2020년도 2학기 컴퓨터공학설계및실험I

2주차 결과 보고서

20170175 김태안

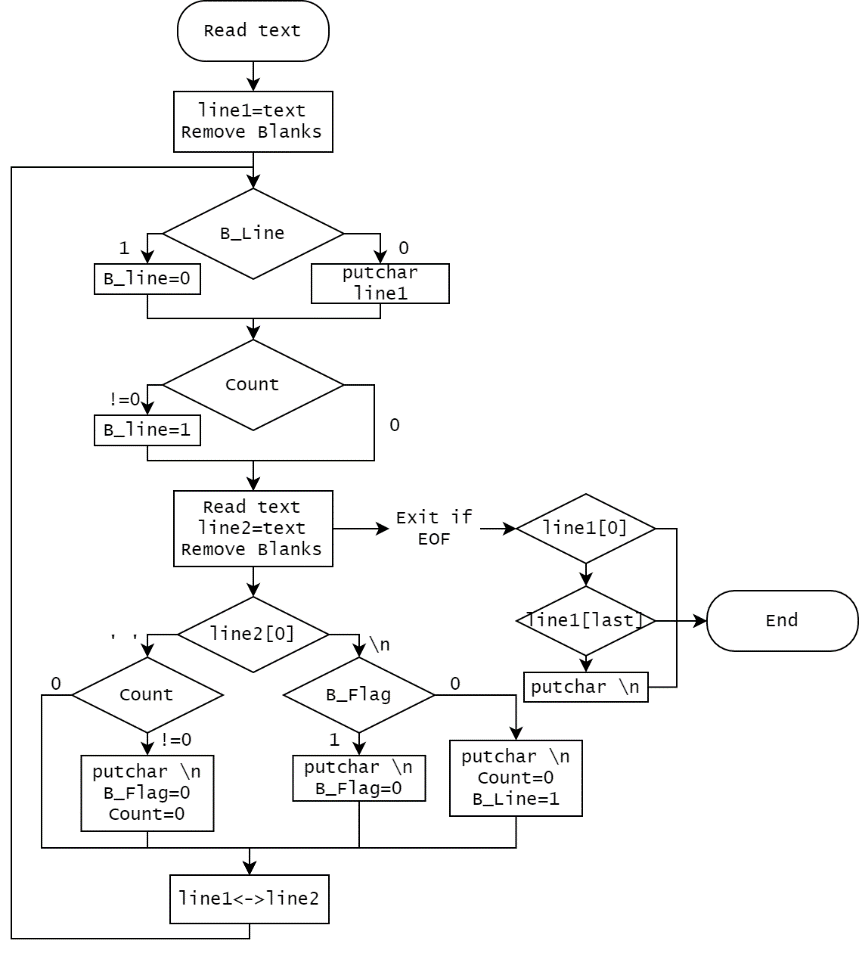
1. 실습 목적

실습 과정에 개발한 fmt에 대하여 결과를 보고한다.

1. 실습 구현 내용

**fmt 프로그램**

fmt는 텍스트를 읽어 줄 당 글자 제한에 맞춰 텍스트를 재정렬하는 프로그램이다. 예를 들어, 줄 당 제한이 30자이고, 읽은 텍스트가 50자라면, 30자 이후에 줄바꿈을 해 글자 제한을 넘어가는 경우를 없앤다.



**그림 1.** fmt의 FlowChart

fmt 프로그램의 흐름은 **[그림 1]**과 같다. 텍스트를 한 줄 씩 읽어 공백문자를 지우고, 공백인지 확인한다. 공백이면 출력을 하지 않고 넘어가고, 공백이 아니면 출력한다. Count에 출력할 글자수를 저장하고, 더 입력할 수 있는 지 확인한다. 그리고 다음 줄을 읽어 flag에 따라 줄바꿈과 flag를 조정한 뒤, 반복한다. 파일의 끝을 읽으면 마지막 줄의 상태를 확인하고 출력한 뒤, 프로그램을 종료한다.

실습을 위해 미리 구현되어 있는 fmt 코드는 아래의 함수로 구성되어 있다.

1. **int main (void)**

main 함수. 텍스트 파일을 입력 받고 **[그림1]**대로 함수를 호출하며 알고리즘을 실행한다.

Input: 텍스트 파일명

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

읽은 텍스트의 공백을 제거하는 함수. 한 글자씩 읽어 \n나 \0을 확인하고 마지막 문자와 \n, \0 사이의 공백을 제거한다.

Input: 읽은 텍스트 line

Output: 공백을 제거한 텍스트 (포인터)

1. **void Get\_Blanks\_Chars (char \*line, int Start, int \*N\_Blanks, int \*N\_Chars)**

N\_Blanks, N\_Chars를 구하는 함수. line을 Start부터 한 글자씩 읽어 공백문자를 읽으면 N\_Blanks를, 아니면 N\_Chars를 증가시켜 개수를 센다.

Input: Start, Line

Output: N\_Blanks, N\_Chars (포인터)

1. **void Print\_Line (char \*line, int \*Count, int \*B\_Flag)**

텍스트를 출력하는 함수. B\_Flag 조건에 따라 line을 한 자씩 읽어 Start부터 조건에 따른 개수만큼 출력한다.

Input: Line, Count, B\_Flag

Output: 텍스트 출력 (Standard Input)

**Makefile**

Unix에서 fmt 프로그램을 컴파일하기 위해서는 gcc/g++을 사용해야한다. 컴파일해야 하는 파일은 Header.h, main.cpp, Output.cpp, String\_Manipulation.cpp의 총 4개이다. 이때, Makefile을 사용하면 여러 개의 소스코드를 쉽게 컴파일할 수 있다. 구현한 Makefile은 다음과 같다.

|  |
| --- |
| CC=g++  target=fmt  objects=main.o Output.o String\_Manipulation.o  CPPFLAGS=-g -c  $(target): $(objects)  $(CC) -o $(target) $(objects)  $(objects): Header.h  .PHONY: clean  clean:  rm -f $(objects) $(target) |

**그림 2.** Makefile의 소스코드

CC는 사용할 컴파일러를 지정한다. target은 출력되는 파일의 이름이다. objects는 생성할 오브젝트 파일로, object를 설정하면 자동으로 c/cpp 파일을 찾는다. CPPFLAGS의 -g는 gdb를 통해 디버그할 수 있게 해준다. 이후의 매크로는 정의한 모든 요소들에 대해 매크로 명령을 실행한다. 위 경우, g++ -o fmt (오브젝트 파일)을 실행한 것과 같은 효과이다. clean은 clean을 입력할 시 실행할 명령어를 미리 지정하고, PHONY는 clean을 입력 받으면 무조건 아래 명령을 실행하도록 설정한다.

1. 실습 환경

cspro.sogang.ac.kr

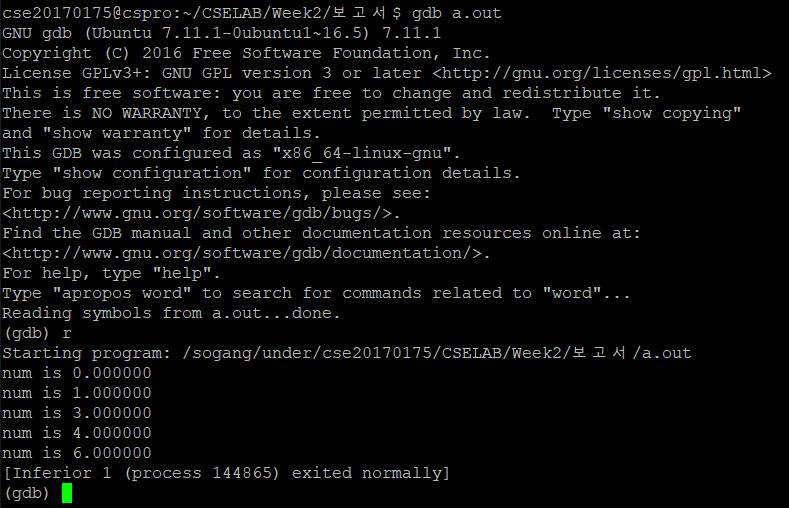
OS: Ubuntu 16.04.2 LTS (GNU/Linux 4/4/0-184-generic x86\_64)

1. 실습 결과 및 분석

**GDB를 이용한 코드의 디버그**

gcc를 이용해 컴파일할 때, -g 옵션을 사용하면 GDB라는 디버그 툴을 사용할 수 있다. 아래 코드와 같은 코드를 생성하고 디버깅해보았다.

|  |
| --- |
| #include <stdio.h>  main(void)  {  int i;  double num;    for (i=0; i<5; i++) {  num=i/2+i;  printf(“num is %f \n”, num);  }  } |



**그림 2.** GDB 실행 결과

프로그램 자체는 작동에 아무 이상이 없다. 다만, main의 return 형식이 지정되지 않아 경고 메시지를 보낸다. num을 double로 두고, num의 값이 i/2+i임을 고려하면 프로그램의 의도는 i/2 + i를 소수까지 표현하여야 하는 것 같다. 그러나 s를 통해 한 줄씩 실행하면, num에는 예상 의도와 다른 정수가 입력된다. 이는 정수형을 나눴을 때 정수 결과를 반환하기 때문이고, i 앞에 (double)을 붙여 해결할 수 있다.

**fmt 프로그램 디버그**

실습을 위해 주어진 fmt 프로그램은 문제가 있다. 프로그램을 돌리면 Segment Fault를 보이며 에러가 발생한다. Segment Fault는 주로 잘못된 배열 참조나 접근할 수 없는 주소에 접근하려 할 때 발생한다. 따라서 프로그램을 디버그하며 문제를 수정해야한다. gdb를 사용해 찾은 에러는 다음과 같다.

|  |  |  |
| --- | --- | --- |
| **int main()** | | |
| int i, B\_Flag, B\_Line; | int i, B\_Flag, B\_Line;  B\_Flag=0;  B\_Line=0; | B\_Flag와 B\_Line을 초기화한다. |
| **void Remove\_At\_The\_End( char \*line )** | | |
| for ( k = 0; ; k++ ) { | for ( i = k-1; i >= 0; i-- ) { | i는 k-1부터이다. |
| if ( newline\_flag == 1 ) {  line[i] = '\n';  line[i+1] = '\0';  }  else {  line[i] = '\0';  } | if ( newline\_flag == 1 ) {  line[i+1] = '\n';  line[i+2] = '\0';  }  else {  line[i+1] = '\0';  } | 공백의 입력 위치가 하나씩 밀려있다. |
| **void Get\_Blanks\_Chars( char \*line, int Start, int \*N\_Blanks, int \*N\_Chars )** | | |
| int i, blank\_flag; | int i, blank\_flag;  \*N\_Blanks=0;  \*N\_Chars=0;  blank\_flag=0; | N\_Blanks와 N\_Chars, blank\_flag가 초기화되지 않았다. |
| void Print\_Line( char \*line, int \*Count, int \*B\_Flag ) | | |
| int i, N\_Blanks, N\_Chars, Start;  for ( i = Start; i < Start + N\_Blanks + N\_Chars - 1; i++ ) { | int i, N\_Blanks, N\_Chars, Start;  N\_Blanks=0;  N\_Chars=0;  Start=0; | 변수를 초기화해야한다. |
| for ( i = Start; i < Start + N\_Chars - 1; i++ ) { | for ( i = Start; i < Start + N\_Chars; i++ ) { | for문의 끝이 잘못 설정되어 있다. |
| for ( i = Start; i < Start + N\_Blanks + N\_Chars - 1; i++ ) { | for ( i = Start; i < Start + N\_Blanks + N\_Chars; i++ ) { |
| for ( i = Start; i < Start + N\_Blanks + N\_Chars - 1; i++ ) { | for ( i = Start; i < Start + N\_Blanks + N\_Chars; i++ ) { |
| for ( i = Start + N\_Blanks; i < Start + N\_Blanks + N\_Chars - 1; i++ ) { | for ( i = Start + N\_Blanks; i < Start + N\_Blanks + N\_Chars ; i++ ) { |

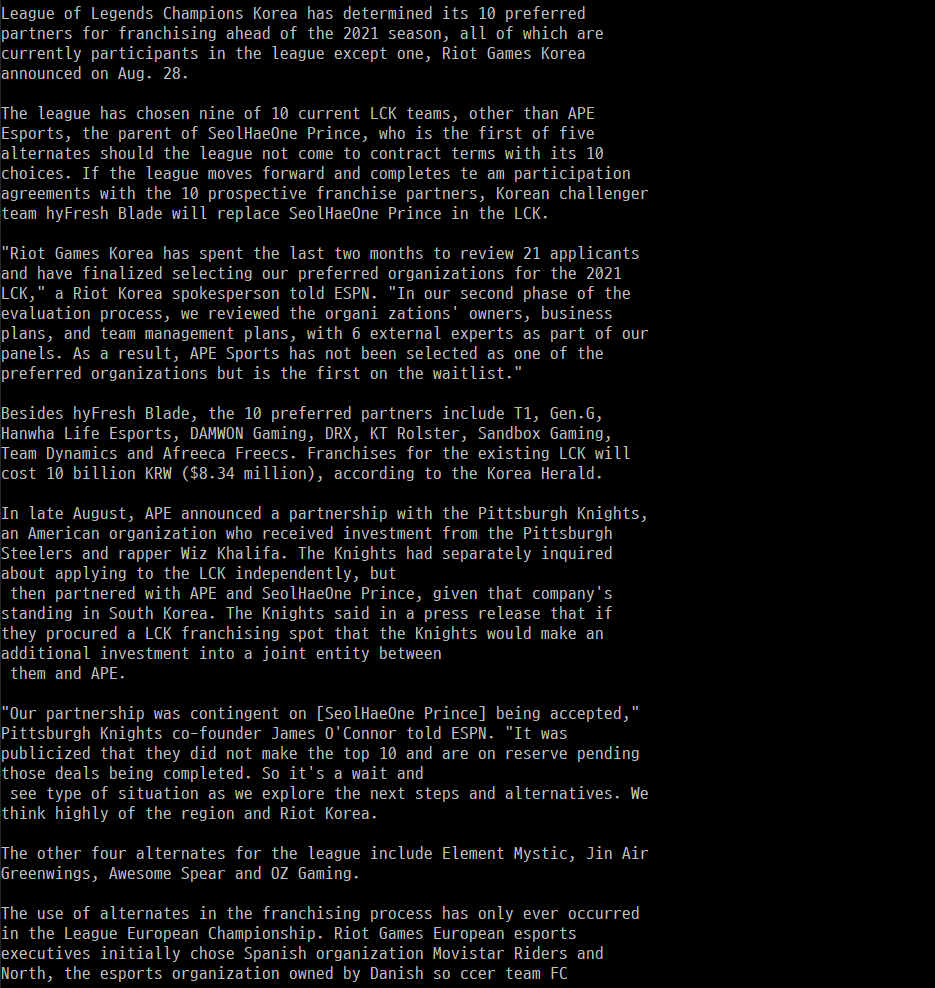
**테스트 데이터와 그 결과**

릇에리그오브레전드 e-sports에 관한 기사에 관한 파일 ESPN.txt를 입력하면 다음과 같은 결과를 출력한다.



**그림3.** fmt 프로그램의 실행 결과 (g++)

런타임 에러가 발생하진 않지만 일부 텍스트에서 올바르지 못한 출력을 보인다. 흥미로운 점은 같은 프로그램을 g++가 아닌 MSVC (마이크로소프트의 C++) 컴파일러를 사용하면 아래와 같이 출력한다.



**그림4.** fmt 프로그램의 실행 결과 (MSVC)

잘리던 텍스트가 그대로 보이며, 에러가 없다.

1. 과제

**For Star Wars-Episode VII**

N까지의 페이지가 있다고 가정할 때, 각 장마다 적혀 있는 페이지를 이루는 숫자를 구해야 한다. 프로그램의 알고리즘은 단순하다. 1부터 N까지의 숫자의 구성 숫자를 구한 뒤 기록하고, 마지막에 출력하면 된다. 이를 위해 필요한 함수는 다음과 같다.

1. **int main()**

메인 함수. 처음엔 입력할 테스트의 개수를 입력 받고 테스트 개수만큼 NCounter를 실행한다.

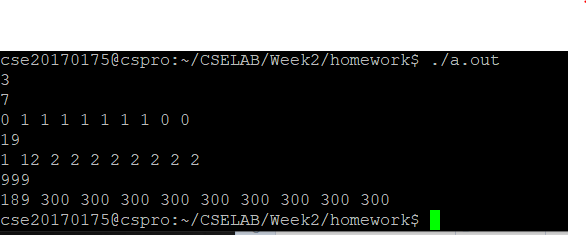
1. **int\* NCounter()**

숫자의 개수를 세는 함수. N을 입력받아 1부터 N까지 숫자를 센다. 숫자는 Numbers 배열에 저장되는데, 배열의 길이가 10이므로 배열의 i번째칸에 숫자 i의 개수를 저장할 수 있다. 1부터 세기 시작한 숫자 n은 나눈 몫이 0이 될 때까지 10으로 나누어 각 자리 수를 구하고, 그 결과를 Numbers에 저장한다. 계산이 모두 끝난 경우, Numbers를 ResultPrinter에 넘겨준다.

1. **void ResultPrinter(int\* Numbers)**

출력함수. NCounter에서 넘겨받은 배열을 한 칸 씩 읽어 출력한다.

실행결과는 다음과 같다.



**그림5.** For Star Wars-Episode VII의 실행 결과

1. 결론

이번 주차의 실습에서는 gcc의 사용법, Makefile, 그리고 GDB에 대해서 배웠다. gcc는 유닉스의 C언어 컴파일러이다. gcc 명령어를 통해 작성한 c파일을 컴파일해 실행 파일을 생성하고, 실행파일을 실행해 프로그램을 작동시킬 수 있다. 컴파일러에 따라 프로그램의 결과가 달라지므로 어떤 컴파일러에서는 정상 작동하는 프로그램이 다른 컴파일러에서는 다르게 실행될 수 있다.

Makefile은 여러 파일을 컴파일할 때 도움을 주는 기법이다. 미리 컴파일할 파일과 옵션을 지정하고 매크로를 이용해 파일을 컴파일 하는 방식이다. 기본적으로 cc는 컴파일러, target은 실행파일명, objects는 오브젝트 파일을 의미한다.

GDB는 gcc 컴파일러를 사용할 때 사용할 수 있는 디버깅 툴이다. gcc에 -g 옵션을 주어 디버깅 파일을 만들면 GDB에서 실행해 프로그램의 동작과정을 순서대로 볼 수 있다. 이를 통해 에러를 쉽게 발견할 수 있다.