

## ■ 실습보고서 2

교과목명	시스템프로그래밍		분반	001
제출일자	2021-10-18		교수자명	장희숙
팀명	이름	학번	이름	학번
초코파이썬칩	김기태	20173152		
	안대현	20173217		

### ■ 실습 주제

실습 2. 프로그래밍 환경, 디버깅 및 기본 함수

### ■ 실습 결과

#### 1. (실습준비 7) GNU C 라이브러리(glibc)에 대하여 조사하고 요약하여 보고서에 정리하여 본다.

GNU C 라이브러리는 일반적으로 glibc라고 불린다. GNU 프로젝트가 C 표준 라이브러리를 구현한 것으로 현재는 C++도 지원한다.

ISO C11, POSIX.1-2008, BSD, OS 별 API 등을 포함한 API를 제공한다.

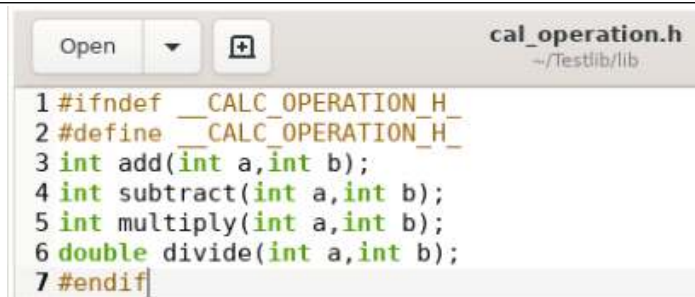
대표적으로 open, read, write, malloc, printf, getaddrinfo, dlopen, pthread\_create, crypt, login, exit 등 기본 기능을 지원한다.

#### 실습 순서

#### 1. 사칙연산 (덧셈 add, 뺄셈 subtract, 곱셈 multiply, 나눗셈 divide)에 대한 함수와 이 함수들을 이용하는 예제 프로그램을 각각 작성하고, 정적 라이브러리, 공유 라이브러리, 동적 라이브러리 방식을 각각 이용하여 예제 프로그램을 실행시킨 결과를 보이시오.

다음 실습을 준비하기 전 mkdir 명령어를 이용하여 Testlib 폴더를 생성하였다. 또한 라이브러리만 들어있는 폴더 lib를 생성하였다.

1) cal\_operation.h



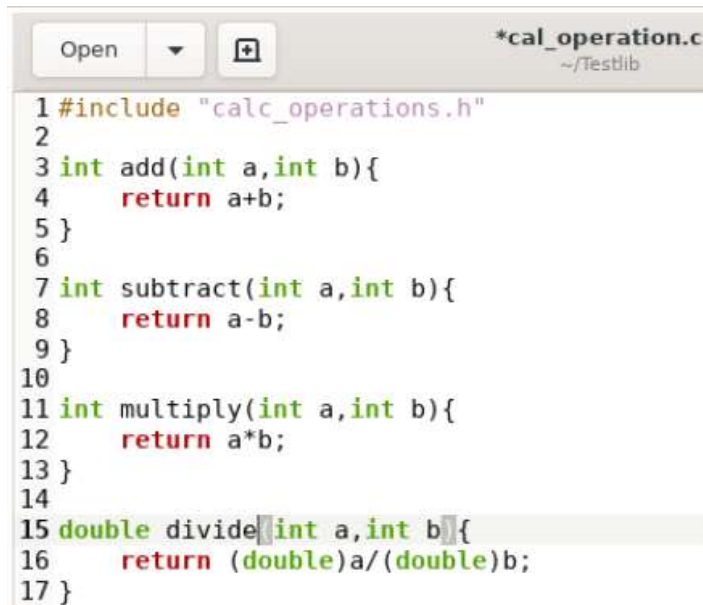
```

1 #ifndef __CALC_OPERATION_H
2 #define __CALC_OPERATION_H
3 int add(int a, int b);
4 int subtract(int a, int b);
5 int multiply(int a, int b);
6 double divide(int a, int b);
7 #endif

```

~/Testlib/lib에 함수를 정의하는 파일 cal\_operation.h을 gedit을 이용하여 작성한다.

## 2) cal\_operation.c



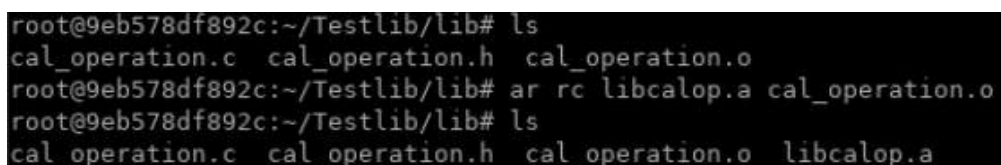
```

1 #include "calc_operations.h"
2
3 int add(int a, int b){
4     return a+b;
5 }
6
7 int subtract(int a, int b){
8     return a-b;
9 }
10
11 int multiply(int a, int b){
12     return a*b;
13 }
14
15 double divide(int a, int b){
16     return (double)a/(double)b;
17 }

```

사칙연산 함수를 구현하는 cal\_operation.c 파일을 작성한 뒤 gcc -c cal\_operation.c를 통해 목적 파일을 생성한다.

## 3) 정적 라이브러리 아카이브 제작



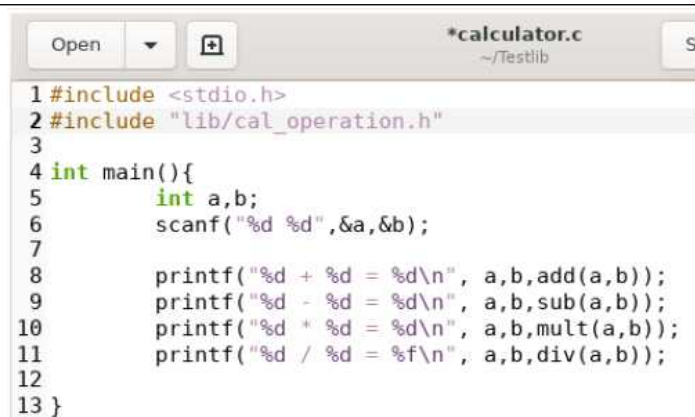
```

root@9eb578df892c:~/Testlib/lib# ls
cal_operation.c  cal_operation.h  cal_operation.o
root@9eb578df892c:~/Testlib/lib# ar rc libcalop.a cal_operation.o
root@9eb578df892c:~/Testlib/lib# ls
cal_operation.c  cal_operation.h  cal_operation.o  libcalop.a

```

lib폴더에서 ar rc lib<라이브러리명>.a <목적파일> 명령어를 통해 정적 라이브러리를 제작한다. ls 명령어를 통해 libcalop.a가 생성됐음을 확인할 수 있다.

## 4) calculator.c



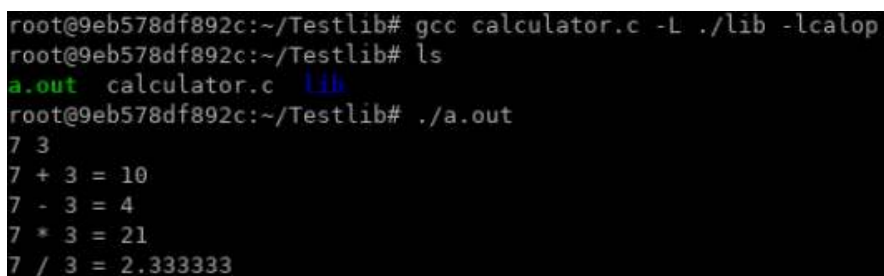
```

1 #include <stdio.h>
2 #include "lib/cal_operation.h"
3
4 int main(){
5     int a,b;
6     scanf("%d %d",&a,&b);
7
8     printf("%d + %d = %d\n", a,b,add(a,b));
9     printf("%d - %d = %d\n", a,b,sub(a,b));
10    printf("%d * %d = %d\n", a,b,mult(a,b));
11    printf("%d / %d = %f\n", a,b,div(a,b));
12
13 }

```

상위 디렉토리 Testlib에서 제작한 라이브러리를 사용하기 위한 calculator.c를 작성한다.

#### 5) 정적 라이브러리 컴파일



```

root@9eb578df892c:~/Testlib# gcc calculator.c -L ./lib -lcalop
root@9eb578df892c:~/Testlib# ls
a.out calculator.c lib
root@9eb578df892c:~/Testlib# ./a.out
7 3
7 + 3 = 10
7 - 3 = 4
7 * 3 = 21
7 / 3 = 2.333333

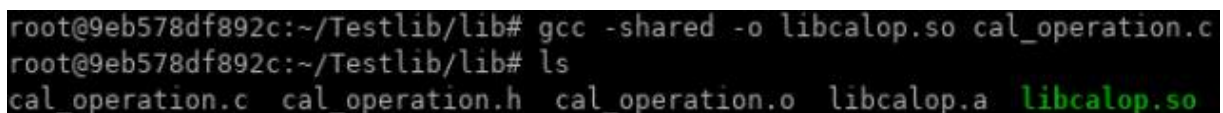
```

gcc calculator.c -L ./lib -lcalop 명령어를 통해 실행파일인 a.out을 생성하여 실행한다. 사용자 입력에 따라 사칙연산 결과가 나오는 것을 확인할 수 있다.

-L : 라이브러리 경로

-l : 라이브러리 명, 대체로 라이브러리명은 lib~로 시작하기 때문에 -l~로 대체한다.

#### 6) 공유 라이브러리 제작



```

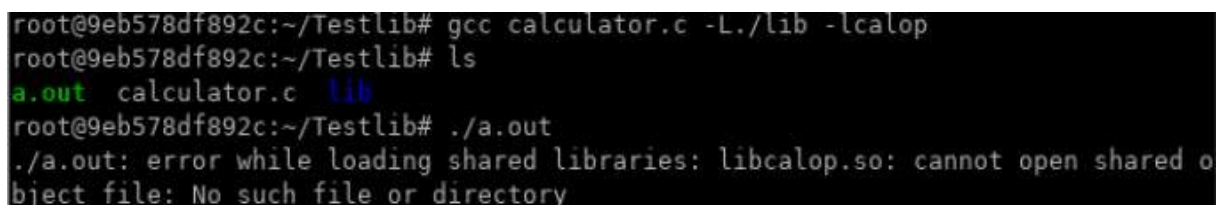
root@9eb578df892c:~/Testlib/lib# gcc -shared -o libcalop.so cal_operation.c
root@9eb578df892c:~/Testlib/lib# ls
cal_operation.c cal_operation.h cal_operation.o libcalop.a libcalop.so

```

gcc -shared -o libcalop.so cal\_operation.c 명령어를 통해 libcalop이라는 이름의 so확장자인 공유 라이브러리를 제작한다.

-shared : 공유라이브러리를 제작한다는 옵션.

#### 7) 공유 라이브러리 컴파일



```

root@9eb578df892c:~/Testlib# gcc calculator.c -L./lib -lcalop
root@9eb578df892c:~/Testlib# ls
a.out calculator.c lib
root@9eb578df892c:~/Testlib# ./a.out
./a.out: error while loading shared libraries: libcalop.so: cannot open shared object file: No such file or directory

```

상위 라이브러리인 Testlib로 이동하여 gcc calculator.c -L./lib -lcalop을 통해 컴파일한다. 실행파일인 a.out이 제작되어 있음을 확인 후 실행하면 오류가 나옴을 알 수 있다. 라이브러리의 경로를 찾을 수 없어 생기는 오류이다.

```

root@9eb578df892c:~/Testlib/lib# pwd
/root/Testlib/lib
root@9eb578df892c:~/Testlib/lib# cd ..
root@9eb578df892c:~/Testlib# gcc calculator.c -L./lib -Wl,--rpath=/root/Testlib/lib -lcalop
root@9eb578df892c:~/Testlib# ls
a.out  calculator.c  lib
root@9eb578df892c:~/Testlib# ./a.out
8 3
8 + 3 = 11
8 - 3 = 5
8 * 3 = 24
8 / 3 = 2.666667

```

이전에 제작한 공유 라이브러리의 위치를 알기 위해 pwd 명령어를 통해 경로를 파악한 후 상위 폴더에서 gcc calculator.c -L./lib -Wl,--rpath=<경로> -lcalop을 통해 경로를 정의해준다. 이후 실행파일 a.out을 실행하면 사칙연산이 실행되는 것을 확인할 수 있다.

## 8) 동적 라이브러리

```

#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

int main(){
    void *handle;
    int (*add)(int, int);
    int (*subtract)(int, int);
    int (*multiply)(int, int);
    double (*divide)(int, int);
    char *error;

    handle = dlopen("./lib/libcalop.so", RTLD_LAZY);
    if(!handle){
        fputs(dlerror(), stderr);
        exit(1);
    }

    add = dlsym(handle, "add");
    if((error = dlerror()) != NULL) {
        fprintf(stderr, "%s", error);
        exit(1);
    }

    // -----종락-----
    int a,b;
    scanf("%d %d",&a,&b);

    printf("%d + %d = %d\n", a,b,(*add)(a,b));
    printf("%d - %d = %d\n", a,b,(*subtract)(a,b));
    printf("%d * %d = %d\n", a,b,(*multiply)(a,b));
    printf("%d / %d = %f\n", a,b,(*divide)(a,b));
    dlclose(handle);
}

```

```

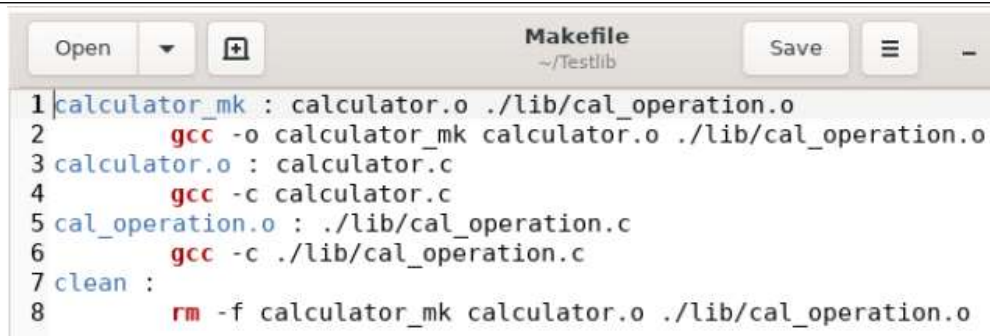
root@9eb578df892c:~/Testlib# gcc -rdynamic cal_dl.c -ldl
root@9eb578df892c:~/Testlib# ls
a.out  cal_dl.c  calculator.c  lib
root@9eb578df892c:~/Testlib# ./a.out
8 4
8 + 4 = 12
8 - 4 = 4
8 * 4 = 32
8 / 4 = 2.000000

```

동적 라이브러리를 제작하기 위해 cal\_dl.c 파일을 작성한다. dlopen 함수를 이용하여 전에 제작한 라이브러리를 적재하며, dlsym 함수를 통해 라이브러리에 선언된 심볼을 찾는다. 다음과 같이 작성 후 gcc -rdynamic cal\_dl.c -ldl 명령어를 통해 동적 라이브러리 실행 파일을 만든 뒤 실행시켜 테스트한다.

**2. 위 문제에 대하여, C 소스 파일들을 목적 파일로 만들고, 또한 이 파일들을 이용하여 라이브러리 또는 실행 파일로 만들어주는 Makefile을 작성하여 실행시켜 보시오. 이때, 라이브러리 관련 파일들은 서브디렉토리에 두도록 한다.**

### 1) Makefile 작성



```

1 calculator_mk : calculator.o ./lib/cal_operation.o
2     gcc -o calculator_mk calculator.o ./lib/cal_operation.o
3 calculator.o : calculator.c
4     gcc -c calculator.c
5 cal_operation.o : ./lib/cal_operation.c
6     gcc -c ./lib/cal_operation.c
7 clean :
8     rm -f calculator_mk calculator.o ./lib/cal_operation.o

```

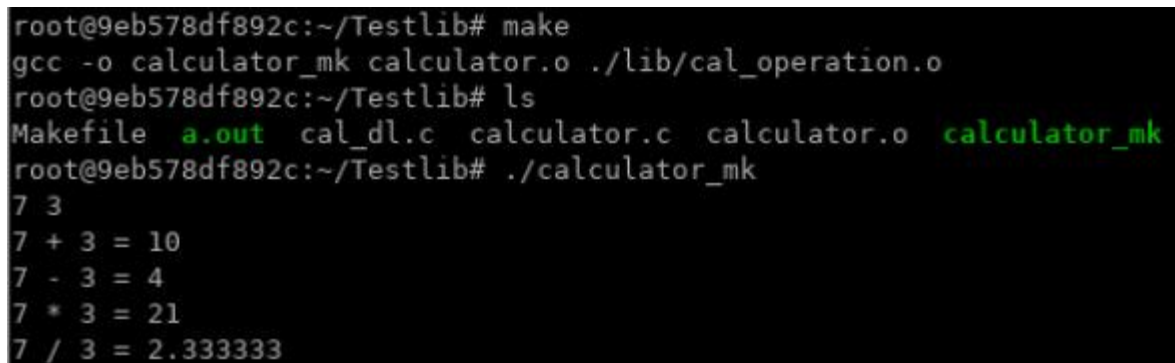
Makefile 파일은 밑의 형식에 따른다.

대상(TARGET) : 의존하는 파일(PREREQUISITES)

명령(COMMAND)

실행파일인 calculator\_mk를 calculator.o와 lib폴더 안에있는 cal\_operation.o를 의존하는 관계임을 지정한 뒤 gcc명령어를 이용해 명령어를 작성한다.

2) make



```

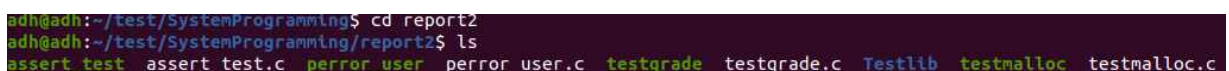
root@9eb578df892c:~/Testlib# make
gcc -o calculator_mk calculator.o ./lib/cal_operation.o
root@9eb578df892c:~/Testlib# ls
Makefile  a.out  cal_dl.c  calculator.c  calculator.o  calculator_mk
root@9eb578df892c:~/Testlib# ./calculator_mk
7 3
7 + 3 = 10
7 - 3 = 4
7 * 3 = 21
7 / 3 = 2.333333

```

작성한 Makefile을 make 명령어를 통해 실행하면 정의한 명령어가 실행되며 calculator\_mk가 생기게 된다. 이후 실행시키면 사칙연산 프로그램이 실행됨을 확인할 수 있다.

### 3. GIT 사용법을 정리하고, Github 사이트에 자신의 계정을 만들고 위아래의 모든 과제를 report2 프로젝트에 올리시오.

우분투에서 git을 사용하기 위해 sudo apt-get install git 명령을 사용하여 패키지 리스트를 업데이트한 뒤, sudo apt install git 명령어를 사용하여 git을 설치한다.



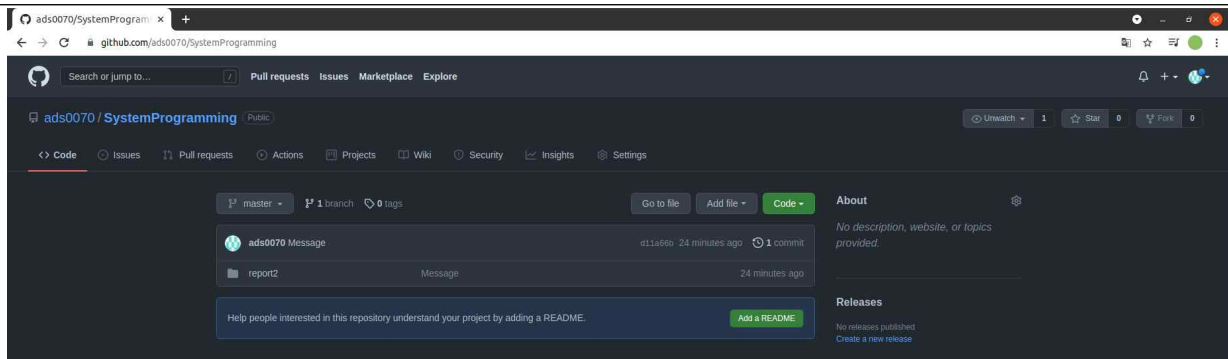
```

adh@adh:~/test/SystemProgramming$ cd report2
adh@adh:~/test/SystemProgramming/report2$ ls
assert_test  assert_test.c  perror_user  perror_user.c  testgrade  testgrade.c  Testlib  testmalloc  testmalloc.c

```

cd 명령어를 사용하여 git 저장소에 올릴 프로젝트 파일 위치로 이동한다. mv 명령어를 사용해 해당 파일에 모든 파일을 이동시켰다.

push 했을 때 올라갈 사용자 정보를 입력하기 위해 git config --global user.name [이름], git config --global user.mail [메일 주소] 명령을 사용한다.



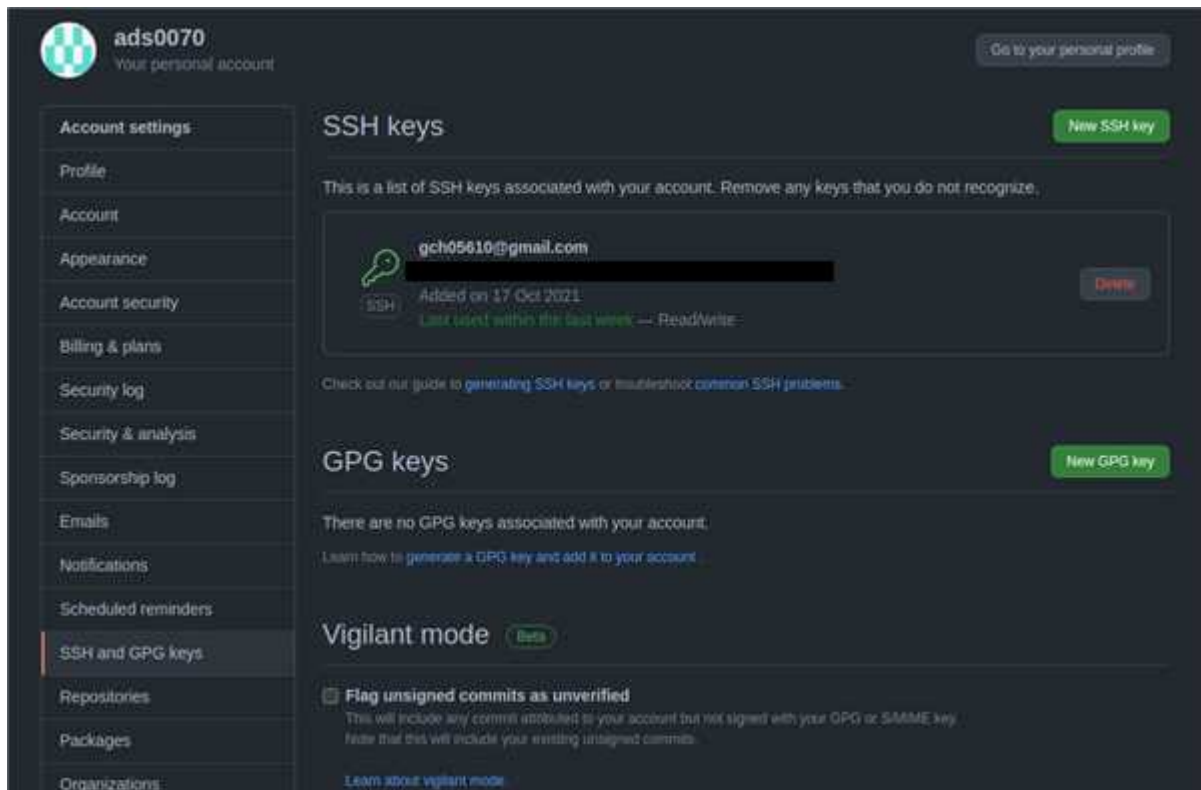
github 사이트에 접속하여 프로젝트를 올릴 SystemProgramming 저장소를 생성하고, git clone 주소를 복사하여 터미널에 git remote add origin [url 주소]를 입력하여 로컬 깃 저장소와 서버 깃 저장소를 연결한다.

이때, 원격 저장소에 접속할 때, 보안 요소를 추가하기 위해 SSH key를 발급받는다.

```
adh@adh:~/test/SystemProgramming$ ssh-keygen -t rsa -b 4096 -C "gch05610@gmail.com"
```

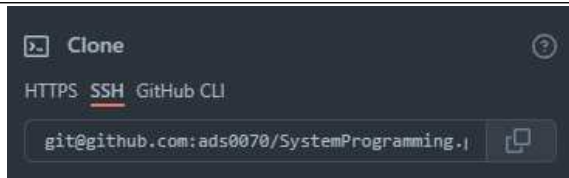
```
adh@adh:~/test/SystemProgramming$ xclip -sel clip < ~/.ssh/id_rsa.pub
```

ssh-keygen 명령어를 사용하여 SSH Key를 생성하고, xclip 명령을 사용하여 SSH Key를 복사한다.



복사한 SSH Key를 github 설정에서 SSH Key 추가에 붙여 넣는다.





```
adh@adh:~/test/SystemProgramming$ git config remote.origin.url git@github.com:ads0070/SystemProgramming.git
```

SSH로 깃을 연결하기 위해 SSH Clone 주소를 복사하여 로컬 깃 저장소와 서버 깃 저장소를 새로 연결한다.

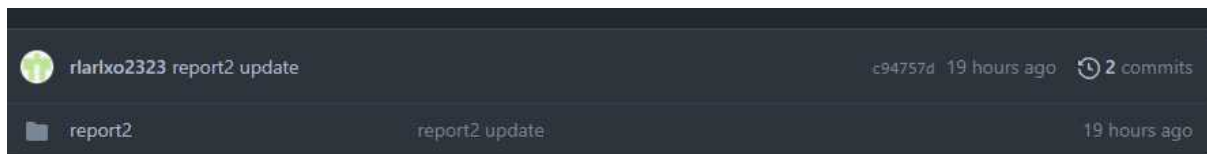
연결된 깃 저장소에 파일을 올리기 위해 아래의 명령어를 입력한다.

git add . : 저장소에 올릴 파일을 정하는 것 (.은 전체 파일을 의미)

git commit -m "message" : 저장소에 올리는데 설명을 붙여주는 것

git push -u origin master : 실제로 파일 업로드

```
adh@adh:~/test/SystemProgramming$ git push -u origin master
```



깃에 있는 파일을 내려받을 때는 git pull 명령어를 사용한다.

```
adh@adh:~/test/SystemProgramming$ git pull
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 18 (delta 4), reused 18 (delta 4), pack-reused 0
오브젝트 묶음 푸는 중: 100% (18/18), 10.88 KiB | 2.18 MiB/s, 완료.
github.com:ads0070/SystemProgramming URL에서
d11a66b..c94757d master -> origin/master
업데이트 중 d11a66b..c94757d
Fast-forward
 report2/Testlib/Makefile      | 8 ++++++++
 report2/Testlib/a.out         | Bin 0 -> 17104 bytes
 report2/Testlib/cal_dl.c      | 52 +++++++++++++++++++++++++++++++++++++
 report2/Testlib/calculator.c | 13 ++++++++
 report2/Testlib/calculator.o  | Bin 0 -> 2536 bytes
 report2/Testlib/calculator_mk | Bin 0 -> 17400 bytes
 report2/Testlib/gmon.out      | Bin 0 -> 493 bytes
 report2/Testlib/grofTest.txt  | 40 ++++++++
 report2/Testlib/lib/cal_operation.c | 17 ++++++++
 report2/Testlib/lib/cal_operation.h | 7 ++++++
 report2/Testlib/lib/cal_operation.o | Bin 0 -> 1720 bytes
 report2/Testlib/lib/libcalop.a | Bin 0 -> 1898 bytes
 report2/Testlib/lib/libcalop.so | Bin 0 -> 15744 bytes
```

#### 4. GDB와 DDD의 사용법을 정리하고, 위 2번 실습 예제의 GDB 및 DDD 테스트 결과를 나타내시오.

1) gdb install

gdb를 실행하기 위해 apt-get install gdb를 통해 다운로드한 뒤에, Makefile 파일에 -g 옵션을 추가하여 컴파일 후 실행한다.

## 2) gdb 실행 및 list 명령어

```
Reading symbols from calculator_mk...
(gdb) list
1      #include <stdio.h>
2      #include "lib/cal_operation.h"
3
4      int main(){
5          int a,b;
6          scanf("%d %d",&a,&b);
7
8          printf("%d + %d = %d\n", a,b,add(a,b));
9          printf("%d - %d = %d\n", a,b,subtract(a,b));
10         printf("%d * %d = %d\n", a,b,multiply(a,b));
(gdb) █
```

gdb <파일이름> 명령어를 이용하여 gdb를 실행한다. 'Reading symbols from calculator\_mk'가 출력되어 선택한 파일의 디버깅 데이터를 확인할 수 있다.

list 명령어를 통해 소스 코드를 확인할 수 있다.

## 3) gdb 명령어

```
(gdb) break 9
Breakpoint 1 at 0x11ba: file calculator.c, line 9.
(gdb) run
Starting program: /root/Testlib/calculator_mk
warning: Error disabling address space randomization: Operation not permitted
7 3
7 + 3 = 10

Breakpoint 1, main () at calculator.c:9
9      printf("%d - %d = %d\n", a,b,subtract(a,b));
(gdb) next
7 - 3 = 4
10     printf("%d * %d = %d\n", a,b,multiply(a,b));
(gdb) next
7 * 3 = 21
11     printf("%d / %d = %f\n", a,b,divide(a,b));
(gdb) next
7 / 3 = 2.333333
13     }
```

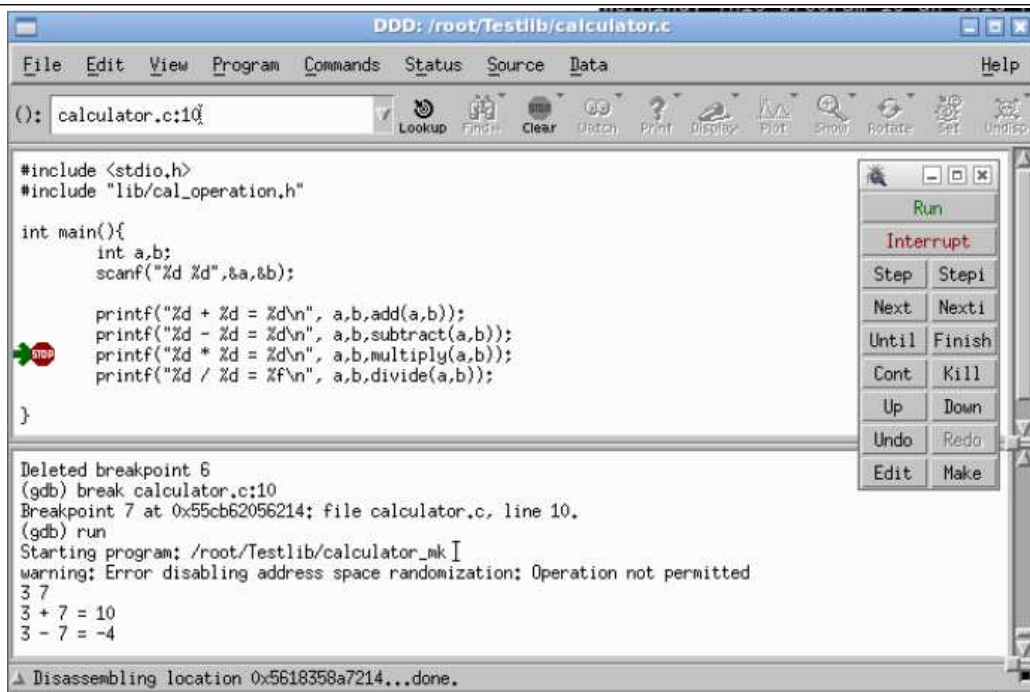
break : 정지점을 설정하여 해당 소스가 정지점까지 컴파일되게 한다.

run : 설정한 정지점까지 컴파일 할 수 있다.

next : 정지점 이후로 한줄씩 컴파일

## 4) DDD





GUI를 지원하는 디버깅 명령어로 GUI를 통해 정지점을 설정할 수 있고 컴파일 할 수 있다.  
하단의 출력창을 통해 gdb와 같이 출력됨을 알 수 있다.

## 5. perror() 함수 구현 : 라이브러리 함수인 perror() 함수와 동일하게 동작하는 my\_perror() 함수를 구현하고, 이를 이용하여 5장 예제 14에서 perror() 함수를 대체하여 프로그램을 실행해 보자.

**perror() 함수 :** 가장 최근의 오류 원인을 문자열로 출력해 주는 라이브러리 함수

인자로 문자열 `s`가 주어지면 "`s` : 최근 오류 발생 원인을 설명하는 문자열"이 출력되고, `s`가 NULL이면 "최근 오류 발생 원인을 설명하는 문자열"만 출력된다.

`perror()` 함수와 같은 기능을 하는 `my_perror()` 함수를 만들기 위해서는 `strerror()` 함수와 오류 번호 `errno`를 사용한다.

**strerror() 함수 :** 오류번호 `errno`와 관련된 오류 설명 문자열을 출력한다.

```
열기(O)  [icon]  perror_user.c  저장(S)  [icon]  [icon]  [icon]
1 /*
2 * my_perror() 함수 구현
3 */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include <errno.h>
9
10 void my_perror(char *front_str)
11 {
12     if (front_str == NULL) {
13         printf("%s\n", strerror(errno));
14     }
15     else {
16         printf("%s: %s\n", front_str, strerror(errno));
17     }
18     return;
19 }
20
21 void main(int argc, char *argv[])
22 {
23     FILE *f;
24
25     // 명령행 인자가 주어지지 않은 경우 예외처리
26     if(argc < 2) {
27         printf("Usage: perror_use file_name\n");
28         exit(1);
29     }
30
31     if((f=fopen(argv[1], "r")) == NULL) {
32         // perror 라이브러리 함수 사용
33         perror(NULL);
34         perror("fopen");
35
36         // my_perror 사용자 정의 함수 사용
37         my_perror(NULL);
38         my_perror("fopen");
39
40         exit(1);
41     }
42
43     printf("Open a file %s\n", argv[1]);
44
45     fclose(f);
46     return;
47 }
```

perror\_user.c 파일을 생성하여 코드를 작성한다.

my\_perror() 함수는 매개 변수로 front\_str 문자열을 사용한다. 해당 문자열이 NULL이면 오류 번호 errno을 사용하여 strerror() 라이브러리 함수를 사용해 오류 발생 원인 설명을 출력한다.

front\_str이 NULL이 아니면 "front\_str : 오류 발생 원인 설명"의 형태로 출력한다.

```
adh@adh:~/test/report2$ gcc -o perror_user perror_user.c
adh@adh:~/test/report2$ ./perror_user
Usage: perror_use file_name
adh@adh:~/test/report2$ ./perror_user nofilename
No such file or directory
fopen: No such file or directory
No such file or directory
fopen: No such file or directory
adh@adh:~/test/report2$ ./perror_user new.txt
Open a file new.txt.
```

perror\_user.c 파일 컴파일을 위해 gcc -o로 실행 파일 perror\_user를 생성한다.

① ./perror\_user 실행 : 명령행 인자로 파일 이름이 입력되지 않았기 때문에 26번 줄의 조건문이 충족되어 "Usage: perror\_use file\_name"이 출력된다.

② ./perror\_user nofilename 실행 : 명령행 인자로 입력된 nofilename이라는 파일이 존재하지 않기 때문에 파일을 오픈할 수 없으므로, perror() 함수와 my\_perror()를 통해 에러 문구가 출력된다. perror 함수가 호출될 때 인자로 문자열이 입력되지 않으면 “No such file or directory”의 오류 설명 문자열만 출력되고, 인자로 "fopen"이 입력되면 "fopen : No such file or directory"의 문자열이 출력된다.

③ ./perror\_user new.txt 실행 : 명령행 인자로 입력된 new.txt 파일이 존재하기 때문에 정상적으로 파일이 열렸다는 "Open a file new.txt" 문자열이 출력된다.

## **6. assert() 함수 구현 : 라이브러리 함수인 assert() 함수와 동일하게 동작하는 my\_assert() 함수를 구현하고, 이를 이용하여 5장 예제 15에서 assert() 함수를 대체하여 프로그램을 실행해 보자.**

**assert() 함수 :** 함수 인자가 거짓이면 오류가 난 파일 이름과 줄 번호, 오류 메시지를 출력하고 abort() 함수를 호출해 프로그램을 종료한다.

조건부 오류 처리를 직접 구현하기 위해서 매크로를 사용하여 파일 이름, 줄 번호, 함수 이름을 호출한다.

```
assert_test.c
~/test/SystemProgramming/report2

1 /*
2  * my_assert() 함수 구현
3  */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include <assert.h>
9
10 char *filename = NULL;
11
12 void my_assert(int expr)
13 {
14     if(expr==0) {
15         fprintf(stderr, "%s: %s:%d: %s: Assertion '((num >= 0) && (num <= 100))' failed. "
16                 ,filename,__FILE__,__LINE__,__func__);
17         abort();
18     }
19 }
20
21 void foo(int num)
22 {
23     // assert 라이브러리 함수 사용
24     //assert( ((num >= 0) && (num <= 100)) );
25
26     // my_assert 사용자 정의 함수 사용
27     my_assert( ((num >= 0) && (num <= 100)) );
28
29     printf("foo: num = %d\n", num);
30 }
31
32 void main(int argc, char *argv[])
33 {
34     int num;
35
36     filename = strtok(argv[0], "./");
37
38     // 명령행 인자가 주어지지 않은 경우 예외 처리
39     if (argc < 2) {
40         fprintf(stderr, "Usage: assert_test aNumber\n(0 <= aNumber <= 100)\n");
41         // stderr : 표준에러출력장치 -> 버퍼링 없이 바로 출력되므로 문제가 생겼을 경우 즉시 출력 가능
42         exit(1);
43     }
44
45     num = atoi(argv[1]); // atoi() : 문자열을 정수로 변환
46     foo(num);
47 }
```

my\_assert 함수는 매개 변수로 사용자로부터 입력받은 숫자인 num과 실행 파일의 이름 filename을 사용한다. 실행 파일 이름이 ./assert\_test로 되어 있으므로 앞 두 글자를 자른 문자열을 전역 변수 filename에 저장한다. 만일 조건식이 거짓인 경우, 표준 오류(stderr)로, “오류가 생긴 실행 파일 이름: c파일 이름:줄 번호: 함수 이름:: Assertion ‘((num >= 0) && (num <= 100))’ failed.”가 출력되고, abort() 함수로 프로그램을 종료시킨다.

strtok() 함수 : 구분 문자를 기준으로 문자열을 자르는 함수

\_\_FILE\_\_ 매크로 : 해당 매크로가 호출된 파일명

\_\_FUNCTION\_\_ 매크로 : 해당 매크로가 호출된 함수명

\_\_LINE\_\_ 매크로 : 해당 매크로가 호출된 라인 번호

abort() 함수 : 프로그램 강제 종료 함수

argv[] : 0번 인덱스에는 ./assert\_test가 들어가고, 1번 인덱스에는 입력한 숫자가 들어간다.

atoi() 함수 : 문자열을 정수로 변환한다.

```
adh@adh:~/test/SystemProgramming/report2$ gcc -o assert_test assert_test.c
adh@adh:~/test/SystemProgramming/report2$ ./assert_test
Usage: assert_test aNumber
(0 <= aNumber <= 100)
adh@adh:~/test/SystemProgramming/report2$ ./assert_test 10
foo: num = 10
adh@adh:~/test/SystemProgramming/report2$ ./assert_test 1010
assert_test: assert_test.c:16: my_assert: Assertion '((num >= 0) && (num <= 100))' failed. 중지됨 (코어 덤프됨)
```

gcc 컴파일러로, assert\_test 실행 파일을 생성한다.

① ./assert\_test 실행 : 명령행 인자로 숫자가 입력되지 않았으므로, “Usage: assert\_test aNumber”가 출력된다.

② ./assert\_test 10 실행 : 입력된 숫자 10은 0 이상 100 이하의 조건에 부합하므로, “foo: num = 10”이 출력된다.

③ ./assert\_test 1010 실행 : 입력된 숫자 1010이 0 이상 100 이하의 조건에 충족되지 않으므로, “assert\_test: assert\_test.c:15: my\_assert: Assertion ‘((num >= 0) && (num <= 100))’ failed.” 문자열이 출력되고, abort() 함수로 인해 프로그램이 종료되었음을 확인할 수 있다.

## 7. 위 1-2번 실습문제의 프로그램에 대하여 gprof 프로파일링과 Valgrind 메모리 누수 디버깅을 적용하여 보시오.

### 1) Makefile 재설정

```
root@9eb578df892c:~/Testlib# touch calculator.o ./lib/cal_operation.o
root@9eb578df892c:~/Testlib# make
gcc -o calculator_mk calculator.o ./lib/cal_operation.o -pg
root@9eb578df892c:~/Testlib# ls
Makefile  a.out  cal_dl.c  calculator.c  calculator.o  calculator_mk  lib
root@9eb578df892c:~/Testlib# ./calculator_mk
7 3
7 + 3 = 10
7 - 3 = 4
7 * 3 = 21
7 / 3 = 2.333333
root@9eb578df892c:~/Testlib# ls
Makefile  cal_dl.c  calculator.o  gmon.out
a.out  calculator.c  calculator_mk  lib
```

Makefile의 명령어 뒤에 gprof를 실행하기 위해 옵션 -pg를 붙인 뒤 다시 make 한다.

이후 calculator\_mk를 컴파일하면 gmon.out 파일이 생성됨을 알 수 있다.

**touch** : 해당 파일의 사용시간 인자를 변경한다.

### 2) gprof



```

root@9eb578df892c:~/Testlib# gprof calculator_mk gmon.out > grofTest.txt
root@9eb578df892c:~/Testlib# ls
Makefile  a.out  cal_dl.c  calculator.o  calculator_mk  gmon.out  grofTest.txt  USB
root@9eb578df892c:~/Testlib# cat grofTest.txt
Flat profile:

Each sample counts as 0.01 seconds.
no time accumulated

%   cumulative  self      self     total
time  seconds  seconds   calls   Ts/call  Ts/call  name
%
the percentage of the total running time of the
program used by this function.

cumulative a running sum of the number of seconds accounted
seconds    for by this function and those listed above it.

self       the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

```

위에서 생성된 gmon.out은 사용자가 이해하기 힘들게 되어있기 때문에 gprof 명령어를 통해 txt파일로 재설정을 함으로써 확인할 수 있다. 결과는 위 사진과 같다.

### 3) Valgrind

```

root@9eb578df892c:~/Testlib# valgrind --leak-check=yes -v ./calculator_mk
==12190== Memcheck, a memory error detector
==12190== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==12190== Using Valgrind-3.15.0-608cb11914-20190413 and LibVEX; rerun with -h
==12190== Command: ./calculator_mk
==12190==
--12190-- Valgrind options:
--12190--    --leak-check=yes
--12190--    -v
--12190-- Contents of /proc/version:
--12190--   Linux version 5.10.16.3-microsoft-standard-WSL2 (oe-user@oe-host)
utils) 2.34.0.20200220) #1 SMP Fri Apr 2 22:23:49 UTC 2021
--12190--
--12190-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-s
--12190-- Page sizes: currently 4096, max supported 4096
--12190-- Valgrind library directory: /usr/lib/x86_64-linux-gnu/valgrind
--12190-- Reading syms from /root/Testlib/calculator_mk
--12190-- Reading syms from /usr/lib/x86_64-linux-gnu/ld-2.31.so
--12190--   Considering /usr/lib/x86_64-linux-gnu/ld-2.31.so ..
--12190--   .. CRC mismatch (computed 975d0390 wanted 30bd717f)
--12190--   Considering /lib/x86_64-linux-gnu/ld-2.31.so ..
--12190--   .. CRC mismatch (computed 975d0390 wanted 30bd717f)
--12190--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.31.so ..
--12190--   .. CRC is valid

```

valgrind --leak-check=yes -v ./calculator\_mk 명령어를 통해 해당 파일의 메모리 누수를 확인 할 수 있다.

**8. 두 행렬의 크기를 입력받아서 필요한 만큼의 메모리를 동적으로 할당하여 행렬의 내용을 저장하고 두 행렬을 더하는 프로그램을 작성하시오.**



**malloc() 함수** : num 바이트 수 만큼 메모리 공간을 할당한다. 할당 성공 시 할당된 메모리의 포인터를 반환하고, 실패 시 NULL 값을 반환한다.

malloc() 함수를 사용하여 사용자가 입력한 크기 만큼의 2차원 배열 두 개의 메모리를 할당받는다.

**free() 함수** : 메모리 공간을 해제하고 반환한다.

```
testmalloc.c
~/test/report2

1 /*
2  * 동적 메모리 할당
3  */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7
8 void main()
9 {
10     int col, row, num = 100;
11     int i = 0, j = 0, k = 0;
12
13     printf("행의 크기(세로), 열의 크기(가로)를 입력하세요: ");
14     scanf("%d %d", &row, &col);
15
16     // 배열의 세로
17     int **arr1 = malloc(sizeof(int *) * row); // 이중 포인터에 (int 포인터 크기 * row)만큼 동적 메모리 할당
18     int **arr2 = malloc(sizeof(int *) * row);
19
20     // 배열의 가로
21     for (i = 0; i < row; i++) {
22         arr1[i] = malloc(sizeof(int) * col); // (int의 크기 * col)만큼 동적 메모리 할당
23         arr2[i] = malloc(sizeof(int) * col);
24     }
25
26     // 두 개의 행렬 각 요소에 값 할당
27     for (i = 0; i < row; i++) {
28         for (j = 0; j < col; j++) {
29             arr1[i][j] = i + j;
30             arr2[i][j] = num;
31             num--;
32         }
33     }
34
35     // 행렬A와 행렬B 출력
36     printf("\n행렬 A와 행렬 B\n");
37     for (i = 0; i < row; i++) {
38         for (j = 0; j < col; j++) {
39             printf("%d ", arr1[i][j]);
40         }
41         printf(" ");
42         for (k = 0; k < col; k++) {
43             printf("%d ", arr2[i][k]);
44         }
45         printf("\n");
46     }
47
48     printf("-----\n");
49     printf("두 행렬의 합\n");
50     printf("-----\n");
51
52     // 행렬 A와 행렬 B를 더한 결과 출력
53     for (i = 0; i < row; i++) {
54         for (j = 0; j < col; j++) {
55             printf("%d ", arr1[i][j] + arr2[i][j]);
56         }
57         printf("\n");
58     }
59
60     free(arr1[0]); // arr1 배열의 가로 공간 메모리 해제
61     free(arr1); // arr1 배열의 세로 공간 메모리 해제
62     free(arr2[0]); // arr2 배열의 가로 공간 메모리 해제
63     free(arr2); // arr2 배열의 세로 공간 메모리 해제
64 }
```

arr1과 arr2에 (사용자가 입력한 row \* int 포인터의 크기)만큼 동적 메모리를 할당한다. 그 다음, 반복문을 사용하여 arr1[i]와 arr2[i]에 (사용자가 입력한 col \* int의 크기)만큼 동적 메모리를 할당한다.

arr2와 arr2 행렬이 생성되었으면, 각 요소에 임의의 값을 할당해 준 뒤, 행렬 A와 행렬 B를 출력한다.

마지막으로는 행렬 A와 행렬 B의 같은 인덱스에 위치한 값을 더해서 행렬의 합을 출력한다.

```
adh@adh:~/test/report2$ gcc -o testmalloc testmalloc.c
adh@adh:~/test/report2$ ./testmalloc
행의 크기(세로), 열의 크기(가로)를 입력하세요: 4 3

행렬 A와 B
0 1 2      100 99 98
1 2 3      97 96 95
2 3 4      94 93 92
3 4 5      91 90 89
-----
두 행렬의 합
-----
100 100 100
98 98 98
96 96 96
94 94 94
```

testmalloc을 실행하여 행과 열로 4, 3을 입력하면 3 x 4 사이즈의 행렬 A와 B가 출력되고, 두 행렬의 합이 아래에 출력된다.

## 9. 학생 수를 입력받은 다음, 학생의 이름, 중간, 기말 점수를 기록하는 구조체를 학생 수만큼 동적으로 할당받아서, 이를 트리 구조체로 활용하여 트리 탐색을 할 수 있는 프로그램을 작성하시오.

학생에 대한 정보를 하나로 묶어 사용할 수 있도록 student 구조체를 정의한다.

compare 함수는 아래의 tsearch() 함수에 사용되는 비교 함수로, strcmp() 라이브러리 함수를 사용하여 tree[i]가 나타내는 객체인 첫째 인자와 배열 요소인 둘째 인자를 비교하여 같으면 0, 아니면 0이 아닌 값을 반환한다.

print\_node 함수는 이진 트리를 방문하는 twalk가 노드를 처음 만날 때 출력하는 함수이다.

```
1 /*
2  * 시험 점수 구조체 동적 할당, 트리 검색
3  */
4
5 #include <stdio.h>
6 #include <malloc.h>
7 #include <stdlib.h>
8 #include <string.h>
9 #include <search.h>
10
11 struct student {
12     char student_name[10];
13     int mid;
14     int final;
15 };
16
17 struct student *root = NULL; // 루트 노드
18
19 // 학생 이름 중복 비교 함수
20 int compare(const void *cp1, const void *cp2) {
21     return strcmp(((struct student *)cp1)->student_name, ((struct student *)cp2)->student_name);
22 }
23
24 // twalk가 노드를 처음 만날때 출력
25 void print_node(const void *nodeptr, VISIT order, int level)
26 {
27     if (order == preorder || order == leaf) {
28         printf("이름 = %s, 중간점수 = %d 기말점수 = %d \n", (*(struct student **)nodeptr)->student_name,
29             (*(struct student **)nodeptr)->mid,
30             (*(struct student **)nodeptr)->final);
31     }
32 }
33
34 void main() {
35     int i, student_number;
36     struct student **ret;
37
38     printf("학생 수를 입력해주세요 : ");
39     scanf("%d", &student_number);
40
41     if (student_number <= 0 || student_number == '\0') {
42         abort();
43     }
44
45     struct student *tree[student_number];
46
47     for (i = 0; i < sizeof(tree) / sizeof(struct student *); i++) {
48
49         tree[i] = malloc(sizeof(struct student));
50         printf("트리에 추가할 학생 이름, 중간 점수, 기말 점수를 입력해주세요 : ");
51         scanf("%s %d %d", tree[i]->student_name, &tree[i]->mid, &tree[i]->final);
52
53         // 트리에 학생 데이터 넣기
54         ret = (struct student **) tsearch((void *) tree[i], (void **) &root, compare); // tsearch() : 이진 트리로부터 특정 항목 탐색
55         printf("\n%s\n", (*ret)->student_name);
56
57         // tree[i]가 tsearch()로 탐색한 항목과 같으면 이미 존재, 아니면 추가되었다고 출력.
58         if (*ret == tree[i])
59             printf("트리에 추가되었습니다.\n\n");
60         else
61             printf("트리에 이미 존재합니다.\n\n");
62     }
63
64     twalk((void *) root, print_node); // twalk() : 이진 트리 방문
65
66     for (int i = 0; i < sizeof(tree) / sizeof(struct student *); i++) {
67         free(tree[i]);
68     }
69 }
```

tsearch() 함수는 둘째 인자 root가 가리키는 이진 트리에 대하여 tree[i]가 가리키는 내용에 대하여 탐색을 수행한다. 이때, 탐색하는 항목이 트리에 존재하면 그 항목에 대한 주소를 반환하고, 일치하는 항목이 없는 경우 key 객체를 트리에 추가하고 그 항목의 주소를 반환한다.

```
adh@adh:~/test/report2$ gcc -o testgrade testgrade.c
adh@adh:~/test/report2$ ./testgrade
학생 수를 입력해주세요 : 3
트리에 추가할 학생 이름, 중간 점수, 기말 점수를 입력해주세요 : 김기태 95 98
"김기태" 님이 트리에 추가되었습니다.

트리에 추가할 학생 이름, 중간 점수, 기말 점수를 입력해주세요 : 안대현 94 90
"안대현" 님이 트리에 추가되었습니다.

트리에 추가할 학생 이름, 중간 점수, 기말 점수를 입력해주세요 : 김승후 80 80
"김승후" 님이 트리에 추가되었습니다.

이름 = 김승후, 중간점수 = 80 기말점수 = 80
이름 = 김기태, 중간점수 = 95 기말점수 = 98
이름 = 안대현, 중간점수 = 94 기말점수 = 90
```

gcc 컴파일러를 사용하여 파일을 실행하면 사용자로부터 학생 수를 입력받아 그 크기만큼 메모리를 동적할당 받아 성적 정보를 저장한다. 그 후, 트리 탐색을 통해 학생 정보를 출력한다.