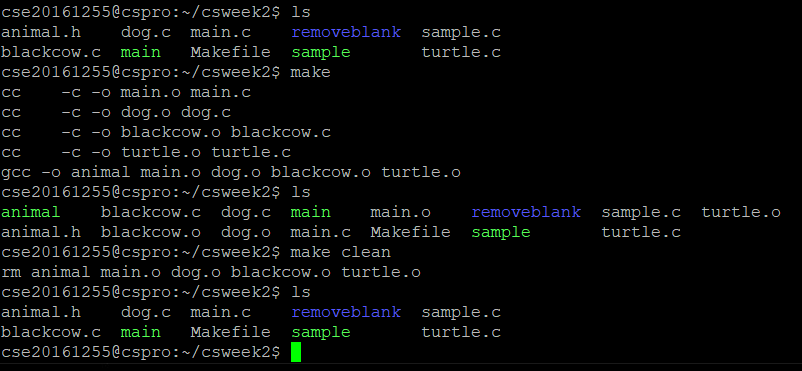
**실험 UNIX-2 결과보고서**

전공: 수학과 학년: 4 학번: 20161255 이름: 장원태

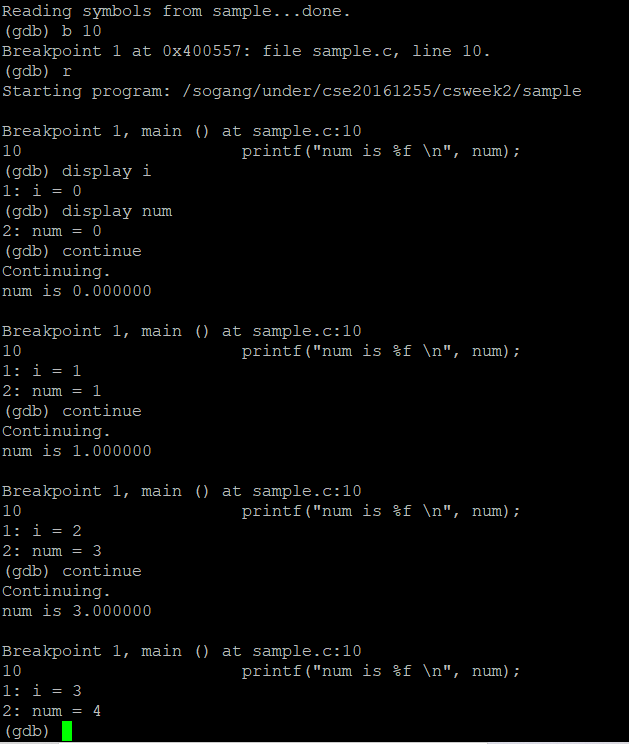
1. **목적**  
   실습 과정에 개발한 fmt에 대하여 결과 보고한다.
2. **문제 풀이 결과**

2-1. 실습 결과 화면을 첨부한다.

첫째로, 실습 1번인 Makefile 간단히 만들기의 경우, makefile을 통해 .o 파일을 만들고, ‘.PHONY clean’을 이용하여 make clean이라는 명령어를 만들었다. 그 결과, 다음과 같이 나타났다.

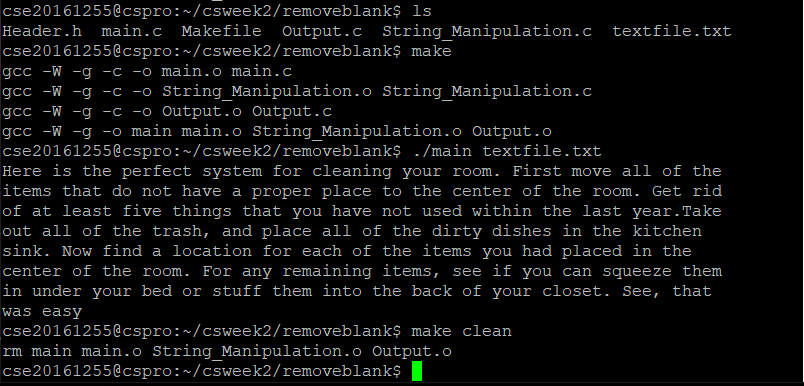


둘째로, 실습 2번인 gdb를 이용하여 코드의 문제 찾기의 경우, 우선 sample.c에 대해 컴파일을 진행하여 sample 파일을 만든 뒤, gdb를 이용하여 디버깅을 진행했다. 결과는 다음과 같이 나타났다. (i=3일 때까지만 진행하여 문제를 파악했다.)



이 경우를 보면, 분명 num에 i/2 + i를 대입해주었으나, i=1인 경우 num이 3/2가 아닌 1이 되었고, i=3인 경우 num이 9/2가 아닌 4가 되었다. 이는 num은 double형 변수이지만, i는 int형 변수이므로, i/2 또한 int형 변수가 되기 때문에, i가 홀수인 경우 i/2는 소수점 이하를 버리고 저장하게 된다. 이 때문에 오류가 발생한 것이므로, 해결하기 위해서는 형변환이 추가로 필요하다.

마지막으로, remove\_blank\_at\_the\_end 함수를 작성하여, 각 줄의 끝부분에서 다음 줄로 넘어갈 때, 그 끝에 있는 공백을 제거하도록 프로그램을 작성했다. 이 함수를 작성한 뒤, main 프로그램을 돌린 결과는 다음과 같이 나타났다.



2-2. remove\_blanks\_at\_the\_end 함수를 어떻게 구현하였는지 자세히 설명하시오. (코드 복사 붙여넣기 및 주석 작성)

#include "Header.h"

void Remove\_Blanks\_At\_The\_End( char \*line ) {

int i, k, newline\_flag = 0;

// 전체 문장에 대하여 line[k] 가 줄바꿈이면 flag를 1로 set, '\0'이면 break

for ( k = 0; ; k++ ) { // 문장의 앞부터 한 글자씩 스캔

if (line[k] == '\n') // 줄바꿈을 발견하면, flag를 1로 set

newline\_flag = 1;

else if (line[k] == '\0') // 문장의 끝을 발견하면, for문을 벗어남

break; // 이 때 k는 전체 문장의 글자수(줄바꿈 포함)가 됨

}

// 전체 문장에 대하여 line[i]가 space가 아니라면 break

for ( i = k-1; i >= 0; i-- ) { // 전체 문장의 끝에서부터, 줄바꿈이 아닐 때까지 스캔함을 통해 줄바꿈이 여러번 일어나지 않게끔 설정함

if (line[i] != '\n') // 줄바꿈이 아닌 다른 문자가 발견된 경우, 줄바꿈을 모두 통과한 것이므로 for문을 벗어남

break; // 이 때 i는 줄바꿈을 제외한 마지막 문자를 가리키는 인덱스가 됨

}

i++; // 마지막 문자의 바로 뒷부분을 가리킴

// flag가 1일때 문장의 마지막은 줄바꿈&space, flag가 1이 아니라면 '\0'

if ( newline\_flag == 1 ) { // 줄바꿈이 필요한 부분이라 flag가 1로 set이 되어 있다면, 마지막 문자의 바로 뒷부분에 줄바꿈을 수행해줌

line[i] = '\n';

}

else { // 줄바꿈이 필요하지 않은 부분이라면, 문장의 끝이 되므로 마지막 문자의 바로 뒷부분에 '\0'을 추가해줌으로써 문장을 끝냄

line[i] = '\0';

}

}

2-3. 실습시간에 작성한 Makefile의 한줄 한줄의 의미를 설명한다.

실습시간에 작성한 Makefile은 다음과 같다.

|  |
| --- |
| cc = gcc  target = animal  objects = main.o dog.o blackcow.o turtle.o  $(target): $(objects)  $(cc) -o $(target) $(objects)  $(objects) : animal.h  .PHONY : clean  clean :  rm $(target) $(objects) |

우선, 최상단 부분은 매크로 설정 부분이다. 첫째 줄의 ‘cc = gcc’의 경우, ‘gcc를 이용하여 컴파일을 진행하겠다’라는 의미이다. 둘째 줄의 ‘target = animal’의 경우, 이 Makefile의 최종 목표가 ‘animal’이라는 뜻으로, ‘make’ 명령을 수행시킬 경우 animal이라는 파일이 생성된다. 그리고, 이 ‘make’ 명령에서 사용되는 오브젝트 파일들의 모음을 나열한 것이 셋째 줄의 ‘objects = main.o dog.o blackcow.o turtle.o’이다.

다음으로, $(target): $(objects)의 경우, target이 objects를 요구한다는 의미이다. 이 예시의 경우, target인 animal이 objects인 main.o, dog.o, blackcow.o, 그리고 turtle.o를 요구한다는, 혹은 의존한다는 의미이다. 그리고, 이어지는 $(cc) –o $(target) $(objects)의 경우는 그에 따른 법칙이다. 이 예시의 경우, gcc –o animal main.o dog.o blackcow.o turtle.o 라는 명령을 압축한 형태이다. 즉, gcc 컴파일러를 이용하여, 네 오브젝트 파일들을 통해 animal이라는 파일을 만들라는 명령이다. 마찬가지로, $(objects) : animal.h 의 부분 또한 objects인 네 오브젝트 파일들이 animal.h을 요구한다는 의미이다.

마지막으로, .PHONY : clean 의 경우는 ‘make clean’이라는 명령어를 추가시킨 것이다. PHONY를 이용하는 이유는, 파일 중에 ‘clean’이라는 이름을 가진 파일이 있는 경우 의도치 않게 파일이 만들어질 수 있기 때문에, 따로 명령어로 추가하기 위함이다. 이 경우, clean이라는 명령어를 사용하면 rm $(target) $(objects) 를 수행하라고 되어 있다. 즉, clean을 사용하면 target과 object들을 모두 삭제하는 것이다. 이 예시의 경우, animal과 네 오브젝트 파일들을 모두 삭제하게 된다.

2-4. make의 옵션들에 대하여 정리한다.

Make의 옵션들은 다음과 같다.

|  |  |
| --- | --- |
| -b, -m | Online help와 exit를 출력한다. |
| -B, --always-make | 무조건 모든 target을 make한다. |
| -C dir, --directory=dir | Makefile을 읽거나 다른 무언가를 하기 전에 dir 디렉토리로 바꾼다. |
| -d | 일반적인 프로세싱에 더하여 디버깅 정보를 출력한다. |
| --debug[=FLAGS] | 일반적인 프로세싱에 대하여 디버깅 정보를 출력한다. FLAGS에 따라 해당 디버깅 정보가 달라진다. |
| -e, --environment-overrides | makefiles로부터의 변수보다 상위의 환경에서 가져온 변수들을 준다. |
| +-f file, --file=file, --makefile=FILE | file을 makefile로 사용한다. |
| -i, --ignore-errors | 파일을 재생성하는 데 수행된 명령에 있는 모든 에러를 무시한다. |
| -I dir, --include-dir=dir | 포함된 makefiles를 검색하기 위해 dir 디렉토리를 특정화한다. |
| -j [jobs], --jobs[=jobs] | 동시에 run된 jobs(명령들)의 수를 특정화한다. |
| -k, --keep-going | 하나의 에러 이후에 최대한 많이 이어서 진행한다. |
| -l [load], --load-average[=load] | 다른 job들이 가동되고 있고 load average가 최소의 load(floating-point number)에 있으면, 새로운 job(명령)이 시작되어선 안 된다는 것을 특정화한다. |
| -L, --check-symlink-times | Symlinks와 target 사이의 최근의 mtime을 사용한다. |
| -n,, --just-print, --dry-run, --recon | 수행될 명령을 출력한다. 단, 그것들을 수행하지는 않는다. |
| -o file, --old- file=file, --assume-old=file | 파일 file이 그 의존성보다 오래되었다면 새로 만들지 않으며, file에서의 변화에 따라 무언가를 재생성하지 않는다. |
| -p, --print-data-base | makefile들을 읽음을 통해 생성된 데이터베이스(법칙과 변수값)를 출력하고, 기존과 같이, 혹은 특정화된 다른 방식으로 수행한다. |
| -q, --question | “Question mode”에 들어간다. 즉, 아무 명령도 수행하지 않고 아무 것도 출력하지 않고, exit status, 즉 특정화된 target이 이미 최신 상태이면 0이고 그렇지 않으면 1인 값을 반환한다. |
| -r, --no-builtin-rules | 내장된 절대적 법칙의 사용을 제거하고, 접미사 법칙에 대한 접미사의 디폴트 리스트를 모두 삭제한다. |
| -R, --no-builtin-variables | 어떤 내장 변수도 정의하지 않는다. |
| -s, --silent, --quiet | 수행된 명령을 출력하지 않는다. |
| -S, --no-keep-going, --stop | -k 옵션의 효과를 취소한다. |
| -t, --touch | 명령을 수행하지 않고 대신 파일을 touch(실제로 바꾸지 않고 최신 상태로 mark)한다. |
| -v, --version | Make 프로그램의 버전과 저작권, 저자의 리스트, 그리고 보증이 없다는 공지를 출력한다. |
| -w, --print-directory | 다른 프로세싱 이전과 이후에 작업중인 디렉토리를 포함하는 메시지를 출력한다. |
| --no-print-directory | -w를 끈다. |
| -W file, --what-if=file, --new- file=file, --assume-new=file | Target 파일이 방금 변경되었다고 가정한다. |
| --warn-undefined-variables | 정의되지 않은 변수가 참조된 것에 대해 경고한다. |

출처: <https://www.tutorialspoint.com/unix_commands/make.htm>