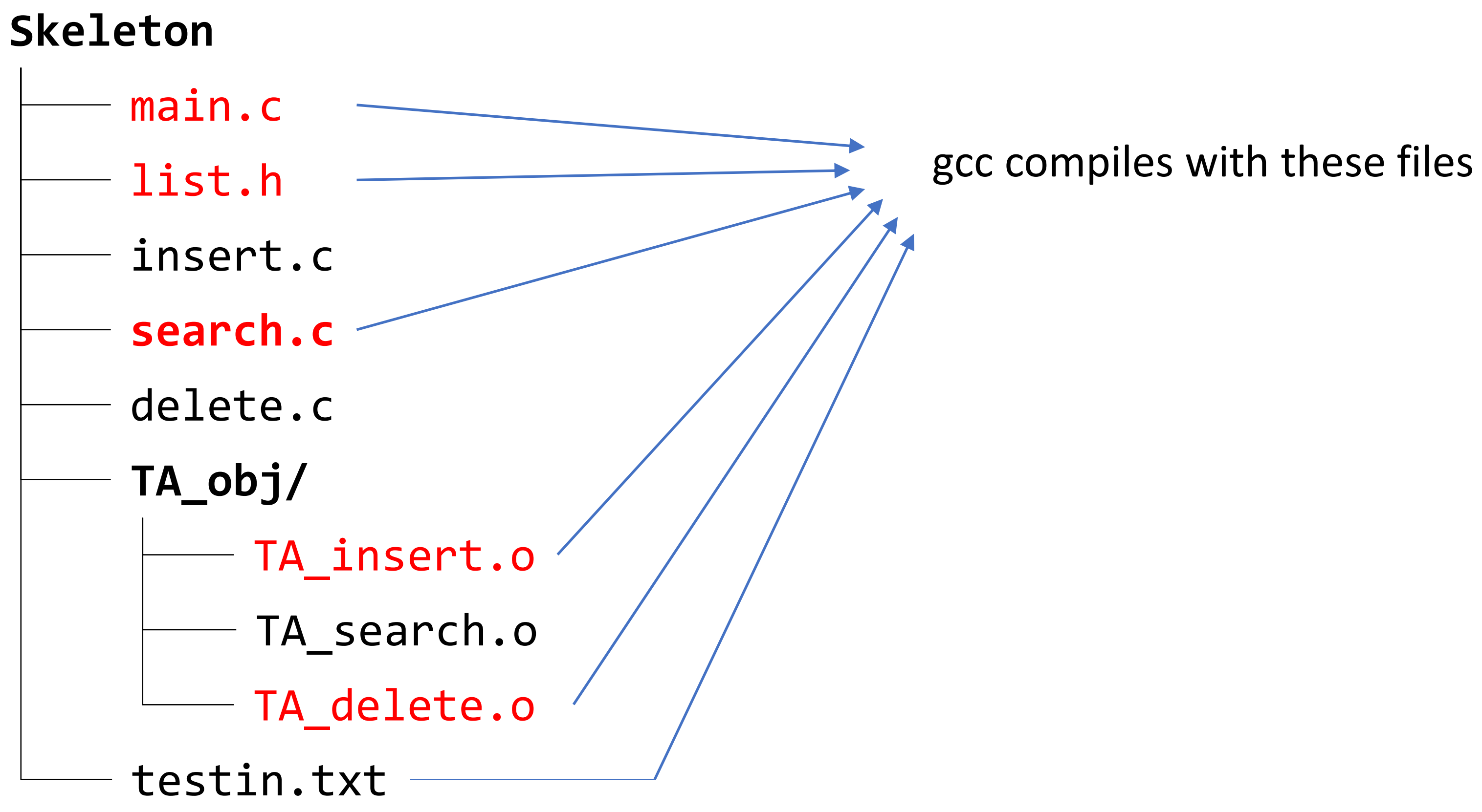
Test command

When you type **> make itest**



Test command

When you type **> make stest**



Test command

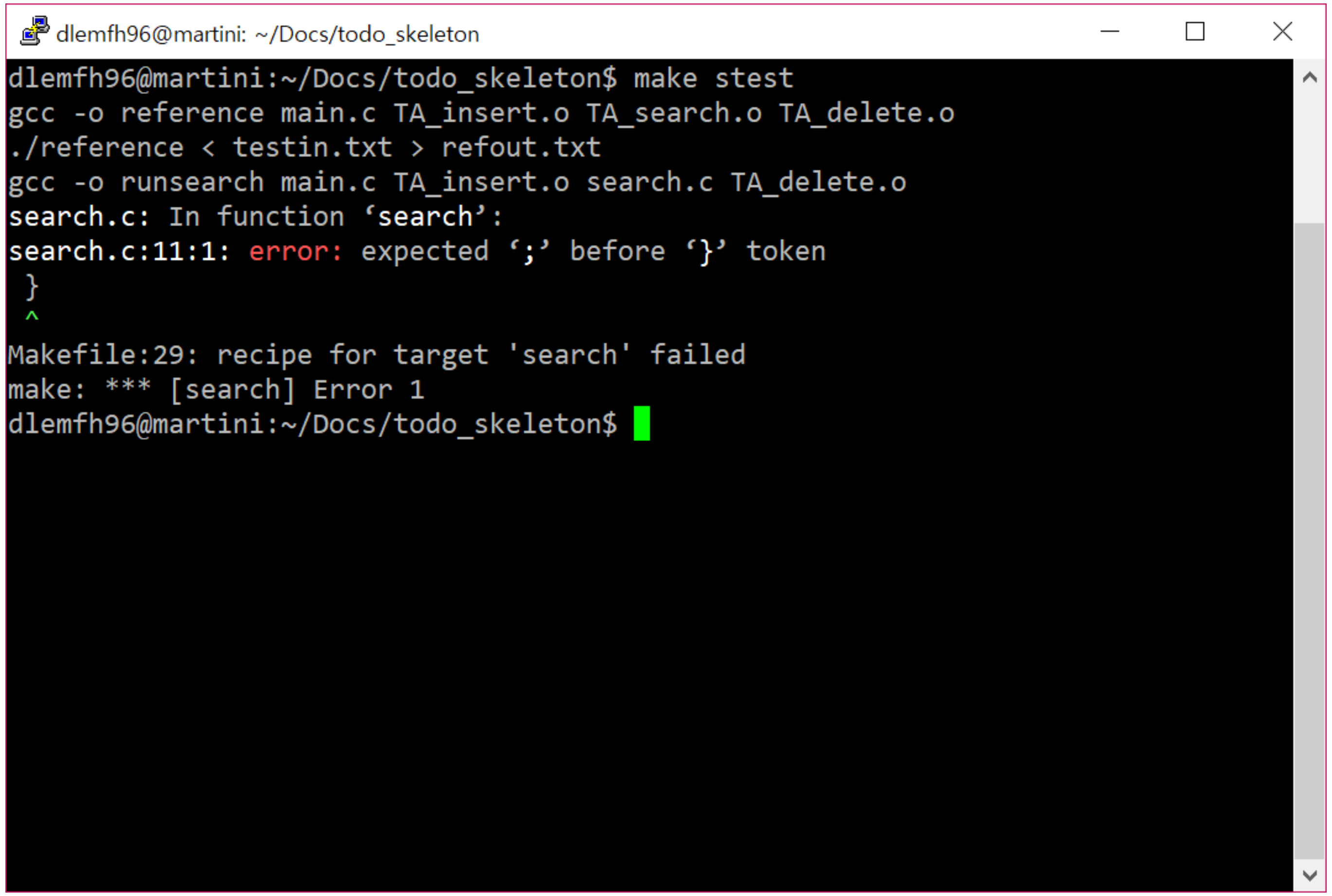
- Testing search and delete functions is the same:

```
> make stest
> make dtest
```

It will test your function same way as with insert.

Reading the result screen

- Compile error:

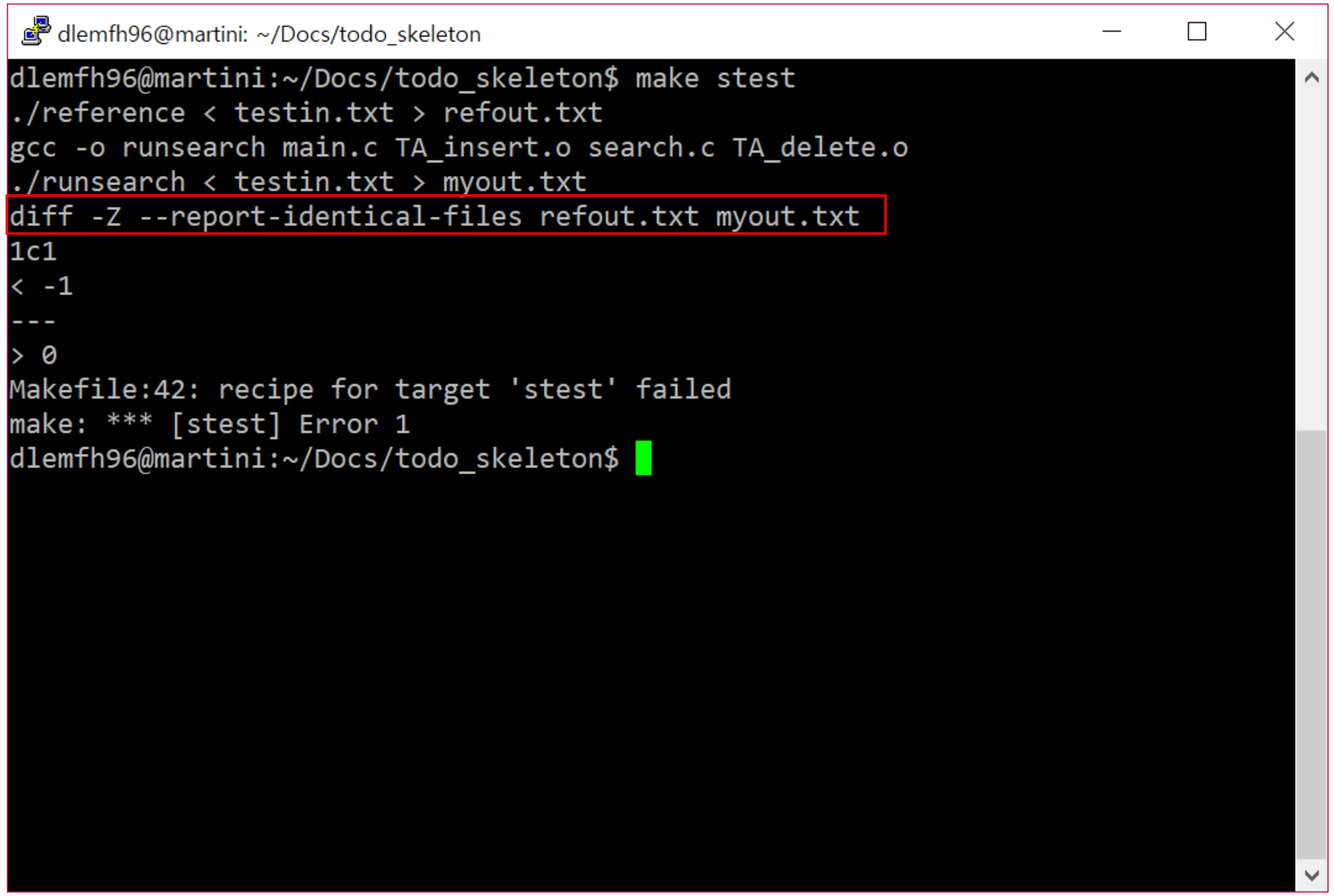


dlemfh96@martini: ~/Docs/todo_skeleton

```
dlemfh96@martini:~/Docs/todo_skeleton$ make stest
gcc -o reference main.c TA_insert.o TA_search.o TA_delete.o
./reference < testin.txt > refout.txt
gcc -o runsearch main.c TA_insert.o search.c TA_delete.o
search.c: In function ‘search’:
search.c:11:1: error: expected ‘;’ before ‘}’ token
 }
 ^
Makefile:29: recipe for target 'search' failed
make: *** [search] Error 1
dlemfh96@martini:~/Docs/todo_skeleton$ 
```

Reading the result screen

- Compiles okay but
  Wrong answer:

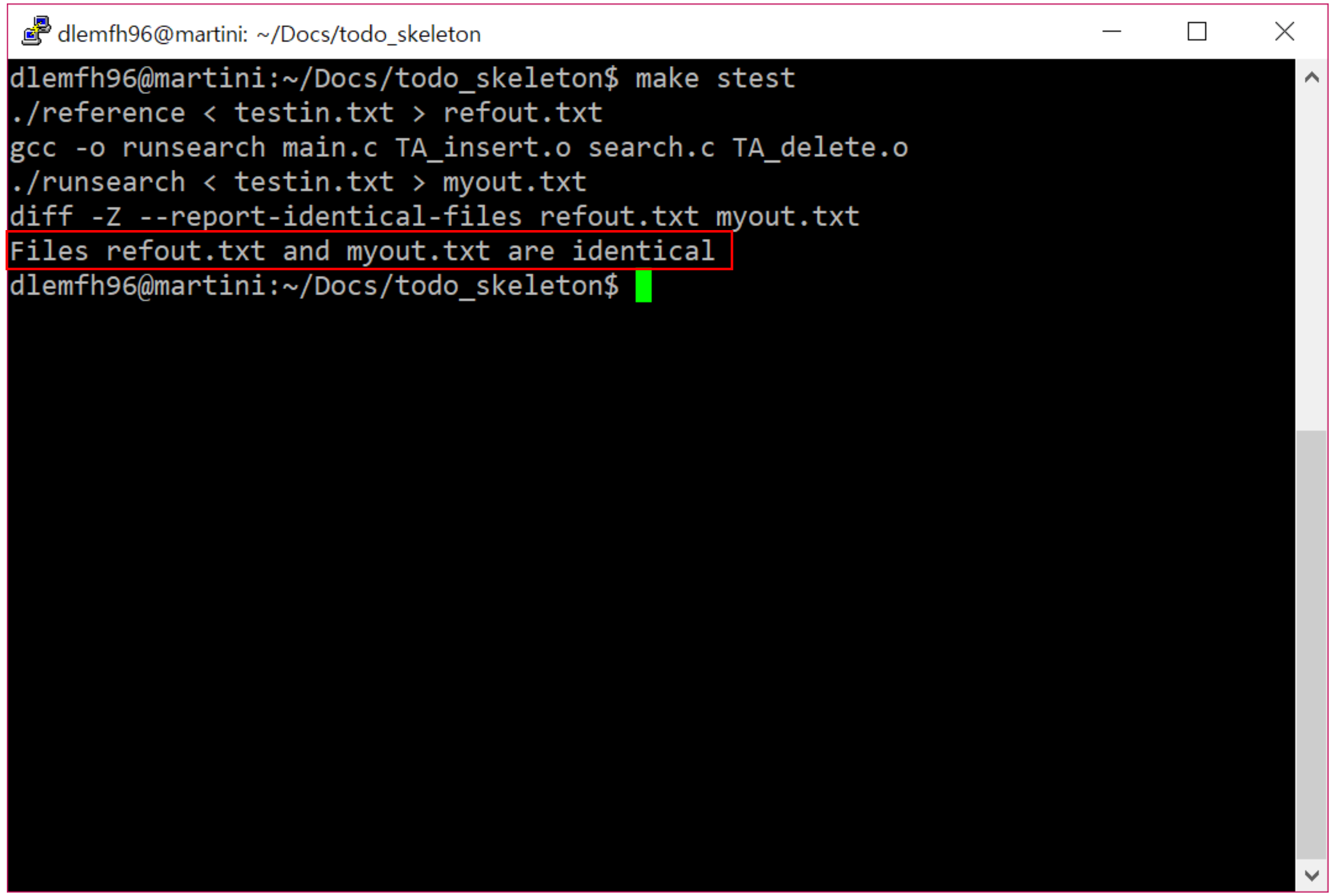


dlemfh96@martini: ~/Docs/todo_skeleton

```
dlemfh96@martini:~/Docs/todo_skeleton$ make stest
./reference < testin.txt > refout.txt
gcc -o runsearch main.c TA_insert.o search.c TA_delete.o
./runsearch < testin.txt > myout.txt
diff -Z --report-identical-files refout.txt myout.txt
1c1
< -1
---
> 0
Makefile:42: recipe for target 'stest' failed
make: *** [stest] Error 1
dlemfh96@martini:~/Docs/todo_skeleton$
```

Reading the result screen

- Correct answer:


dlemfh96@martini: ~/Docs/todo_skeleton

```
dlemfh96@martini:~/Docs/todo_skeleton$ make stest
./reference < testin.txt > refout.txt
gcc -o runsearch main.c TA_insert.o search.c TA_delete.o
./runsearch < testin.txt > myout.txt
diff -Z --report-identical-files refout.txt myout.txt
Files refout.txt and myout.txt are identical
dlemfh96@martini:~/Docs/todo_skeleton$
```

Test command (recap)

```
> make insert
> make search
> make delete
```

- You can also use "**`make clean`**" to remove unnecessary files.

Input file format

You can make your own input file to test your program.

There are 3 types of request. **`i`**`nsert`, **`s`**`earch`, **`d`**`elete`

Formats are as follows.

```
i (index) (value)
s (value)
d (value)
```

Example)

`i 0 7`	$\longrightarrow$	Insert 7 at index 0
`i 1 4`	$\longrightarrow$	Insert 4 at index 1
`s 6`	$\longrightarrow$	Search 6
`d 4`	$\longrightarrow$	Delete 4
`q`	$\longrightarrow$	Input file should finish with '**q**'

Filename should be ' **`testin.txt`**'

index for insert operation must be in the correct range $(0 \le index \le (current\ length))$.