

쉘 스크립트 실습

시스템 프로그래밍

이태용 | 20143254 | 컴퓨터 소프트웨어공학과| 2019.09.30

개 요

20143254 이 태용

본 보고서는 김성우 교수님의 시스템 프로그래밍 수업의 과제물 제출을 목적으로 작성되었습니다.

본 보고서는 MS WORD로 작성되었고, 실습 환경은 아래와 같습니다.

* WSL(Window Subsystem for Linux)
* Docker : Ubuntu 16.04
* Ubuntu 16.04

대부분의 결과물은 **emacs**로 작성하였습니다..

결과에 대한 이해를 돕기 위해 소스코드와 결과 화면을 첨부하였습니다.

본 보고서에서는 리눅스 프로그래밍 환경, 디버그 및 오류 제어, 기본 함수에 관련된 개념과 직접 실습해 본 내용이 담겨있습니다.

내용

[준비 사항 5](#_Toc22584374)

[1. 교재에서 C 프로그래밍 환경에 대한 내용을 읽고 이해한다. 5](#_Toc22584375)

[2. GNU autotools (autoconf, automake, libtool) 과 CMake 에 대하여 조사하고 사용 방법을 보고서에 정리하여 본다. 6](#_Toc22584376)

[3. RCS (Revision Control System)에 대하여 조사하고 사용 방법을 보고서에 정리하여 본다. 9](#_Toc22584377)

[4. 교재 등을 읽고 디버깅 및 오류 처리에 대해 내용을 이해한다. 11](#_Toc22584378)

[5. Valgrind 에 대해 조사하고 요약하여 보고서에 정리하여 본다. 16](#_Toc22584379)

[16](#_Toc22584380)

[6. 교재에서 POSIX 표준 및 기본 라이브러리 함수에 대한 내용을 읽고 이해한다. 19](#_Toc22584381)

[POSIX란? 19](#_Toc22584382)

[7. GNU C 라이브러리(glibc) 에 대하여 조사하고 요약하여 보고서에 정리하여 본다. 20](#_Toc22584383)

[GNU C 라이브러리란? 20](#_Toc22584384)

[기능 20](#_Toc22584385)

[호환성 계층 20](#_Toc22584386)

[실습 21](#_Toc22584387)

[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. 21](#_Toc22584388)

[2. 위 문제에 대하여, C 소스 파일들을 목적 파일로 만들고, 또한 이 파일들을 이용하여 라이브러리 또는 실행 파일로 만들어 주는 Makefile을 작성하여 실행시켜 보시오. 이 때, 라이브러리 관련 파일들은 서브 디렉토리에 두도록 한다. 27](#_Toc22584389)

[3. GIT의 사용법을 정리하고, Github 사이트에 자신의 계정을 만들고 아래의 모든 과제를 lab2 프로젝트에 올리시오. 36](#_Toc22584390)

[4. GDB 와 DDD의 사용법을 정리하고, 위 2번 실습 예제에 대하여 GDB 및 DDD 테스트 결과를 나타내시오. 37](#_Toc22584391)

[5. perror() 함수 구현 : 라이브러리 함수인 perror() 함수와 동일하게 동작하는my\_perror() 함수를 구현하고, 이를 이용하여 예제 13에서 perror() 함수를 대체하여 프로그램을 실행해 보자. 45](#_Toc22584392)

[6. assert() 함수 구현 : 라이브러리 함수인 assert() 함수와 동일하게 동작하는my\_assert() 함수를 구현하고, 이를 이용하여 5장 예제 14에서 assert() 함수를 대체하여 프로그램을 실행해 보자. 48](#_Toc22584393)

[7. 위 1-2번 실습문제의 프로그램에 대하여 gprof 프로파일링과 Valgrind 메모리 누수 디버깅을 적용하여 보시오. 49](#_Toc22584394)

[8. 두 행렬의 크기를 입력받아서 필요한 만큼의 메모리를 동적으로 할당하여 행렬의 내용을 저장하고 두 행렬을 더하는 프로그램을 작성하시오. 59](#_Toc22584395)

[9. 학생 수를 입력받은 다음, 학생의 이름, 중간, 기말 점수를 기록하는 구조체를 학생 수만큼 동적으로 할당 받아서, 이를 트리 구조체로 활용하여 트리 탐색을 할 수 있는 프로그램을 작성하시오. 63](#_Toc22584396)

# 준비 사항

1. 교재에서 C 프로그래밍 환경에 대한 내용을 읽고 이해한다.

리눅스 환경에서의 C언어 중요성

* 유닉스나 리눅스의 경우 많은 부분이 C언어로 이루어져 있음.
* 리눅스에서는 C 컴파일러가 기본적으로 존재하고 없을 경우 gcc 컴파일러를 설치할 수 있음.
* C언어는 고급언어로 이식성이 높고, 코드가 직접적임
* 또한 메모리에 직접 접근할 수 있음
* 따라서 하드웨어를 제어하는 경우 어셈블리어와 함께 사용하는 경우도 많음.
* 빠른 성능을 보여줌

GCC

* UNIX 또는 LINUX등의 운영체제 상에서 가장 많이 쓰이는 C컴파일러
* 컴파일/어셈블 단계 외에도 전처리 단계, 링크 단계 등을 처리하는 기능을 포함

|  |
| --- |
| 컴파일러  ccl  전처리기  cpp  링커  ld  cpp  어셈블러  as  cpp  소스  파일들 |

1. GNU autotools (autoconf, automake, libtool) 과 CMake 에 대하여 조사하고 사용 방법을 보고서에 정리하여 본다.

GNU autotools란?

* Autoconf, automake, libtools 도구들로 구성되어 있는 Build System
* 대부분의 GNU 프로그램들은 Autotools를 사용해 빌드 환경을 구성함
* 기존의 make를 이용하기 위한 Makefile을 쉽고 이식성있게 작성하기 위한 기법

autoconf

* Configure 파일을 생성하는 툴
* Configure.ac 파일을 직접 작성해야 함

Automake

* Makefile을 생성하는 툴
* Makefile.am 파일을 직접 작성해야 함

Libtool

* 라이브러리를 쉽게 제작(컴파일) 하기 위해 만든 툴
* Autoconfig와 같이 사용할 수도 있고, 독립적으로 사용할 수도 있음.

GNU Autotools 사용 순서

1. configure.ac 에 Makefile 이 필요로 하는 사항들을 기록한다.
2. Makefile.am 에 이들 가변수를 사용하여 Makefile 의 초안을 작성한다.
3. configure 실행시에 configure.ac 에 지정한 항목들이 check 되면서 Makefile.am 에서 필요한 정보들이 유효한 값들로 치환된다.
4. Makefile.am 이 유효한 값들을 가지면서 Makefile 이 된다.

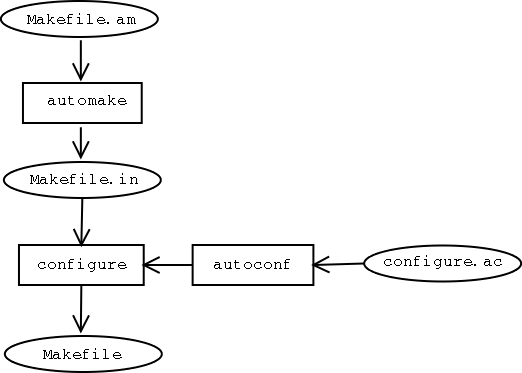


Figure 1. how GNU Autotools works

Configure.ac

* autoconf, automake 의 입력값
* autoconf는 이를 받아서 configure 파일을 만든다.
* automake는 이를 makefile.am 과 함께 받아 makefile.in 을 만든다.
* 보통 이미 지정된 macro 를 사용하여 원하는 정보를 얻거나 설정할 수 있다.
* 예
  + - AC\_CANONICAL\_SYSTEM : 현재 시스템에 대한 정보를 가져온다.
    - AC\_PROG\_CC : cc 가 사용가능한지를 check
    - AC\_CHECK\_HEADERS : 지정하는 header file 들이 시스템에 있는지 검사한다. HAVE\_NAME\_H 를 만들어준다.
    - AC\_TYPE\_ : 지정한 type definition 을 확인한다.
    - AC\_SUBST : configure.ac 에서 사용한 변수들이 configure 시 \*.in 파일에 지정한 변수를 찾아 유효값을 넣어준다.
    - AC\_CONFIG\_FILES : 생성될 파일들을 지정한다. 흔히 Makefile을 의미한다

Makefile.am

* automake 의 입력값
* 기본적인 Makefile의 틀을 지니고 있지만, configure에 따라 결정되는 변수들을 사용하고 있다.
* Makefile.am 은 해당 소스를 컴파일 하고자 하는 위치면 어디든 놓이게 된다.
* 가장 상위 directory 의 Makefile.am 에는 컴파일할 하위 directory 를 정하게 된다.
* configrure.ac 에서 정의되고 configure 를 통해 유효값들을 가지게 되는 변수를 가져오기 위해 @variable@을 사용한다. \_PROGRAMS, \_SCRIPTS, \_SOURCES 등 지정된 primary 들이 있고 앞에 인스톨될 위치나 프로그램명, 프로그램의 소스 등을 지정하게 되어있다.

1. RCS (Revision Control System)에 대하여 조사하고 사용 방법을 보고서에 정리하여 본다.

RCS란?

* 버전 관리(version control, revision control), 소스 관리(source control), 소스 코드 관리(source code management, SCM)란 동일한 정보에 대한 여러 버전을 관리하는 것을 의미
* [공학](https://ko.wikipedia.org/wiki/%EA%B3%B5%ED%95%99)과 [소프트웨어 개발](https://ko.wikipedia.org/wiki/%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4_%EA%B0%9C%EB%B0%9C)에서 팀 단위로 개발 중인 [소스 코드](https://ko.wikipedia.org/wiki/%EC%86%8C%EC%8A%A4_%EC%BD%94%EB%93%9C)나, [청사진](https://ko.wikipedia.org/wiki/%EC%B2%AD%EC%82%AC%EC%A7%84) 같은 설계도 등의 디지털 문서를 관리하는데 사용된다. 그러한 문서의 변경 사항들에 숫자나 문자로 이뤄진 [버전](https://ko.wikipedia.org/wiki/%EB%B2%84%EC%A0%84)을 부여해서 구분
* 비슷한 세스템으로는  CVS(Current Version System), SCCS(Source Code Control System)이 있음..

1.1.1. RCS

RCS는 소스 파일을 관리하는 여러 명령으로 구성되어 있다. 한 파일의 변경 사항을 하나의 파일로 관리하는 방식으로 소스 파일의 변경을 사항을 추적한다.

1.1.1.1. rcs 명령

rsc -i <filename>의 형식으로 명령을 내리면, 파일을 관리하기 위한 초기화  
작업이 수행된다. 관리 파일이 생성되는데 원 파일 명 뒤에 ,v가 추가된 형태이다. RCS라는 하위 디렉터리를 만들어두면 관리 파일은 자동으로 이 디렉터리에 저장된다.

1.1.1.2. ci 명령

현재 버전을 저장하는 ci 명령을 사용해 파일을 Check In 할 수 있다. ci  
<filename>의 형식으로 명령을 내리면 원래 파일은 지워지고 모든 내용과 제어 정보가 RCS파일인  
“filename,v” 파일에 들어가게 된다.

1.1.1.3. co 명령

파일을 변경하려면 파일을 Check Out 해야 한다. co -l <filename>을  
호출하면 디렉터리에 filename파일이 생기며 CVS 파일은 잠기게 된다. 따라서 다른 사용자가 동시에 파일을 수정하는 것을  
방지한다. 수정 후에는 ci를 이용해 다시 Check In 해야 한다. ci -l <filename>의 형식으로  
호출하면, 파일을 CVS에 넣고 자동으로 체크아웃 되면서 CVS가 잠긴다. 즉, co -l <filename>을  
호출하는 것과 같은 효과가 있다.

1.1.1.4. rolg 명령

rolg <filename> 명령으로 파일의 변경 내역을 볼 수 있다. 파일의 첫 번째 버전으로 돌아가고 싶다면 co -r.1.1 <filename>을 사용할 수 있다. 파일의 버전을 강제로 지정하려면 ci -r2 <filename>과 같은 방법을 사용할 수 있다.

1.1.1.5. rcsdiff 명령

rcsdiff -r1.1 -r1.2 <filename>과 같은 형식으로 두 버전의 차이점을 알아볼 수 있다.

1.1.1.6. 버전 식별

소스 파일에 특수한 매크로를 사용하여 Check In 하면 매크로가 확장되어 내용으로 대체된다. $RCSfile$ 매크로는 파일의 이름으로 확장되고, $Id$ 매크로는 버전을 식별하는 문자열로 확장된다.

1. 교재 등을 읽고 디버깅 및 오류 처리에 대해 내용을 이해한다.

* 소프트웨어 개발을 위해서는 분석, 설계 단계에서 만들어진 결과를 바탕으로 구현 단계에서 이를 소스 코드 형태로 잘 구현해주어야 함.
* 소스 코드 수준의 테스트를 위한 방법으로는 프로그램 소스코드 내에 매크로 등을 이용하여 GDB와 같은 디버거를 사용하는 방법이 있음

디버깅 단계

* 테스트
  + - 결함이나 버그가 존재하는지 확인
* 안정화
  + - 버그를 반복 가능한 상태로 만듦
* 구역화
  + - 코드의 위치 탐색
* 정정
  + - 코드 수정
* 검증
  + - 제대로 동작하는지 확인

GDB

* GNU 프로젝트의 디버거
* C, C++, Objective-C, Modular2, Ada 등의 언어 지원

GDB를 이용한 디버깅

* Gcc로 컴파일 할 때 -g 옵션을 사용
* GDB의 명령어들은 gdb –help 명령어 입력시 확인 가능



Figure 2. gdb --help

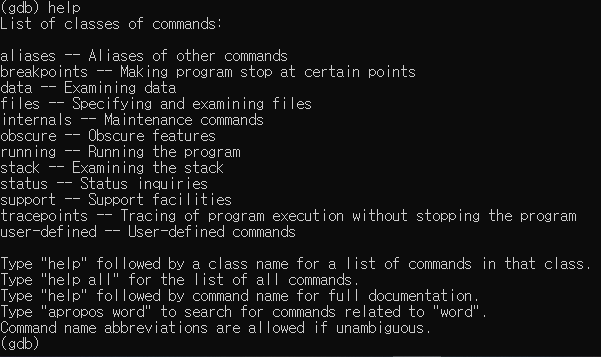


Figure 3. gdb 입력 후 help 명령어 입력

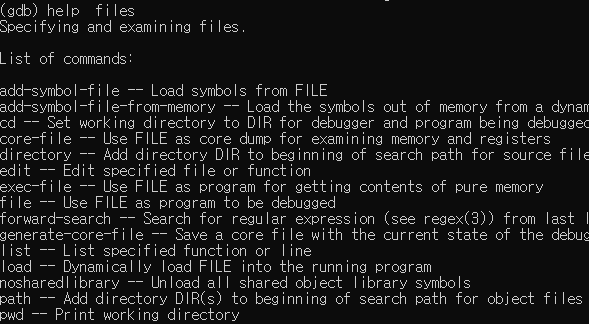
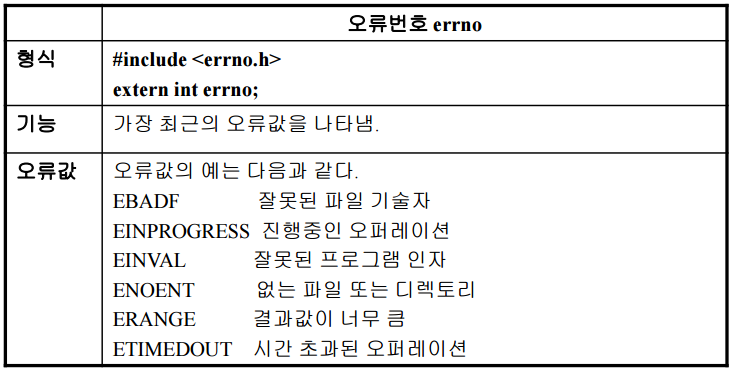


Figure 4. gdb – help files

**오류 처리**

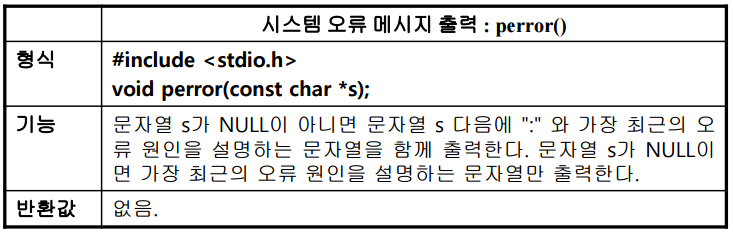
* **오류번호(errno) 사용**
  + - 많은 라이브러리 함수들이 오류 시 오류번호를 반환해당 오류 정보 제공



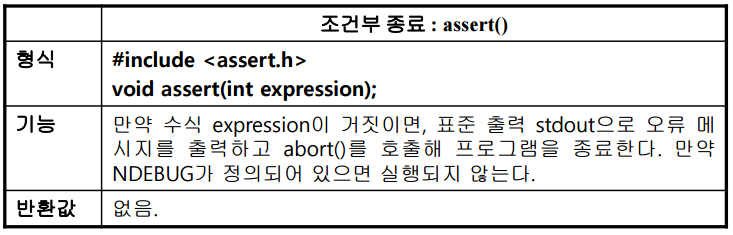
* **strerror 처리**
  + - 발생 오류 번호 errno 에 대해 strerror() 함수를 사용하여 오류 원인 출력 가능



* **perror 처리**
  + - perror() 함수는 가장 최근의 오류 원인을 문자열로 출력해 줌
    - perror() 함수 사용 시 오류 발생 함수를 사용하여 오류 발생 지점을 명시하는 것이 좋음



* **조건부 오류 처리**
  + - 특정 수식을 이용한 오류 조건을 검사하여 거짓이면, 오류 메시지를 출력하고 프로그램 종료



1. Valgrind 에 대해 조사하고 요약하여 보고서에 정리하여 본다.

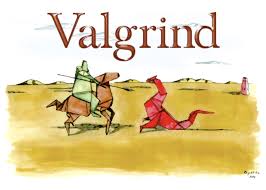


Figure 5. Valgrind Logo

**Valgrind란?**

* 리눅스 기반의 오픈소스 DBI 도구
* 클라이언트 프로그램의 실행 코드를 실행 시간에 직접 가공하는 기회를 제공
* 크게 Core와 Tool로 구성되어 있는데, 일반적으로 Valgreid는 Core를 의미함.

Instrumentation(수단)

* 코드 영역에서 코드에 필요한 내용을 추가하여 원하는 결과를 내도록 하는 작업을 통틀어 일컬음
* Valgrind는 대표적인 Dynamic instrumentation Framework임
* 다음의 어느 단계에서든 가능함
  + - 소스 코드
    - 전처리기
    - 어셈블리 코드
    - 기계 코드
    - 이진 실행 파일

**구조**

* Valgrind는 크게 코어와 도구들로 구성되어 있음.
* 코어는 instrumentation을 위한 환경을 제공해 주며, 도구들은 instrumented될 코드들을 포함함
* 모든 도구의 사용에 있어서 코어는 빌드하는 과정에서 tool의 function들에 wrapper를 사용하여 모든 tool에 core가 공통적인 동작을 수행토록 함
* 코어에 사용되는 모든 함수들은 valgrind의 실행 파일이 생성되는 과정에서 도구들의 함수 이름을 pointing함.
* 따라서 도구들에 링크되는 코어는 해당 도구가 필요로 하는 동작을 자동으로 수행

**확장성**

* 리눅스와 MacOS. 솔라리스를 지원
* AMD, X86, PPC, AIX, MIPS 등의 아키텍처를 지원

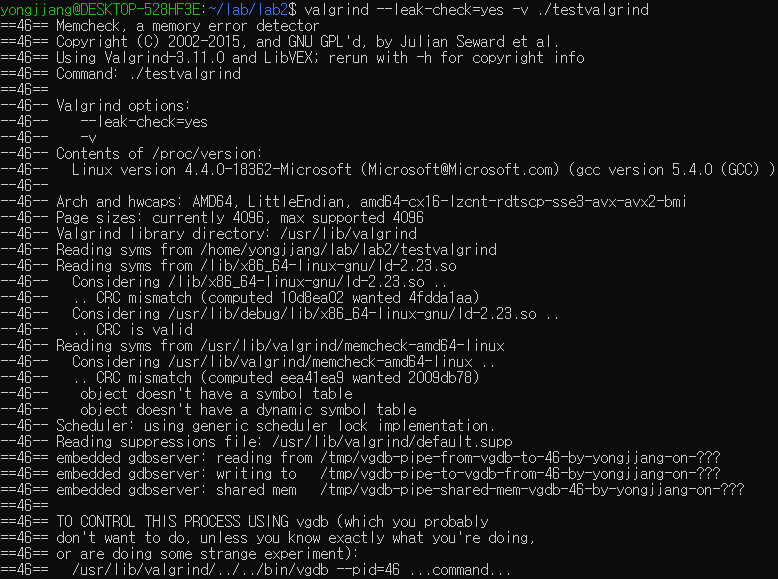
 

Figure 6. valgrind 실행 화면

1. 교재에서 POSIX 표준 및 기본 라이브러리 함수에 대한 내용을 읽고 이해한다.

POSIX란?

* 리눅스에서 사용하는 C 언어 라이브러리인 GNU C 라이브러리의 표준

일반적으로 사용되는 표준 C 라이브러리 함수

|  |  |  |
| --- | --- | --- |
| 헤더 파일 | 함수 기능 | 표준 함수 |
| stdio.h | 표준입출력 | printf(); scanf(); putchar(); getchar(); ... |
| 화일입출력 | fopen(); fclose(); fprintf(); ... |
| stdlib.h | 수치변환 | atoi(); itoa(); ... |
| 난수발생 | rand(); srand(); ... |
| 탐색 및 정렬 | bsearch(); qsort();... |
| ctype.h | 문자변환 | tolower(); toupper(); ... |
| 문자판별 | isalpha(); isdigit(); isupper(); ... |
| String.h | 문자열처리 | strcpy(); strlen(); strcmp(); ... |
| 동적메모리관리 | memcpy(); memset(); memchr(); ... |
| Math.h | 수학함수 | sin(); cos(); sqrt(); ... |
| Search.h | 탐색 | lsearch(); hsearch(); tsearch(); ... |
| Time.h | 시간관련함수 | time(); difftime(); ctime(); ... |

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

### GNU C 라이브러리란?

* GNU 프로젝트가 C 표준 라이브러리를 구현한 것
* 이름과는 다르게 현재는 C++도 지원
* 1990년대 초반 자유 소프트웨어 재단이 자신의 GNU 운영체제를 위해 시작되었음
* GNU 약소 일반 공중 사용 허가서 하에 배포되는 glibc는 자유 소프트웨어임.

### 기능

* 단일 유닉스 규격과 POSIX가 요구하는 기능들을 제공
* ISO C11, IKSO C99, 버클리 유닉스 (BSD) 인터페이스 등에서 요구하는 몇몇 기능들도 제공
* 추가적으로 glibc는 GNU 개발에 필요하다고 여겨지거나 유용한 확장도 제공

### 호환성 계층

* 안드로이드와 윈도우 같이 다른 시스템을 위해 쓰여진 프로그램드릉ㄹ 시스템이 제공하는 glibc 인터페이스에서 돌아가게 하는 호환성 계층이 존재
* Wine도 Win32 API/ABI에서 glibc로의 호환성 계층으로 여겨짐

# 실습

## 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.

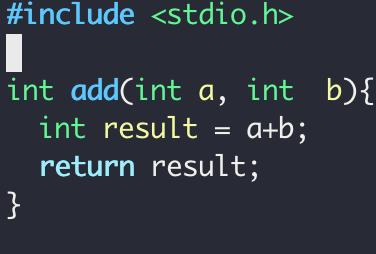


그림 1 덧셈 함수

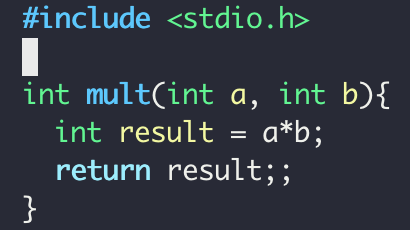


그림 2 곱셈 함수

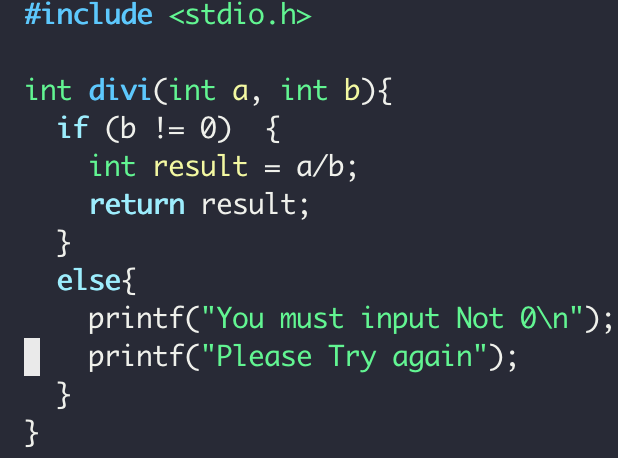


그림 3 나눗셈 함수

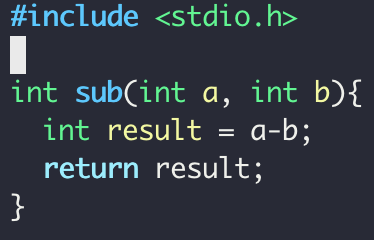


그림 4 뺄셈 함수

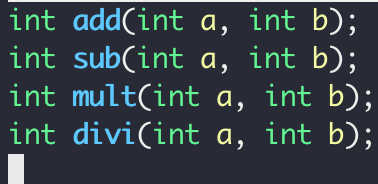


그림 5 헤더 파일 생성

그림 6 정적 라이브러리 제작

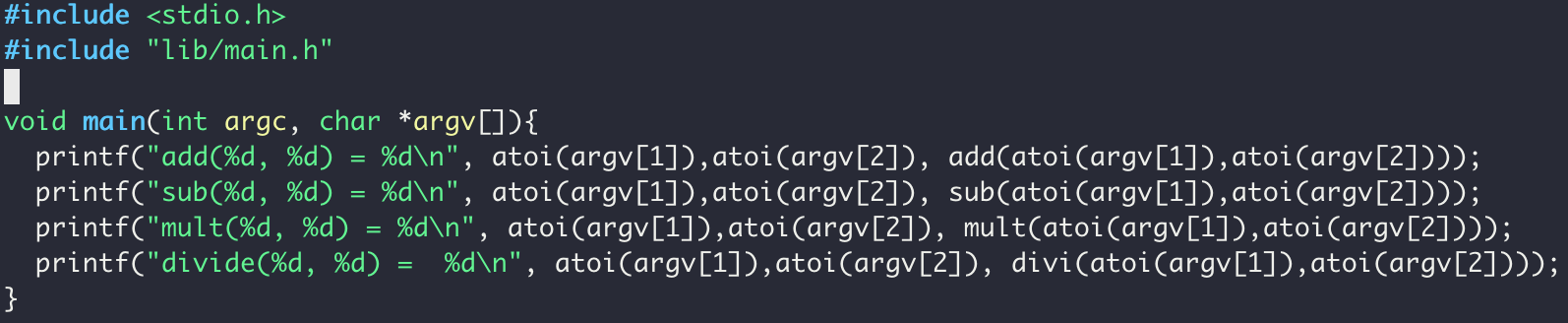


그림 7 main.c 소스

정적 라이브러리

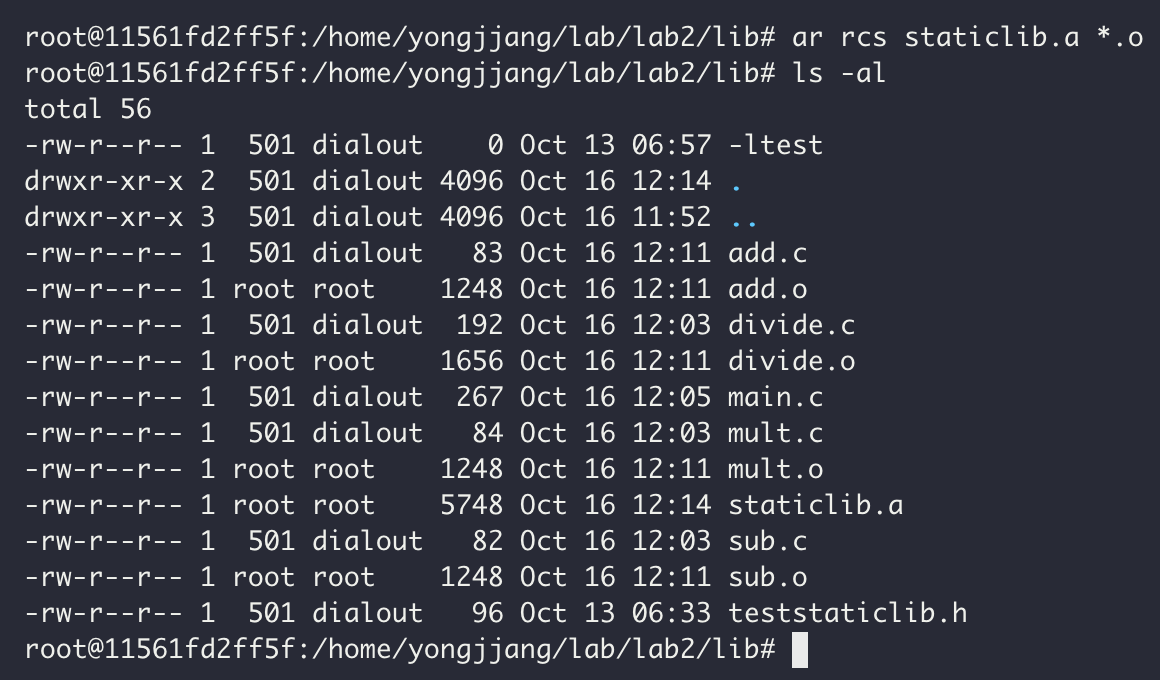


그림 정적 라이브러리 생성

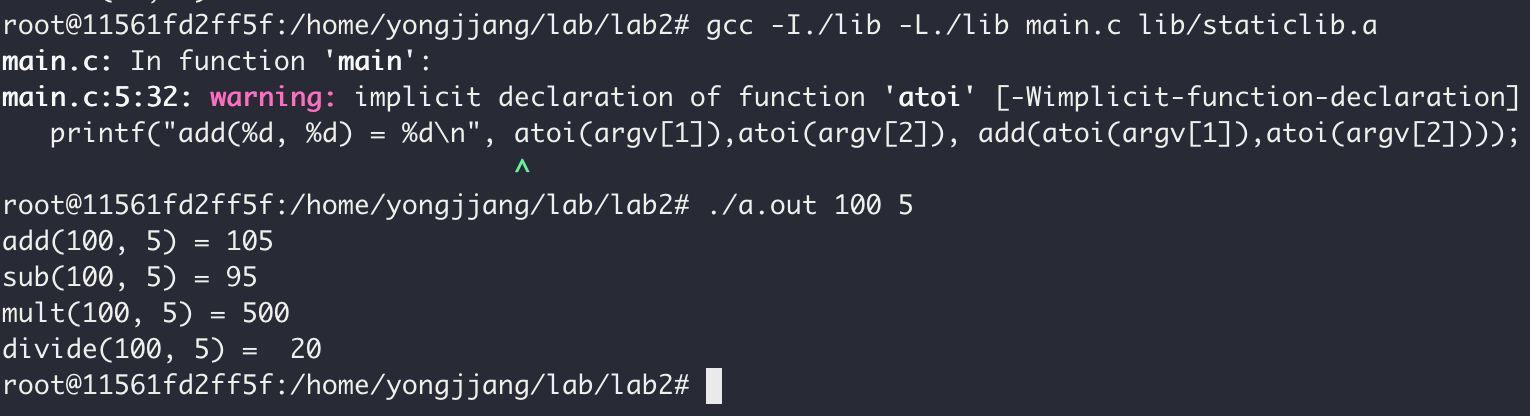


그림 . 정적 라이브러리 실행 파일 생성

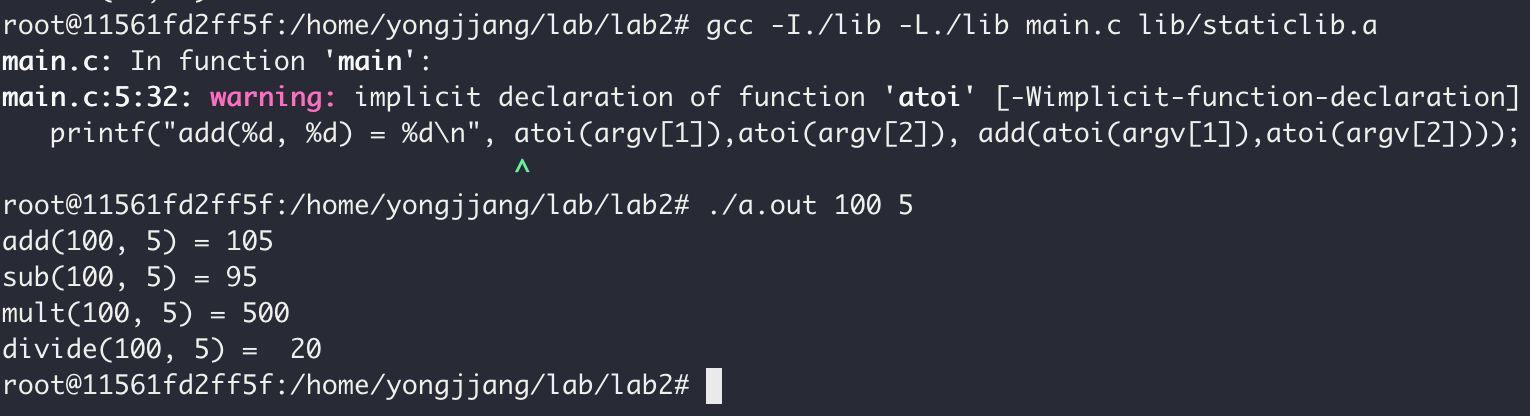


그림 . 정적라이브러리 사용

공유 라이브러리

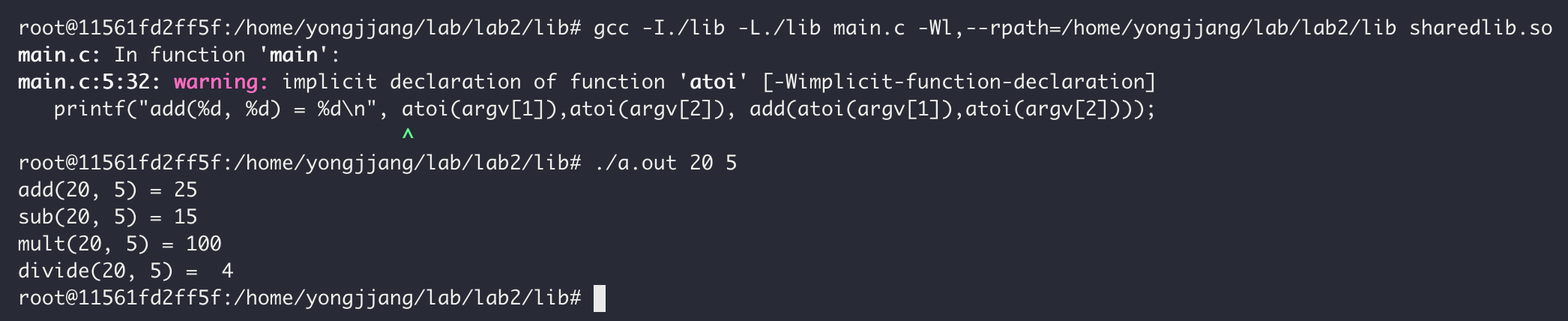


그림 . 공유 라이브러리 생성

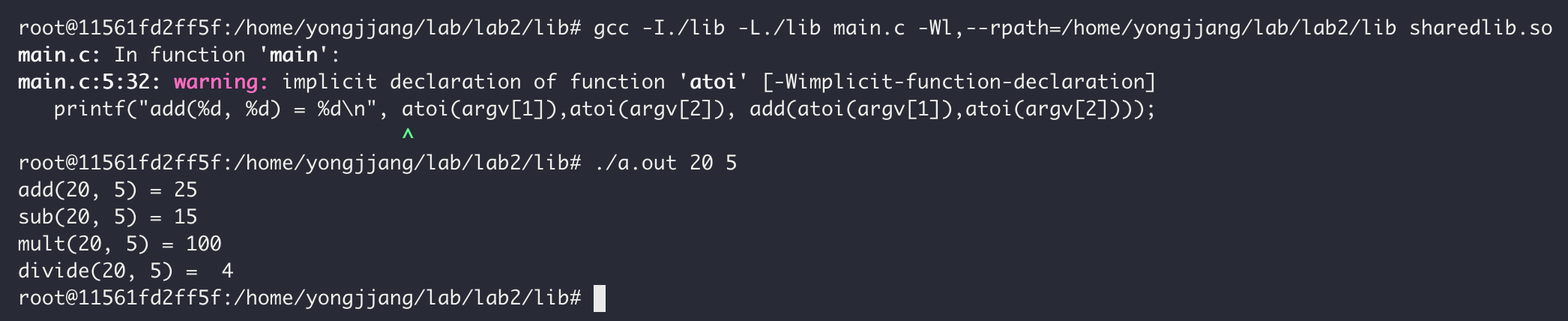


그림 . 공유 라이브러리 실행 파일 실행

동적 라이브러리

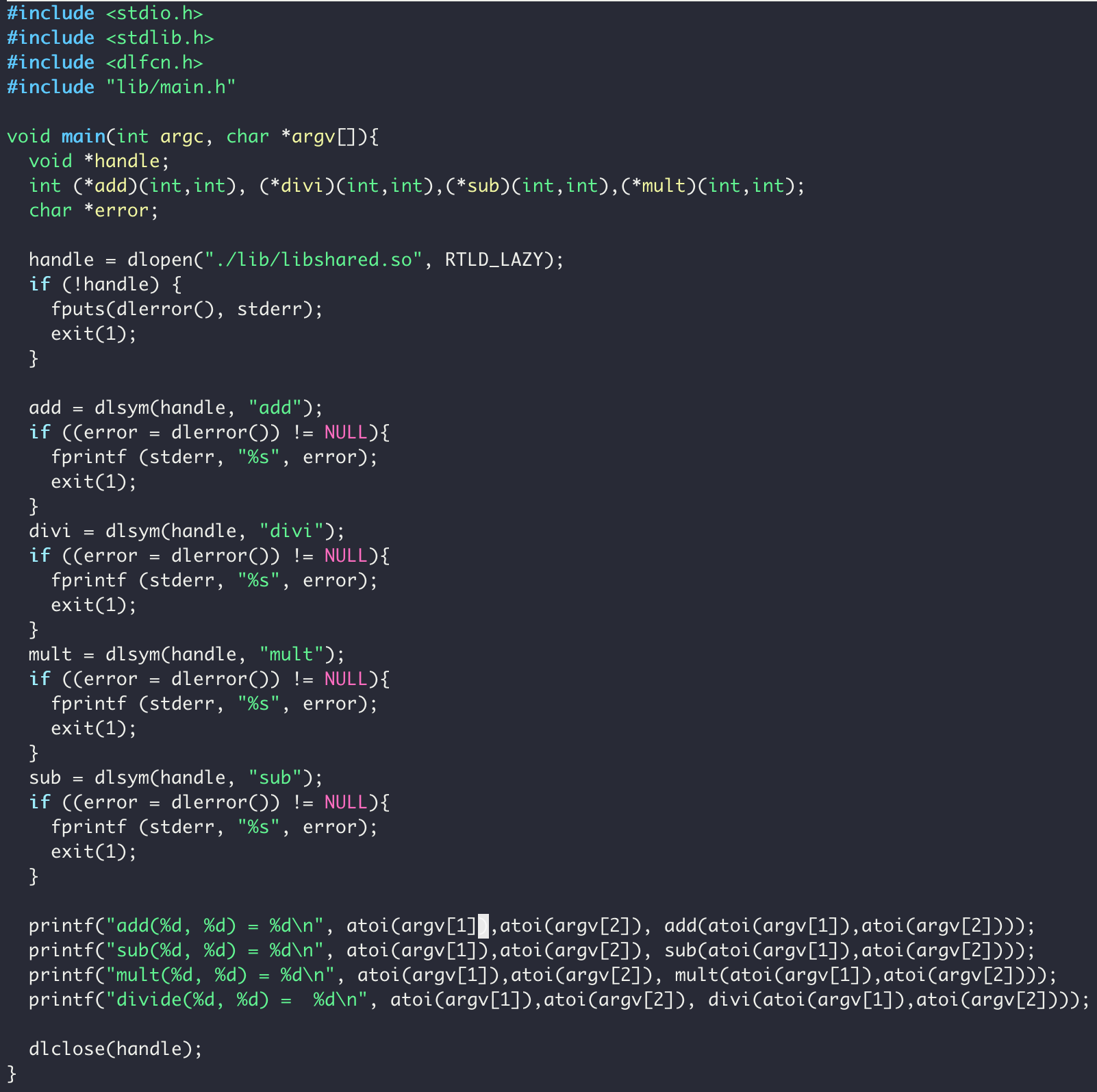


그림 동적 라이브러리 사용 main 소스



그림 동적 라이브러리 실행 파일 실행

## 위 문제에 대하여, C 소스 파일들을 목적 파일로 만들고, 또한 이 파일들을 이용하여 라이브러리 또는 실행 파일로 만들어 주는 Makefile을 작성하여 실행시켜 보시오. 이 때, 라이브러리 관련 파일들은 서브 디렉토리에 두도록 한다.

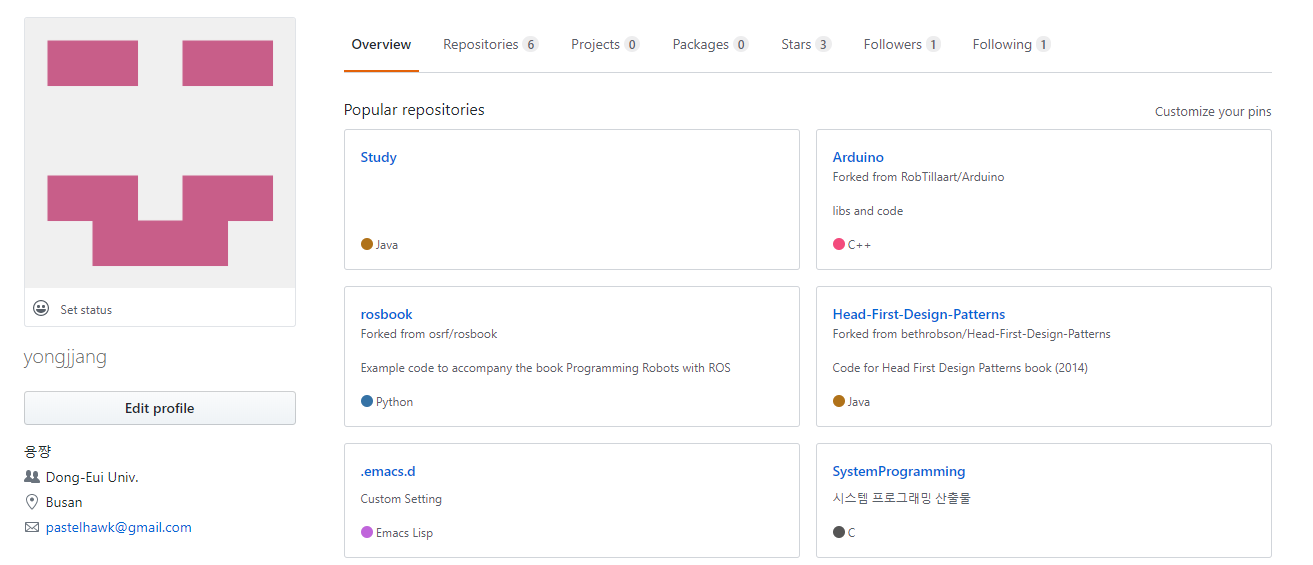
|  |
| --- |
| **정적 라이브러리 Makefile** |
|  |
| 정적 라이브러리 Makefile 실행 화면 |
| 그림 . Make 실행 이전 디렉토리 구조    그림 make 실행    그림 . Make 실행 후 디렉토리 구조    그림 . Make clean 실행, 실행 후 디렉토리 구조 |

|  |
| --- |
| **공유 라이브러리 Makefile** |
|  |

|  |
| --- |
| **공유 라이브러리 Makefile 실행 화면** |
| 그림 . Make 실행 전 디렉토리 구조 |

|  |
| --- |
| 그림 . Make 실행 |
| 그림 . Make 실행 후 생성된 파일들  실행 파일 실행 화면 |
|  |

## GIT의 사용법을 정리하고, Github 사이트에 자신의 계정을 만들고 아래의 모든 과제를 lab2 프로젝트에 올리시오.





## GDB 와 DDD의 사용법을 정리하고, 위 2번 실습 예제에 대하여 GDB 및 DDD 테스트 결과를 나타내시오.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| GDB (GNU DeBugger)  - GNU 프로젝트의 디버거  - C, C++, Objective-C, Modular 2, Ada 등의 언어 지원  - 기능  - 프로그램을 시작한다.  - 프로그램이 특정 조건을 만족할 때 정지시킨다.  - 프로그램이 정지했을 때 어떤 일이 벌어졌는지 조사한다.  - 프로그램 일부를 수정하여 발견한 버그를 제거했을 때의 실행 결과를  미리 알아볼 수 있게 한다.  EMB00002a78540f  EMB00002a785410   |  |  | | --- | --- | | break n | n번째 줄에 정지점 설정 | | run | 디버깅 시작을 위하여 프로그램 실행 | | next | 한 단계 실행 | | quit | gdb 종료 | | step | 다음 줄에 도달할 때까지 프로그램을 진행.  next와는 다르게 함수를 만나면 함수로 제어가 넘어감. |   EMB00002a785411  EMB00002a785412EMB00002a785413  EMB00002a785414C 선행처리기를 활용한 디버깅 코드  EMB00002a785415 |



그림 . 공유 라이브러리 gdb 실행창

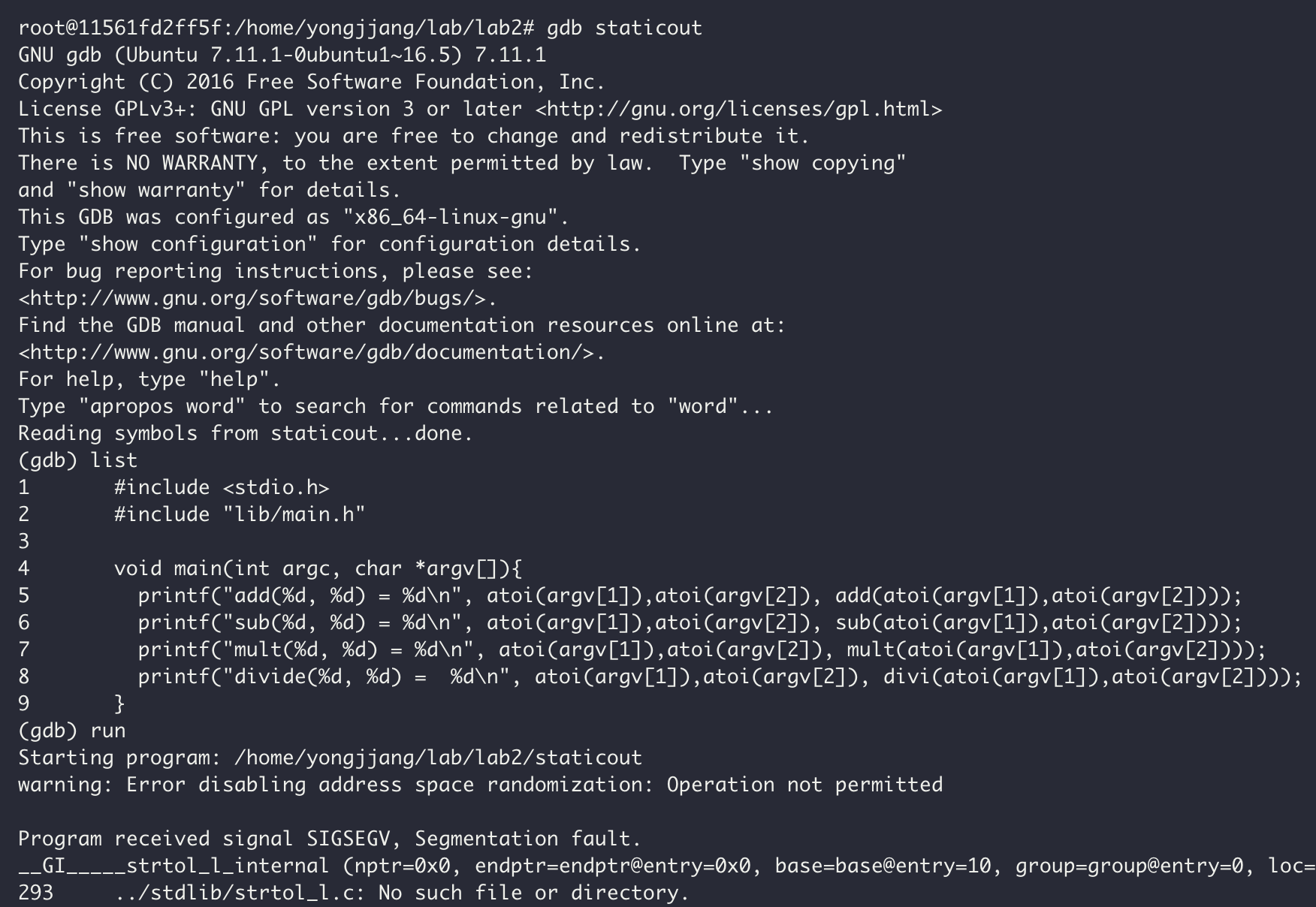


그림 . 정적 라이브러리 gdb 실행창



그림 . 동적 라이브러리 gdb 실행창

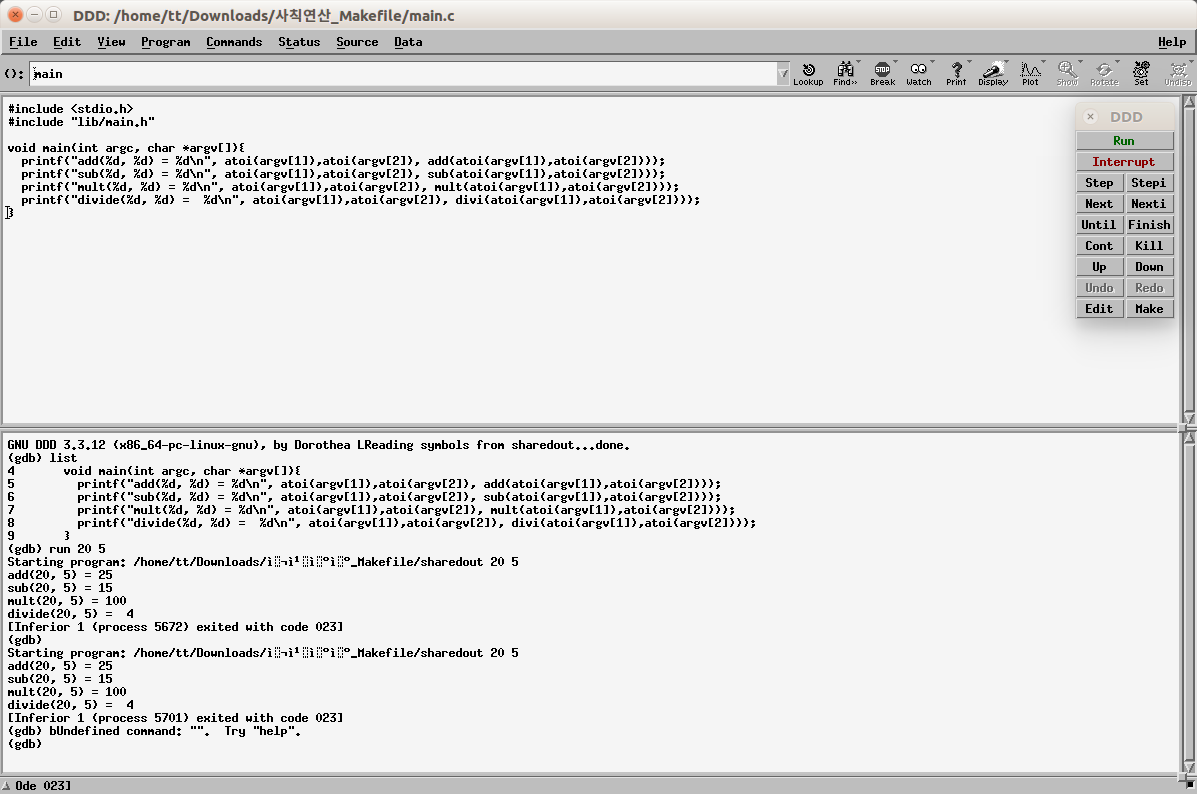


그림 . DDD에서 sharedlib out 파일 실행

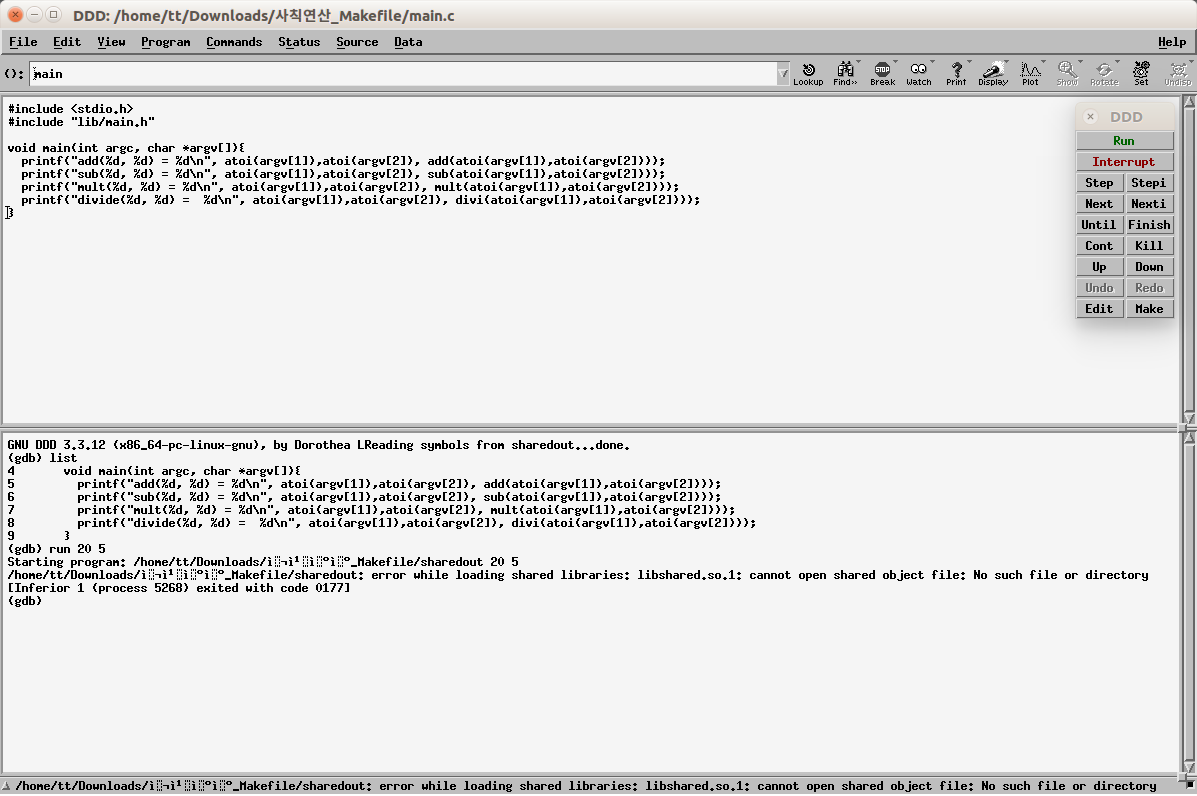
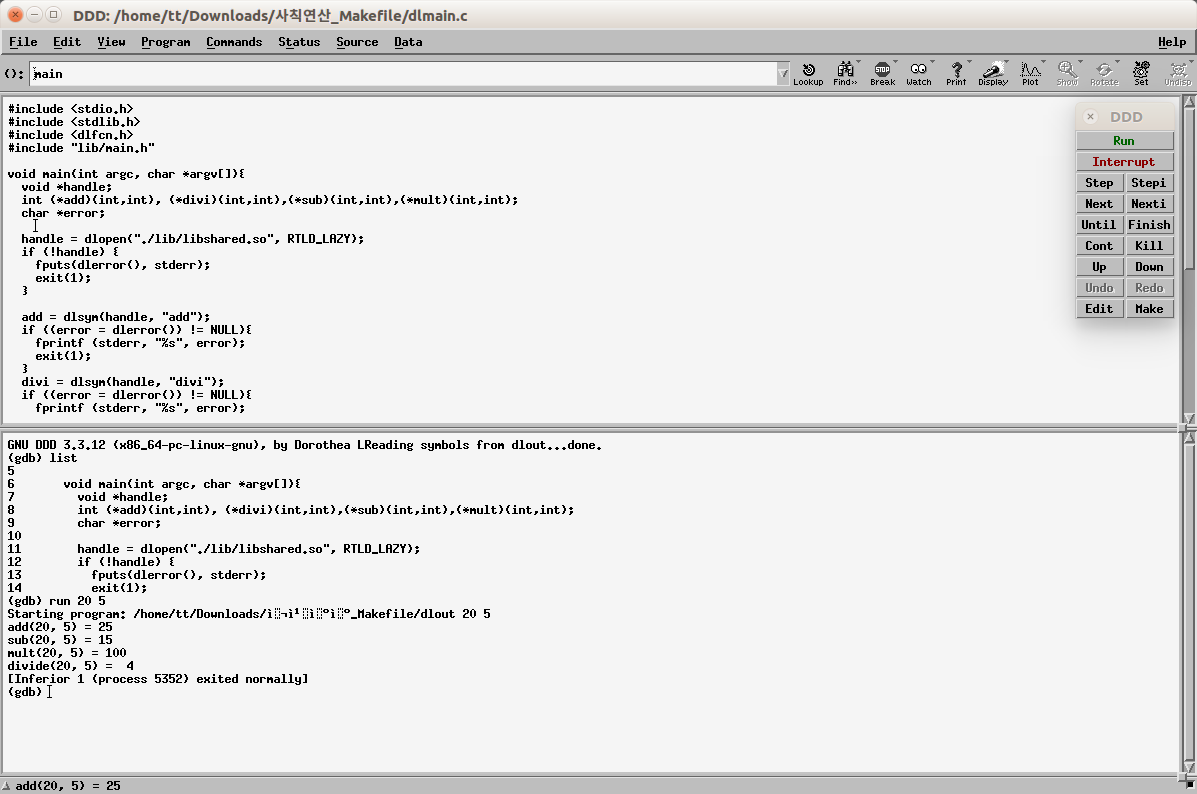


그림 . DDD에서 staticlib out 파일 실행

그림 . DDD에서 dynamiclib out 파일 실행

## perror() 함수 구현 : 라이브러리 함수인 perror() 함수와 동일하게 동작하는my\_perror() 함수를 구현하고, 이를 이용하여 예제 13에서 perror() 함수를 대체하여 프로그램을 실행해 보자.

|  |
| --- |
| **Perror.c** |
|  |
| **Perror.h** |
|  |
| **Perror\_use.c** |
|  |
| **Gcc -c perror.c** |
|  |
| **Gcc perror\_use.c perror.o** |
|  |
| **실행 화면** |
|  |

## assert() 함수 구현 : 라이브러리 함수인 assert() 함수와 동일하게 동작하는my\_assert() 함수를 구현하고, 이를 이용하여 5장 예제 14에서 assert() 함수를 대체하여 프로그램을 실행해 보자.

|  |
| --- |
| **My\_assert.c 소스 파일** |
|  |
| **실행 화면** |
|  |

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

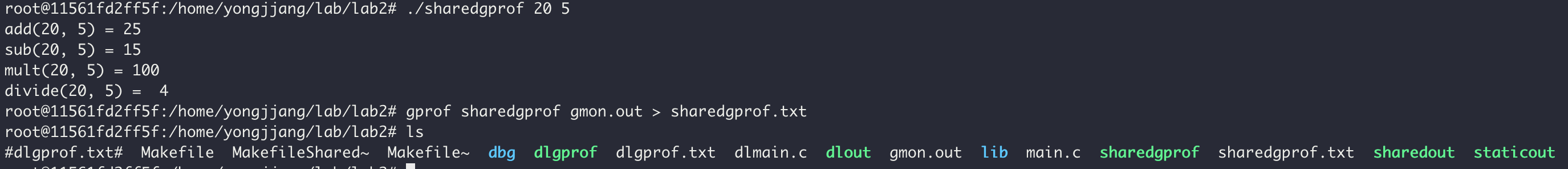


그림 . Shardgprof 실행 화면

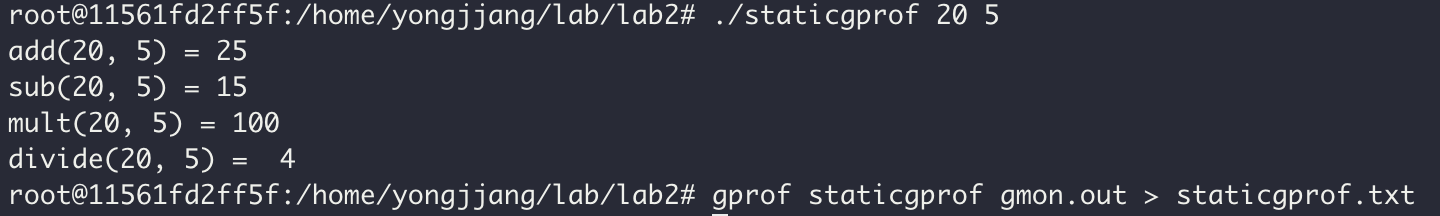


그림 staticgprof 실행 화면

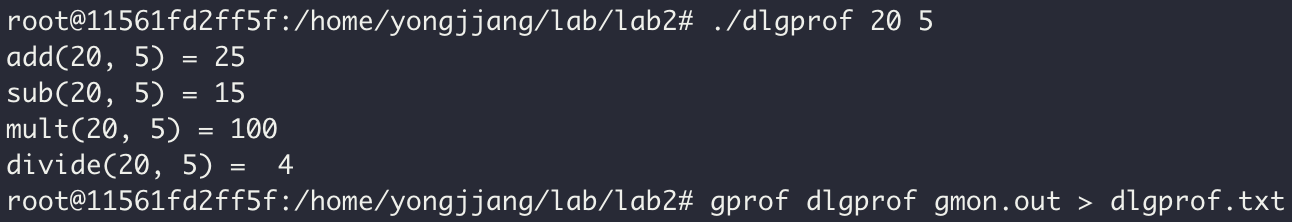


그림 dlgprof 실행 화면

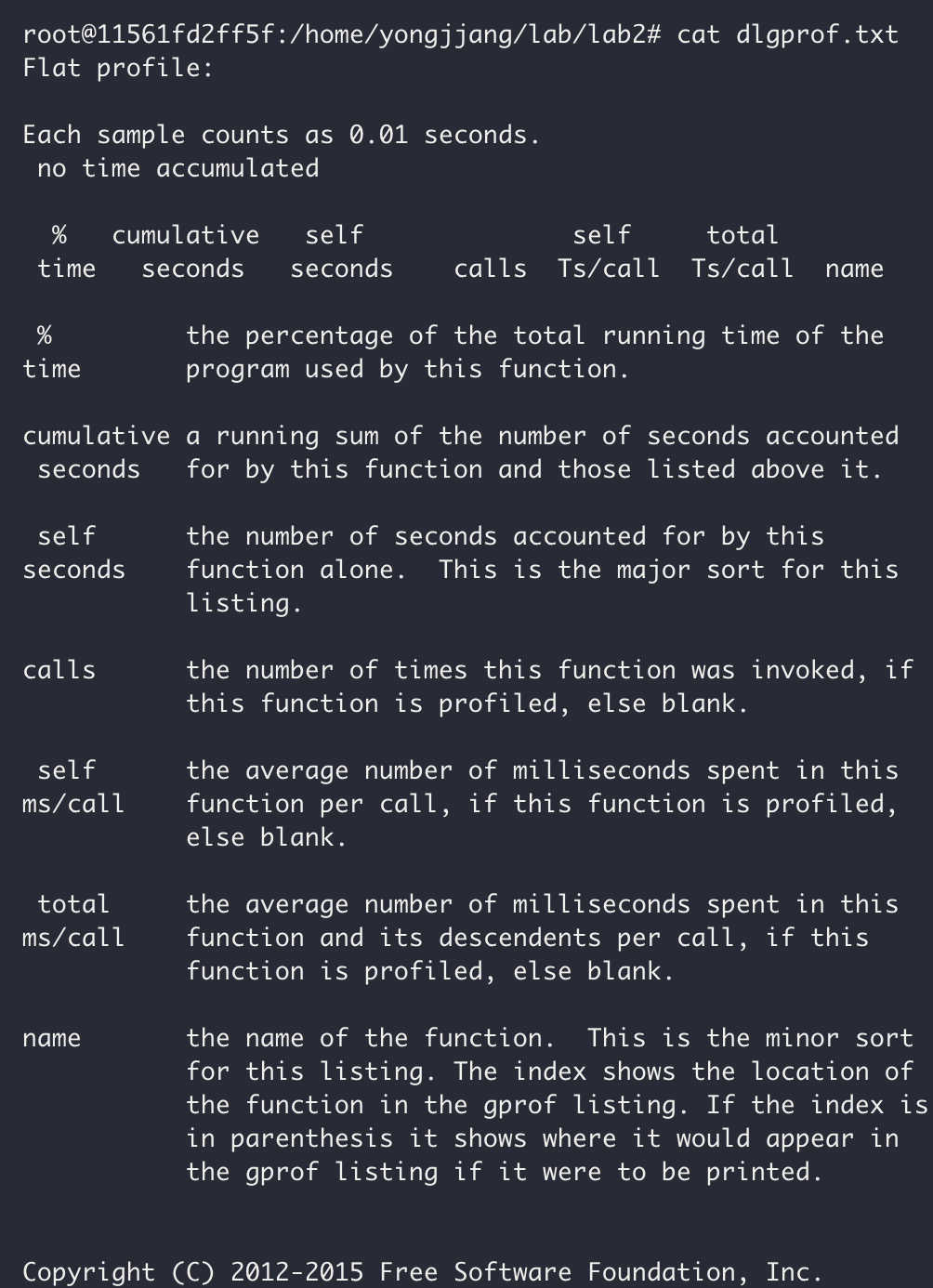


그림 gprof산출물

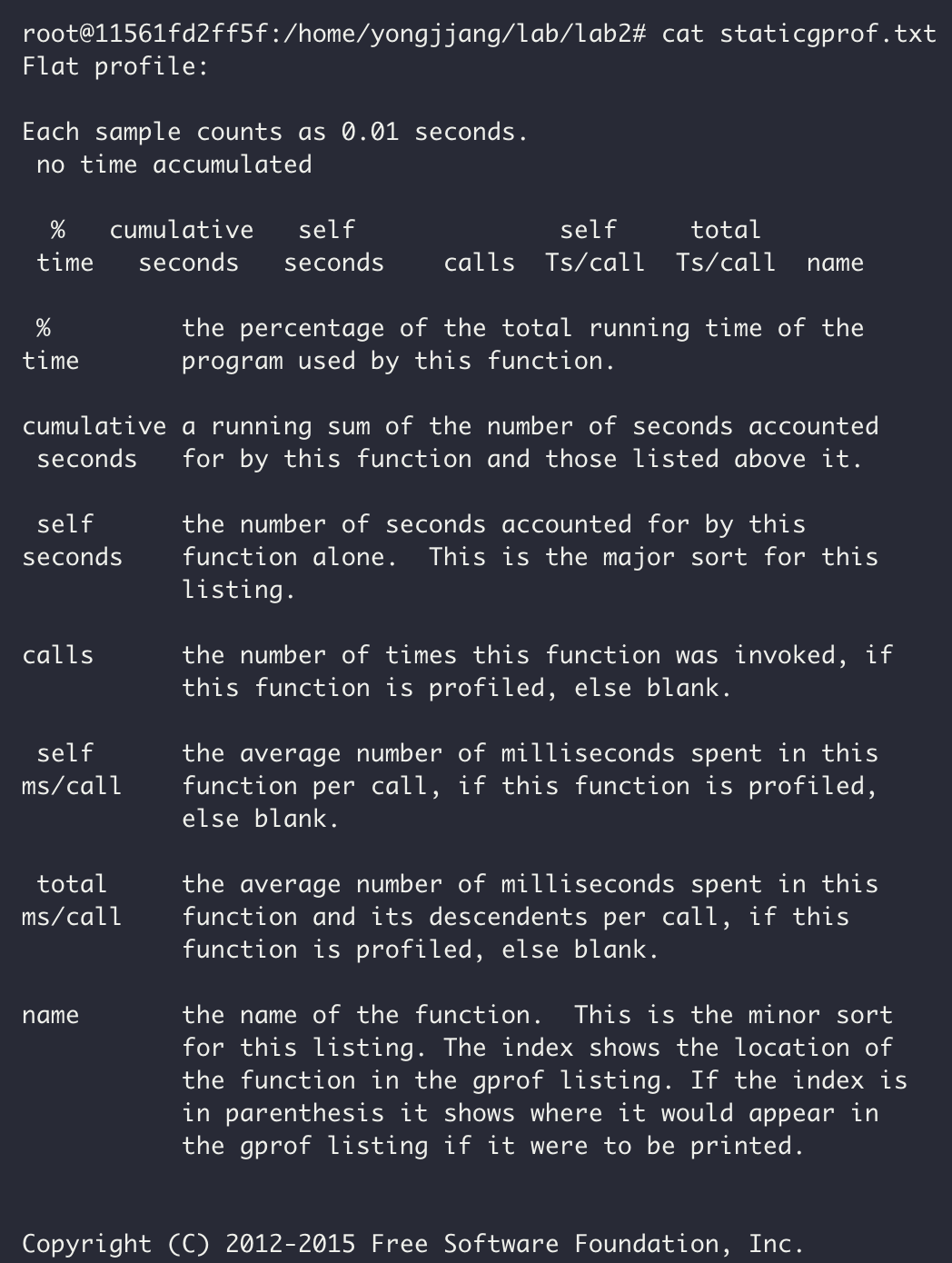


그림 gprof산출물

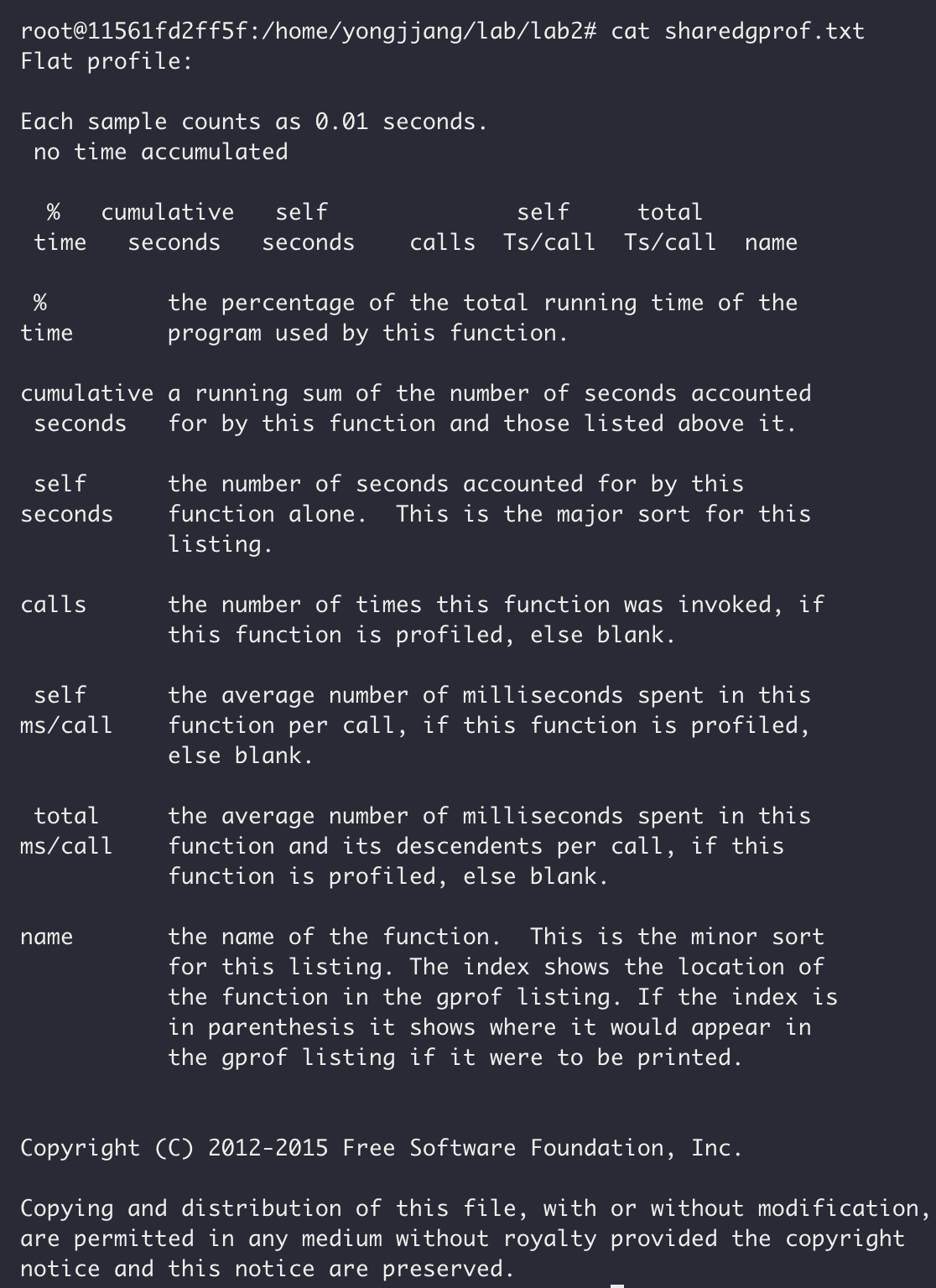


그림 gprof산출물

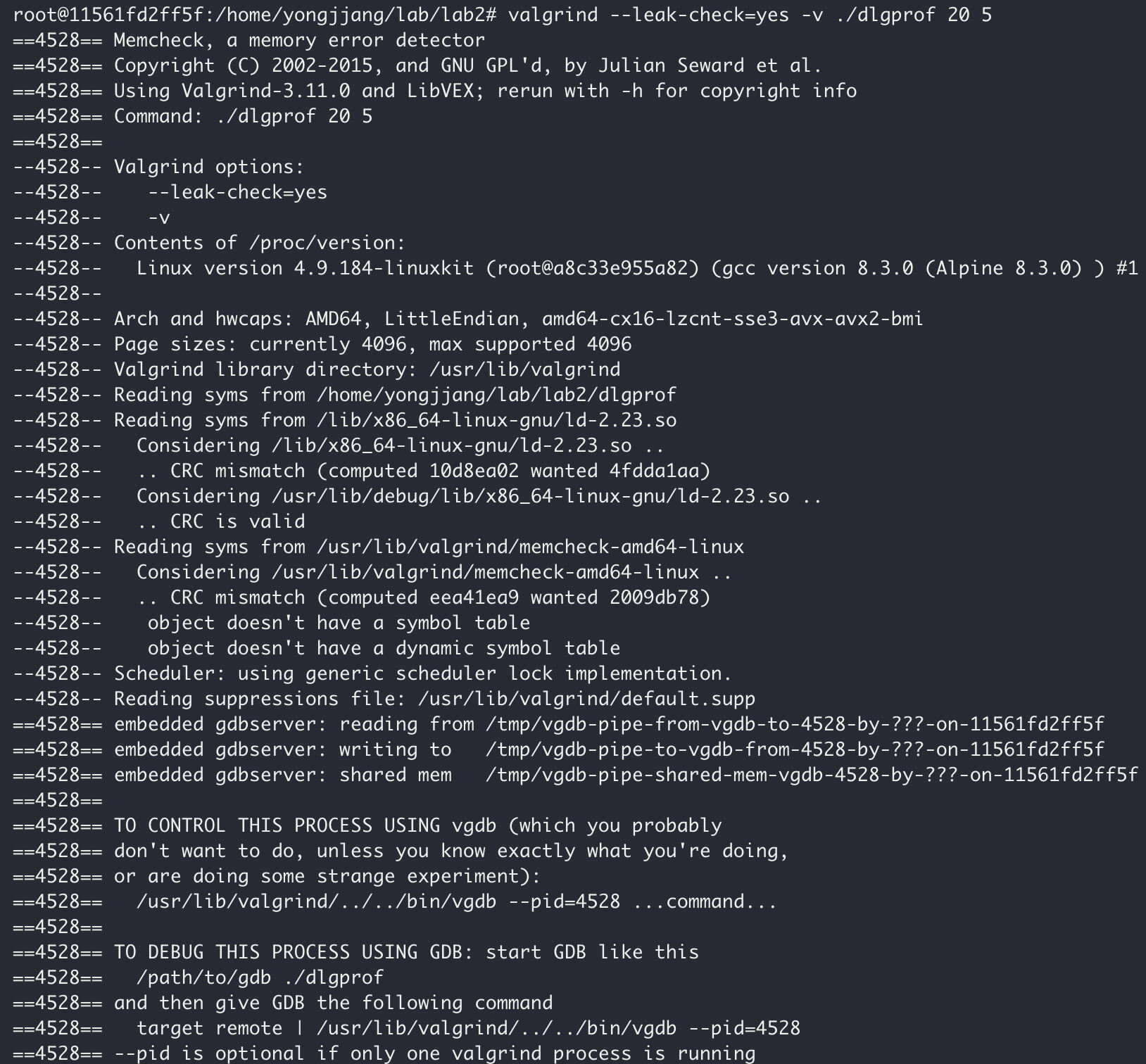
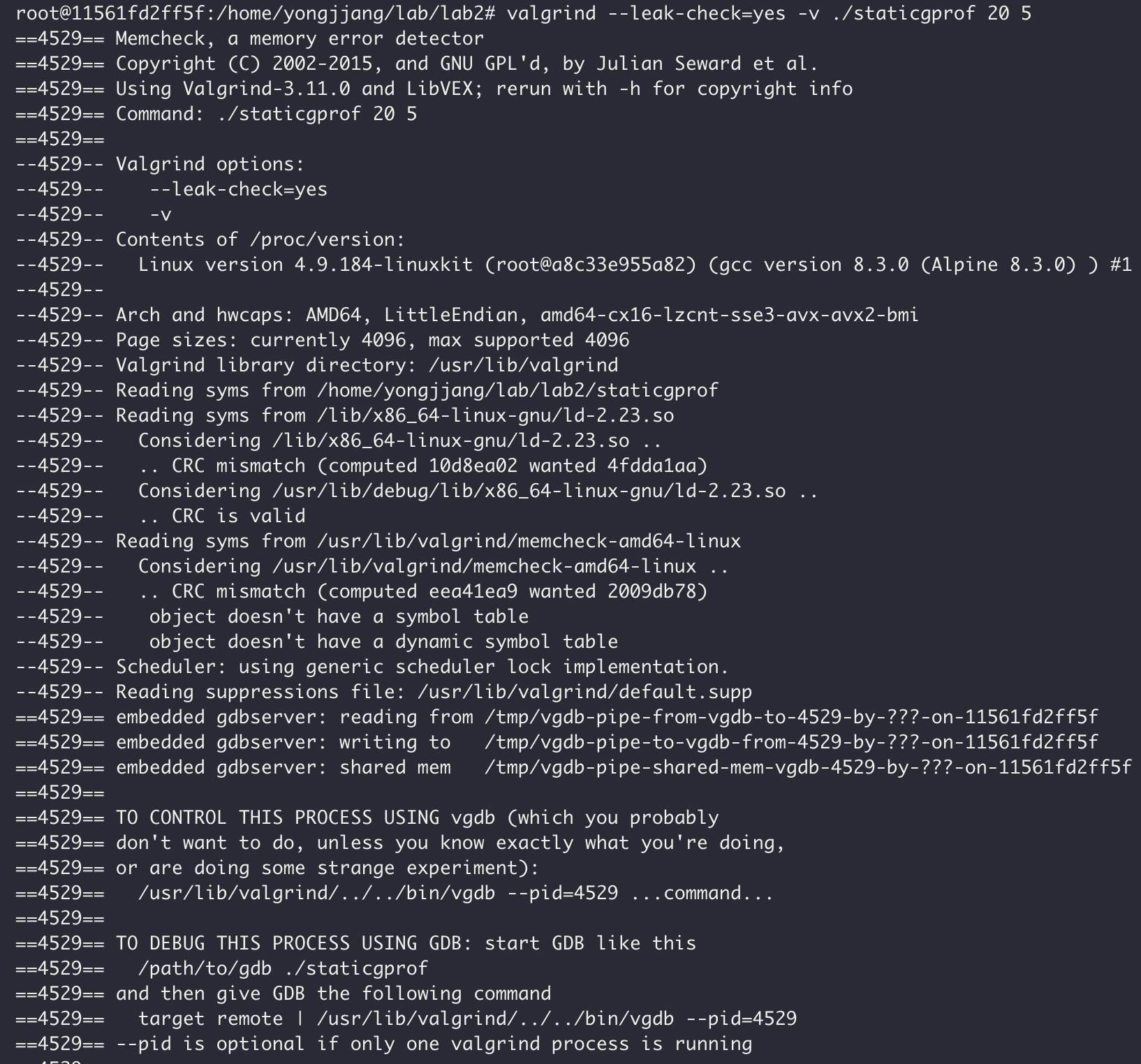
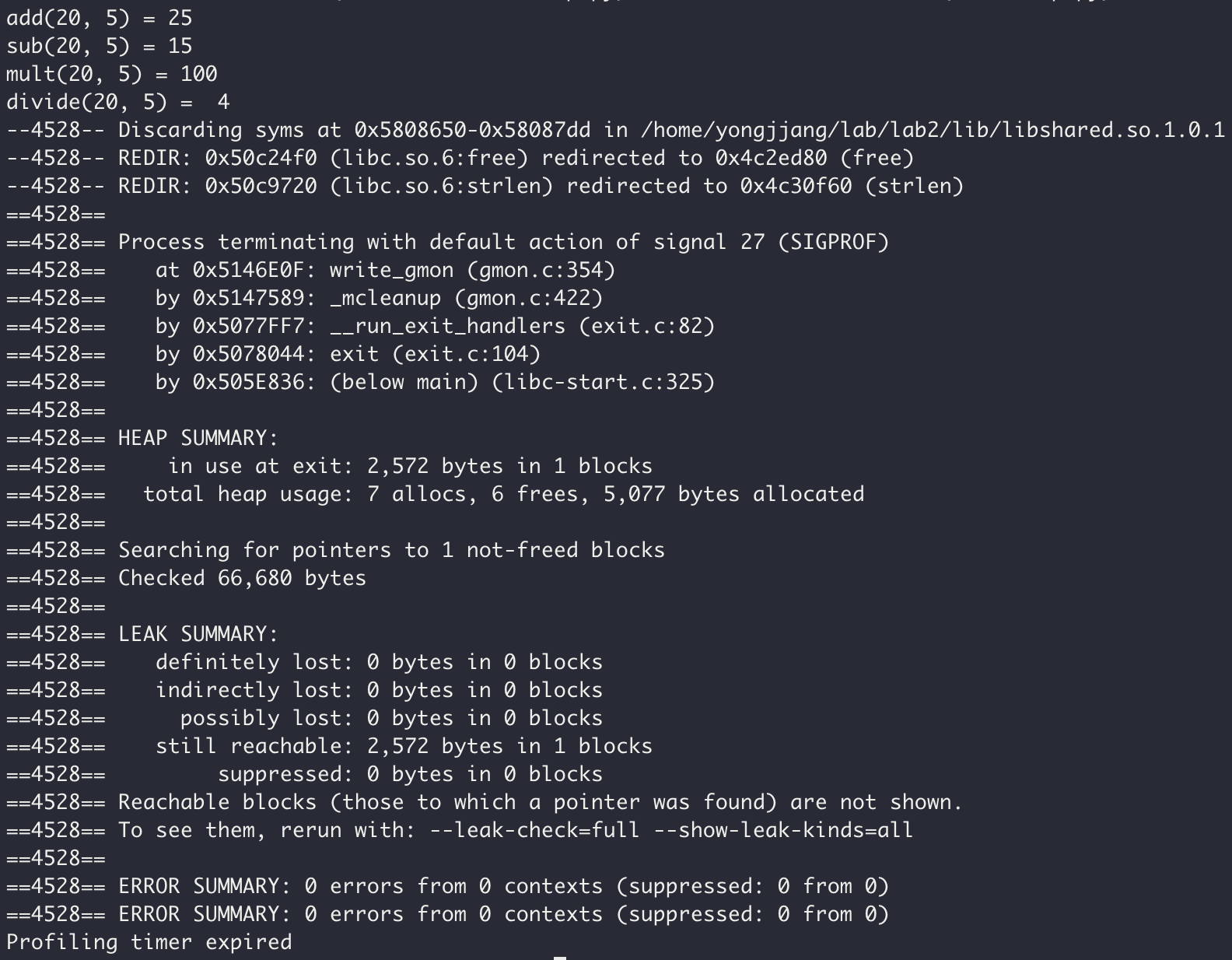


그림 . Valgrind dyname lib 실행 파일 메모리 누수 체크

 그림 . Valgrind static lib 실행 파일 메모리 누수 체크

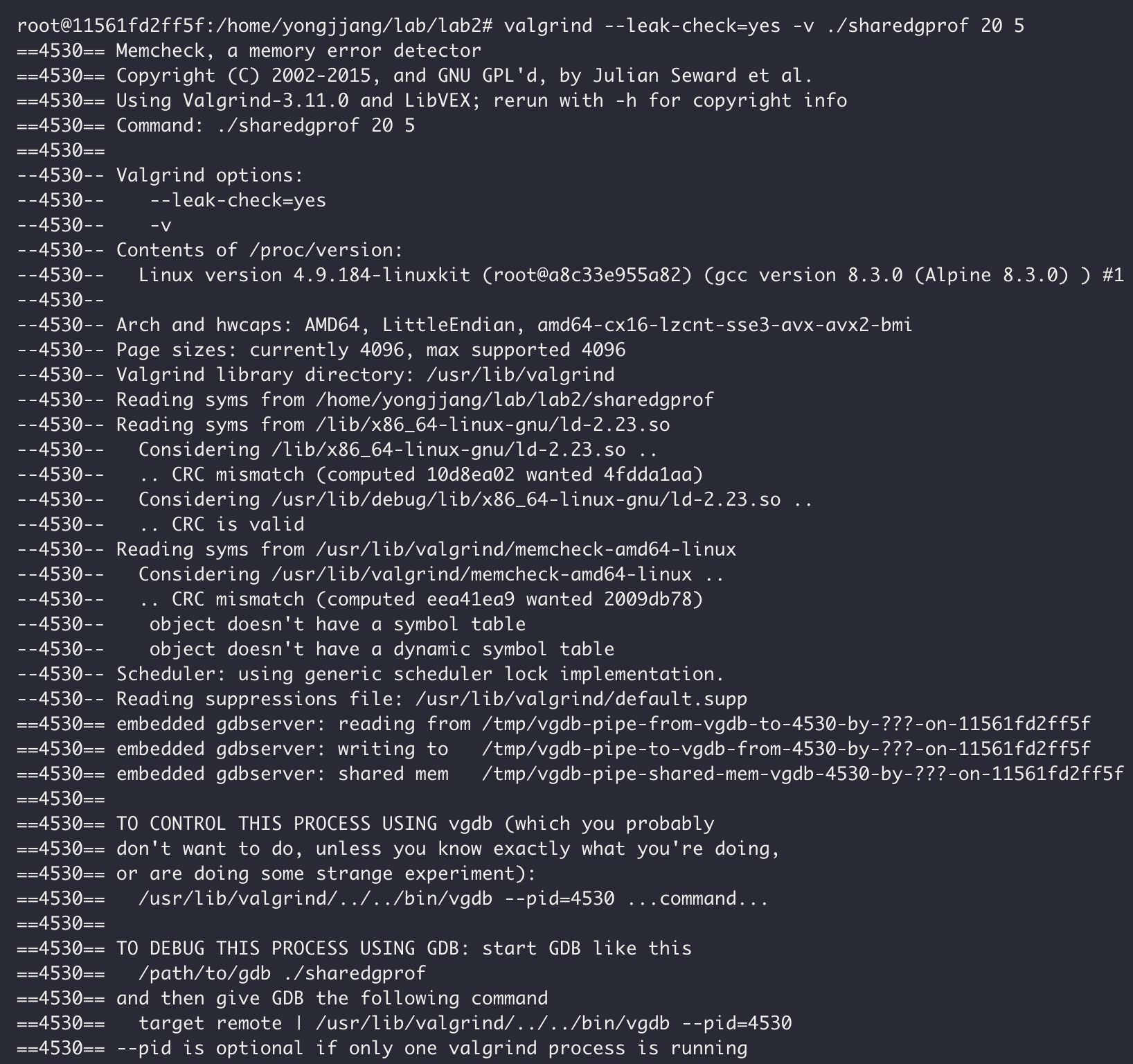
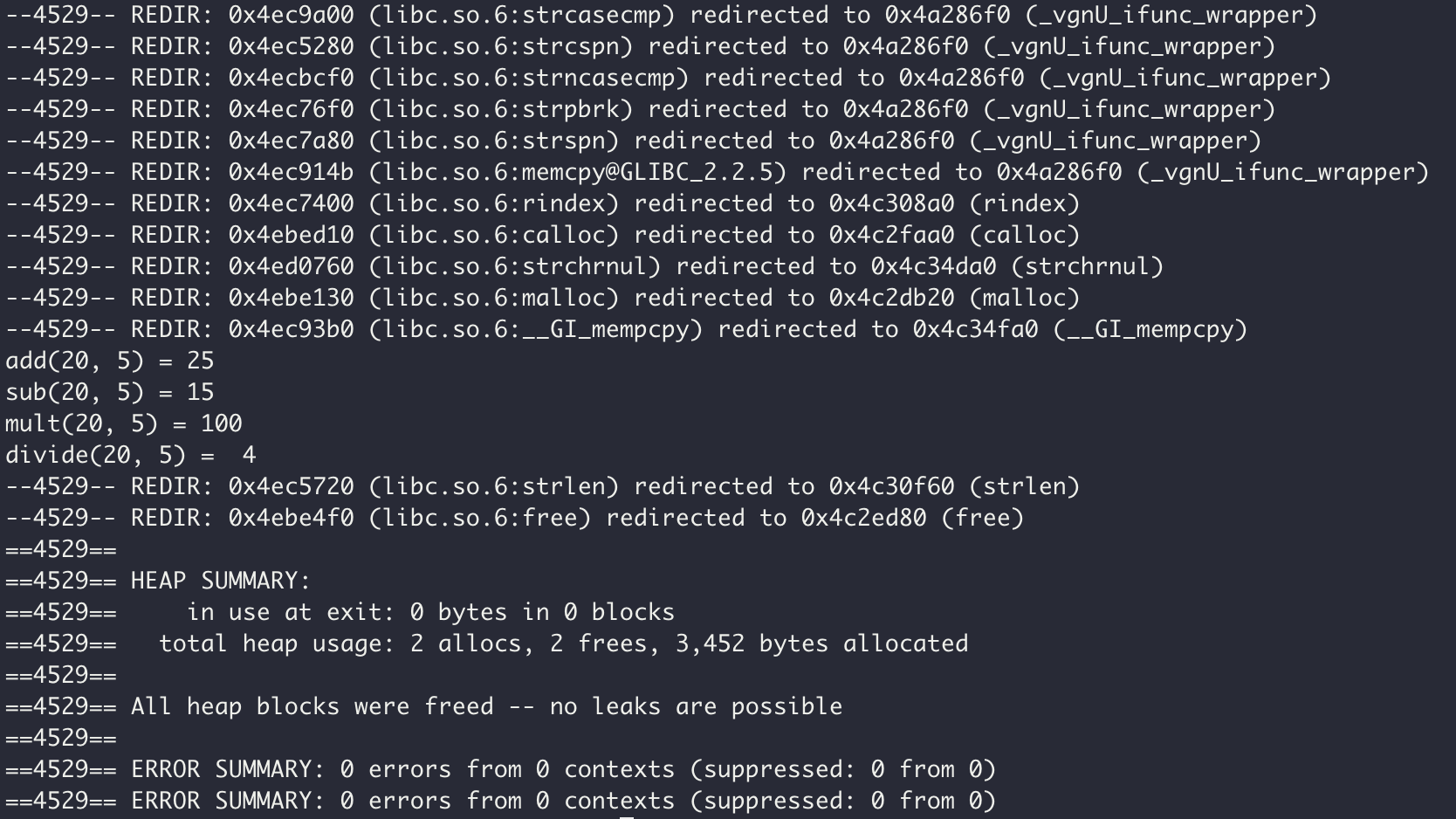
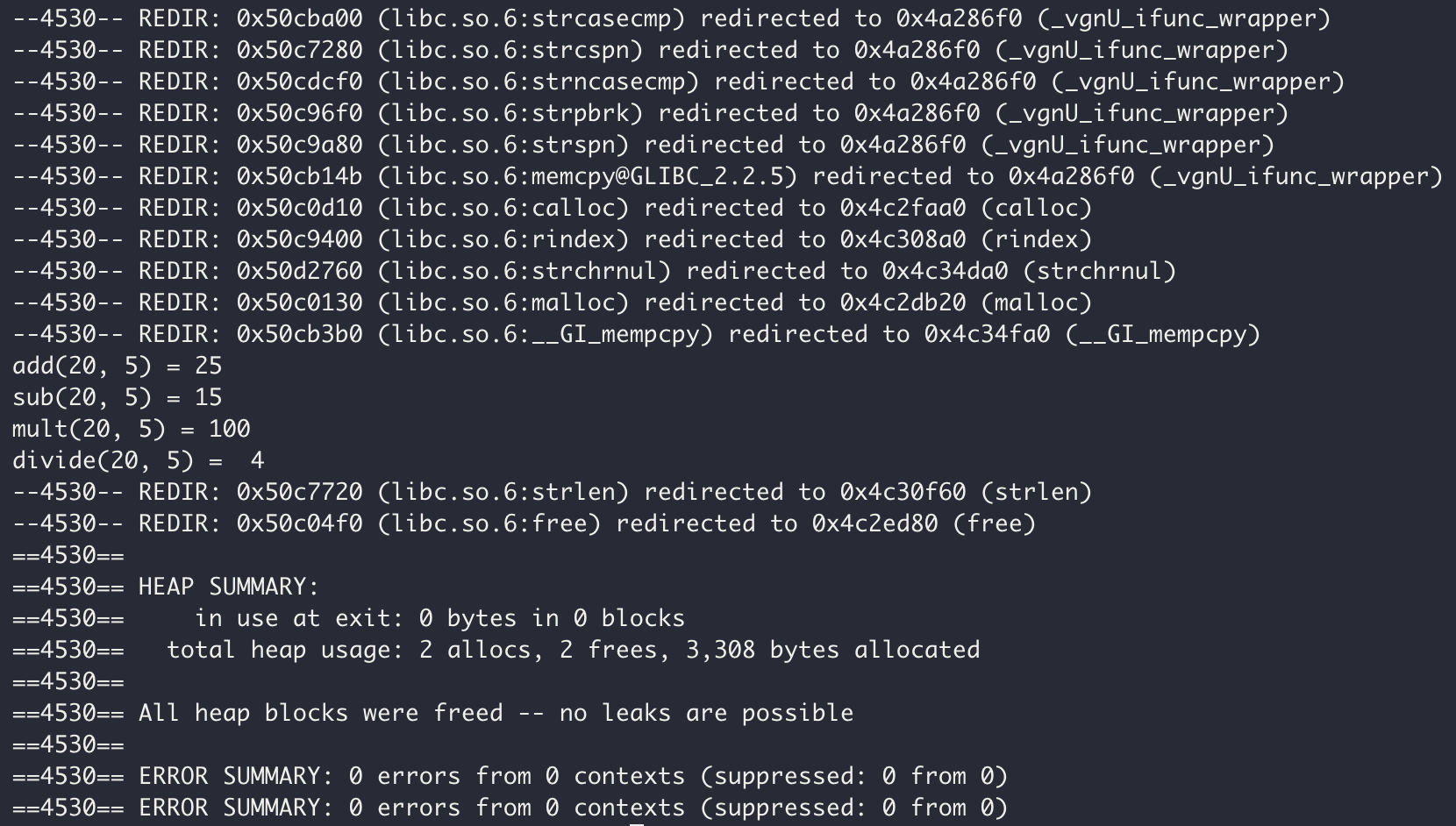
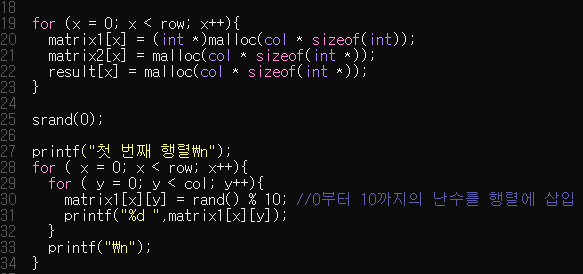
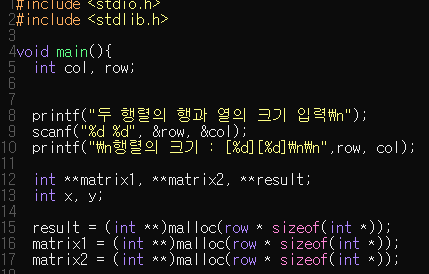


그림 . Valgrind shared lib 실행 파일 메모리 누수 체크



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



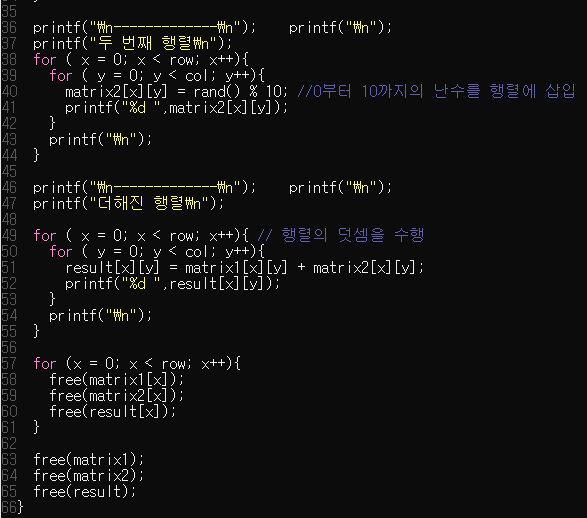
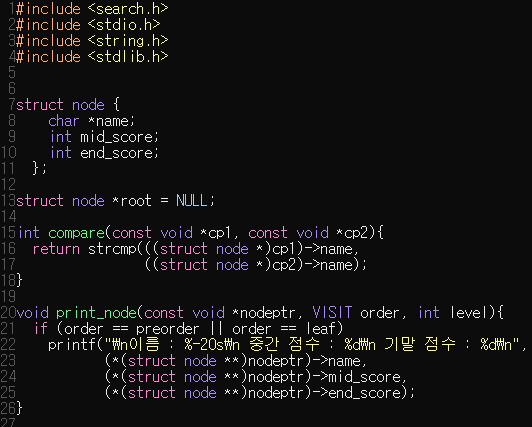


그림 . 7. 8번 문제 소스



그림 . 결과

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



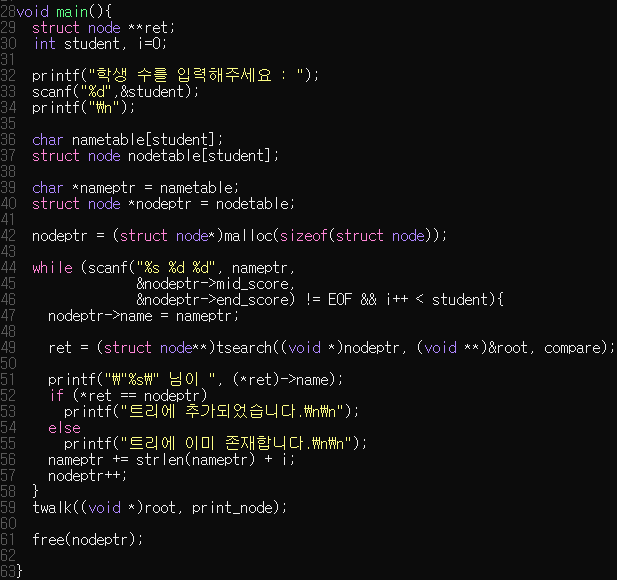


그림 39 . 동적 할당 트리 프로그램 소스 코드

실행 결과

|  |
| --- |
| 학생 수 : 1. 입력 값 : yongjjang 100 100 |
|  |
| 학생 수 : 2, 입력 값 : 아래 참조 |
|  |