1. 자신의 홈 디렉토리(**$cd** [엔터])에서 강의노트 0-2. Unix Shell의 맨 마지막 페이지(Resource File 복사하기)를 참고하여 교수님 디렉토리에 있는 .bashrc, .vimrc, .profile 파일을 자신의 홈 디렉토리로 복사하라. 복사한 후 logout하고, 다시 login하라. 이제 명령 프롬프트가 자신의 계정이름과 현재 작업 디렉토리만 보일 것이다.

   자신의 홈 디렉토리에서 **$ cat .vimrc**에서 행번호 보기, 탭크기 4, 자동인덴트가 설정하는 내용이 있어야 한다. 또한 **$alias** [enter]를 실행시켜 보면 **l, la, ll, ls, rm** 등의 명령어 별칭이 미리 설정되어 있는 것을 확인할 수 있다. 즉, 앞으로 **ls** 하면 자동으로 **"ls --color=auto"**가 실행되고 **"rm"** 하면 **"rm -i"**가 실행된다. **la**와 **ll**도 필요한 경우 사용하라.

   주의: 위 세 개의 파일을 제대로 복사하지 않은 경우 향후의 모든 실습을 정상적으로 수행할 수 없다.

   ---

2. up 밑에 pr3 디렉토리를 만들고, 그 디렉토리로 이동하라.

   $ cd ~/up **$ pwd** (up여야 함)
   $ mkdir pr3 **$ ls** (pr1, pr2, pr3가 있음)
   $ cd pr3 **$ pwd** (pr3여야 함)

3. pr3 디렉토리에서 교수님 폴더에 있는 test1.c 파일을 복사하라. **(**아래에서 **.** 을 잊지 말 것**)**

   $ cp /home/jhshim/up/pr3/test1.c . (. 조심)
   $ ls (test1.c가 존재해야 함)

4. 소스파일 test1.c를 이용하여 바로 test1라는 실행파일을 만들어라. 즉, test1.c -> test1 생성(강의노트 참조)

   $ gcc [실행파일과 소스파일 이름을 옵션과 함께 지정하라]

   // 어떠 에러 메시지도 출력되지 않아야 정상임

   1) 실행 파일 test1이 만들어졌는지 확인
      $ ls -l
      // test1이 존재해야 하고, 실행파일이어야 함. 즉, 왼쪽 끝의 접근 권한에 실행파일을 의미하는 x가 들어가 있어야 함(예, rwxrwxr-x)
   2) 생성된 실행파일 test1을 실행시켜 본다.
      $ test1 [enter]
      (test1 실행 후 다음을 입력하라. 각 연산자 앞뒤에 반드시 공백이 있어야 함. 예, " + ")

      Expression: 2 + 3[enter] (수식 입력**)**

      Result: 2 + 3 = 5 (출력된 계산 결과**)**
      Expression: 2 - 3
      Expression: 2 * 3
      Expression: 2 / 3

      Expression: 2 % 3 (구현되지 않아 에러 발생할 것임)

      Expression: exit (종료하고자 할 때, 기억해 둘 것)
   3) 위 2)에서 exit를 입력하여 test1을 종료한다. 그런 후 아래 명령어는 앞 전에 실행한 명령어를 재실행하는 방법이다. 유용하니 잘 활용하기 바란다.

      $!! (직전의 명령어 재실행)
      $!g (**g**로 시작하는 명령어 재실행; 주로 **gcc** 임)
      $!l (**l**로 시작하는 명령어 재실행: **ls**임)
      $!t (test1임)
      (앞 전에 실행한 명령어가 **gcc**와 **grep**이 있다면 두 단어가 구분되는 시점까지 **!gc** 를 주어야 함)

   또는 위쪽 화살표 키를 눌러 앞 전에 실행한 명령어를 다시 불러 와서 엔터 치면 된다. 계속 위쪽 화살표 키를 누르면 계속 그 앞 전의 명령어를 볼 수 있다.

5. 이제는 컴파일을 먼저 하여 목적어 파일을 만들고, 생성된 목적어 파일을 이용하여 실행파일을 만드는 과정을 실습할 것이다. 먼저 위 파일을 컴파일만 하라. 생성된 목적어 파일을 확인하라. 즉, test1.c -> test1.o 생성(강의노트 참조).

   $ gcc [컴파일 옵션과 소스파일 이름 지정할 것]
   // 어떠 에러 메시지도 출력되지 않아야 정상임
   $ ls (목적어 파일 test1.o가 존재해야 함)

6. 위에서 컴파일된 목적어 파일 test1.o을 이용하여 링크를 해서 **test2**라는 실행파일을 만들어라. 실행파일 이름이 test1이 아니라 **test2**임을 명심할 것. 즉, test1.c -> test1.o -> test2 생성(강의노트 참조).

   $ gcc [실행 파일과 목적어 파일이름을 옵션과 함께 지정]
   // 어떠 에러 메시지도 출력되지 않아야 정상임
   1) 실행파일 test2가 생성되었는지 확인
      $ ls -l
      // test2가 존재해야 하고 실행파일이어야 함
   2) 생성된 실행파일 test2를 실행시켜 본다.
      $ test2 [enter]
      (위 실습 4.2의 test1처럼 입력하여 테스트해 본다.)

7. 이제부턴 터미널 창을 하나 더 띄워 로그인 한 후

   $ cd ~/up/pr3 (pr3로 이동) **$ pwd** **$ ls**

   를 실행하라. 이 터미널 창(vi 창)에선 항상 vi 에디터로 소스만을 수정할 예정이다. 기존의 다른 터미널 창(명령어 창)에선 컴파일 등 일반 명령어를 실행하면 편리하다. 이유는 컴파일 에러 발생시 에러 원인을 보면서 vi 창에

서 소스 수정이 가능하기 때문이다. 새로운 터미널 창(vi 창)에서 $ vi test1.c를 실행하라. 그런 후 변수 oprd1을 oprd로 아래처럼 변경하여 컴파일 에러를 의도적으로 발생시켜 보자.

```
int oprd, oprd2, res;
```

1) 수정 후 **:w[enter]**로 저장만 하고 vi에서 빠져 나오지 마라. 그리고 다른 터미널 창(명령어 창)에서 test1.c를 실습 4번처럼 gcc 명령어를 이용하여 소스파일을 컴파일 해서 바로 실행파일 test1을 만들어 보라. 다음의 에러 메시지가 나올 것이다.
   test1.c:27: 'oprd1' undeclared
   (그 밑은 이러한 에러 메시지가 여러 번 반복되지만 한번만 출력하였다는 의미이다.)
   이는 27행(행 번호는 다르게 나올 수도 있음)에서 사용된 oprd1 변수가 선언되지 않았다는 에러 메시지이다. 항상 에러 메시지를 유심히 보고 에러가 발생한 행 번호가 몇 번(27행)인지 확인하기 바란다.
2) 이제 vi 창에서 27G로 에러가 발생한 행을 찾아가서 확인한다. 여기선 변수 선언이 잘못된 것이기 때문에 변수선언을 수정해야 한다. oprd1이 선언되지 않았으니 변수 선언한 곳으로 이동하여 아래처럼 원상 복구한다.
   `int oprd1, oprd2, res;`
   이제 vi를 빠져 나오지 말고 **:w**[enter]로 저장만 하라.
3) 다시 컴파일 하던 명령어 터미널 창으로 와서 새로 컴파일 하여 **[실습 4번]**처럼 실행파일을 생성하라. 컴파일 에러가 없어야 한다. **$ ls**로 test1이 생성되었는지 확인할 것. 첫 번째 에러가 뒤쪽 에러에도 영향을 미치므로 에러를 모두 한번에 수정하려 하지 말고, 항상 첫 번째 에러를 수정/저장/컴파일 하여 하나씩 고쳐 나가기 바란다. 참고로 직전에 실행한 명령을 다시 반복할 경우는 **$!! 를, 직전에 실행한 컴파일 명령을 반복할 경우 $!g 를 실행**하면 된다.
4) 생성된 실행파일 test1을 실행시켜 본다.
   $ test1
5) 이처럼 앞으로는 터미널 창을 항상 두 개 실행시켜, 한 쪽 vi 창은 vi로 소스를 수정한 후 **:w** 로 저장하고, 다른 명령어 창은 컴파일 및 일반 명령어를 실행하면 좋다. 이는 컴파일 에러 메시지와 행 번호를 확인하면서 vi 창에서 바로 수정/저장할 수 있기 때문이다.

8. 이번엔 vi 창에서 test1.c 내의 함수 정의에서 **add()**를 **addd()**로 변경한 후, 즉
   `int addd(int a, int b) { return (a+b); }`
   로 수정한 후 저장한다. 함수 호출하는 곳에서는 그대로 add()를 호출하게 둔다. 이후 명령어 창에서 실습 4번처럼 gcc를 이용하여 실행파일 test1을 다시 만들어라. 즉, **($!g)** 아마 아래와 비슷한 에러가 발생할 것이다.

```
/tmp/ccDgtveL.o: In function `main':
test1.c:(.text+0x1c3): undefined reference to
`add'
collect2: error: ld returned 1 exit status
```

   1) 이 에러는 main()에서 함수 add()를 호출했지만 add() 함수가 정의되어 있지 않아 찾을 수 없다는 의미이다. 주로 함수 이름을 실수로 잘못 입력했을 때 발생한다. 예를 들어, printf() 함수를 실수로 print()로 호출했을 때 print() 함수를 찾을 수 없다는 위와 비슷한 에러 메시지가 나올 것이다. **향후 이런 종류의 에러 메시지를 종종 만나게 될 것인데 잘 기억해 두기 바란다.**
   2) 다시 vi 창에서 test1.c 내의 함수 정의에서 **addd()**를 다시 **add()**로 복원한 후 저장한다. 이후 다른 터미널 창에서 실습 4번처럼 gcc를 이용해 실행파일 test1을 다시 만들어라. **($!g)** 정상적으로 test1이 생성되어야 한다. 생성된 실행파일 test1을 실행시켜 본다. **($!t)**

9. 이제 하나의 소스파일을 여러 개의 소스파일들로 분리한 후 실행파일을 만들어 보자. 먼저 아래처럼 test1.c를 add.c 로 복사한 후 vi로 add.c로 들어 간다.

```
$ cp test1.c add.c
$ vi add.c
```

   그런 후 vi 내에서 add() 함수의 정의, 즉

```
int add(int a, int b) { return (a+b); }
```

   만 남겨 놓고 나머지는 모두 지워라.

   주의: add.c에서 #include들도 모두 지우고, 위 한 줄만 남기고 파일 처음부터 끝까지 모두 삭제하라.

9-1. 위와 같은 방법으로 test1.c을 sub.c, mul.c, dvd.c라는 파일로 따로따로 복사하고, sub.c에는 sub() 함수, mul.c에는 mul() 함수, dvd.c에는 dvd() 함수의 정의만 남겨 놓고, 모든 include 문장을 포함하여 파일 처음부터 끝까지 나머지는 모두 지워라.

9-2. test1.c를 t1.c 복사한 후 vi로 t1.c로 들어 간다. t1.c에서는 add() 함수 정의를 외부(extern)함수 선언으로 수정하여라. 즉, **int add(int a, int b) { return (a+b); }** 를 **extern int add(int, int);** 로 수정하라. 나머지 세

함수도 아래와 같이 수정하라.

```
extern int sub(int, int);
extern int mul(int, int);
extern double dvd(int, int);
```

경고: 위의 각 함수는 함수 선언이므로 함수 이름 뒤에 함수의 정의, 즉 { return ...; } 등이 없어야 한다.

위 내용은 함수들이 t1.c에는 정의되어 있지 않고 다른 소스 파일에 정의되었음을 컴파일러에게 알려 주는 역할을 한다. 이상을 정리해 보면, 계산을 하는 네 개의 함수는 각각 다른 소스파일에 정의되어 있고, 이 함수들은 t1.c의 main() 함수에서 호출될 것이다.

10. 이제 add.c, sub.c, mul.c, dvd.c, t1.c를 이용하여 목적어 파일은 생성하지 말고 바로 t1이라는 실행파일을 만들어 보자. 명령어를 실행시키는 터미널 창에서 (강의노트 참조)

    $ gcc [실행 파일과 소스파일 이름들을 옵션과 함께 지정]

    // 에러가 발생했다면 **vi** 창에서 수정 및 **:w** 하고

    // 다시 명령 창에서 **$!g** 실행하라.

    $ ls (t1이 존재해야 함)

    $ t1 [enter] (t1 실행)

    (위 실습 4.2의 test1처럼 입력하여 테스트해 본다.)

11. 다음은 각 소스파일을 먼저 컴파일시키고, 그후 생성된 목적어 파일들을 따로 링크만하여 실행파일을 만들어 볼 예정이다. 먼저 명령어를 실행하는 터미널 창에서 add.c, sub.c, mul.c, dvd.c, t1.c 를 각 파일마다 따로 컴파일 하라. (강의노트 참조)

    $ gcc [add.c에 대해 목적어 파일 생성하는 옵션과 함께 지정]

    // 어떤 에러 메시지도 출력되지 않아야 정상임

    // 각각의 소스파일에 대해 위 과정을 반복하라.

    $ ls (add.o 존재해야 함)

    각 소스 파일에 대해 위 과정을 반복 한다.

12. 위에서 컴파일된 목적어 파일 add.o, sub.o, mul.o, dvd.o, t1.o을 이용하여 실행파일 **t2**을 생성하라. 실행파일 이름이 t1이 아니라 **t2**임을 명심할 것. 링크만 하는 것임(강의노트 참조).

    $ gcc [실행 파일과 목적어 파일이름들을 옵션과 함께 지정]

    // 어떤 에러 메시지도 출력되지 않아야 정상임

    $ ls (t2이 존재해야 함)

    $ t2 [enter] (t2 실행)

    위 실습 4. 2)의 test1처럼 입력하여 테스트해 본다.

13. 이번엔 add.c 파일에서 add() 함수 정의를

    `int add(int a, int b) { return (a*2+b*2)/2; }`

    로 수정한 후 저장하라. 명령어 창에서 11번처럼 add.c만 컴파일 하라. (다른 파일은 컴파일 하지 마라.) 이 경우 **$ history** 를 먼저 실행한다. 그런 후 11번에서 add.c를 컴파일한 명령어 번호를 찾는다. 번호가 120이라 가정하자. 그러면 **$!120 [enter]**하면 재 실행될 것이다. 그 후 실습 12번을 재 실행하라. (이것도 history로 명령어 번호를 찾아 재 실행하라.) 이는 실습 10번처럼 하나의 소스 파일이 수정되어도 매번 모든 소스 파일을 컴파일 하는 것이 아니라, 수정된 소스 파일만 다시 컴파일 하고 링크만 다시 해 주면 된다는 것을 확인한 것이다.

14. t1과 t2를 삭제한 후 다시 만들어라.

    $ rm t1 t2 (t1, t2 삭제) **$ ls** (삭제 확인)

    이제 t1을 다시 생성하기 위해 실습 10번을 재 실행하라. 모든 소스 파일을 새로 컴파일 한다. 소스 파일 길이가 길면 시간도 많이 걸린다. 그러나 t2를 생성하기 위해선 컴파일 할 필요 없이 실습 12(링크)만을 재 실행하라.

15. 다시 vi 에디터 창에서 test1.c 내에서 C언어에서 연산자 %와 동일한 기능을 수행하는 10 % 3(10을 3으로 나눈 나머지 값)과 같은 수식을 처리하는 기능을 삽입해 보자.

    1) 먼저 10 % 3을 처리하는 함수인 mod()를 dvd() 함수 정의 뒤에 추가하라. (mul() 함수와 형태가 유사함)
    2) 그리고 이 함수를 호출하는 코드를 test1.c에 삽입하라. 즉, main() 함수 내에서 else 문장 앞에 else if 를 새로 추가하여 mul() 함수 호출하는 것과 같은 방식으로 mod() 함수를 호출하는 코드를 삽입하라.
    3) 저장한 후 다른 터미널 창에서 실습 4번처럼 실행파일 test1을 새로 생성하라.
    4) 생성된 실행파일 test1을 아래처럼 실행시켜 본다.

       $ test1 [enter]
       Expression: 7 % 3[enter]
       // 결과 값은 1이어야 함

16. 이상의 모든 실습을 정상적으로 수행했다면 test1.o, test1, test2, t1.o, add.o, sub.o, mul.o, dvd.o, t1, t2라는 파일이 존재해야 한다. (물론 소스 파일도 존재함. t2와 test2의 소스와 목적어 파일은 없다.)

    1) 다음의 프로그램을 실행하여 정상적으로 실습했는지 확인하기 바란다. (아래처럼 입력하면 된다.)

       $ fshw 3
       $ eshw 3 // 에러만 체크

2) 또한 다음 프로그램을 실행하여 test1, test2, t1, t2 프로그램이 정상적으로 실행되는지 확인하기 바란다. 이 프로그램은 여러 분이 작성한 test1 실행 프로그램을 대신 실행하고, 다양한 입력을 대신 입력하여 test1이 여러 입력을 정상적으로 처리하는지를 테스트하는 프로그램이다. (메시지를 확인하고 enter만 계속 쳐주면 된다.)

```
$ progtest 3
```

3) 이상의 세 프로그램은 앞으로 항상 실습이 끝나고 나면 반드시 실행하여 소스파일과 실행파일이 정상적으로 작동하는지 확인하기 바란다. 교수님도 이 프로그램으로 학생들 전체를 확인한다.

4) 어떤 프로그램 또는 명령어든 실행 중에 끝내려고 한다면 (위 fshw나 progtest도 마찬가지) 무조건 **Ctrl+C** 를 누르면 종료된다.

```
// add.c: 함수 정의
int add(int x, int y)
{ return x + y; }
```

```
// mul.c: 함수 정의
int mul(int x, int y)
{ return x * y; }
```

```
// sub.c: 함수 정의
int sub(int x, int y)
{ return x - y; }
```

```
// dvd.c: 함수 정의
double dvd(int x, int y)
{return (double)x/y;}
```

```
// t1.c
extern int add(int x,int y);
extern int sub(int x,int y);
extern int mul(int x,int y);
extern double dvd(int x,int y);

int main()
{  ... }
```

gcc 사용법 정리 (4 가지 경우)