Programming Assignment #1:

Converting text file formats

Due: 1th Oct. (Sun), 11:59 PM

1. Introduction

이번 과제의 목표는, 텍스트 파일의 개행 문자와 공백 문자를 변환하거나 정규화시키는 프로그램을 제작하는 것이다. 이 과제의 주 목표는 리눅스 환경에서 파일 입출력을 수행하는 것이다. 또한, 파일의 개행 문자에 대해 배울 수 있다.

2. Problem specification

파일 `convert.c` 에 포함된 `convert()` 함수를 구현한다. `convert()` 함수의 원형은 다음과 같다:

```
int convert (const char *input);
```

첫번째 인자 `input`은 처리할 텍스트 파일의 이름이다. `input` 파일을 열고 텍스트 파일의 포맷을 판별한 뒤, [문자열 `input + ".out"`] 의 이름을 가진 파일을 생성하여 새로운 포맷으로 변환하여 저장한다. `input`이 `"hello.txt"`라면 새로 변환될 파일은 `"hello.txt.out"` 이 될 것이다. 2.2절에 제시된 규칙에 따라 텍스트 파일을 처리하여 새로운 포맷의 파일로 저장한 다음, 2.3절에서 지정한 대로 처리 결과 메시지를 출력하고, 2.4절에 해당하는 결과를 반환한다.

2.0. Abbreviations

<LF>	'\n'	Line feed character	0x0A
<CR>	'\r'	Carriage return character	0x0D
<TAB>	'\t'	Horizontal tab character	0x09
<space>	' '	Space character	0x20

2.1. Definition of Unix and Dos formats in this assignment

	Unix	Dos
Newline	<LF>	<CR><LF>
<TAB> char.	8 consecutive <space> → <TAB>	<TAB> → 4 consecutive <space>

2.2. Conversion rules

2.2.1. File format

- 입력 파일의 개행 문자를 확인했을 때 Unix 포맷인 경우, 파일을 Dos 포맷으로 변환한다.
- 입력 파일의 개행 문자를 확인했을 때 Dos 포맷인 경우, 파일을 Unix 포맷으로 변환한다.
- Unix와 Dos 포맷이 섞인 파일의 경우, 대세를 이루는 쪽으로 변환한다. Unix 포맷과 Dos 포맷의 개행 문자의 개수가 동일하다면, 해당 파일을 Unix 포맷으로 변환한다.
- 바꿀 포맷에 맞춰 개행 문자를 바꾼다.

2.2.2. <TAB> character

- 파일을 Unix 포맷으로 바꿀 경우, 원본 파일이 프로그램 소스 코드인 경우에 한해 8개의 <space> 문자를 <TAB> 문자 하나로 변환한다.
  - 프로그램 소스 코드는 원 파일의 확장자가 `.c`, `.h`, `.java` 인 경우이다.
- 파일을 Dos 포맷으로 바꿀 경우, 모든 파일 종류에 대해 <TAB> 문자 하나를 4개의 <space> 문자로 변환한다. 즉, Dos 포맷으로 바뀐 파일엔 <TAB> 문자가 존재할 수 없다.

2.2.3. Trimming whitespaces (Optional)

- 파일 내의 불필요한 공백 문자들을 제거한다.
- 불필요한 공백 문자란 한 line에서, 개행 문자 직전에 있는 공백 문자의 집합을 뜻한다.
- 여기서 공백 문자는 <space>, <TAB> 두 문자만 고려한다.
- 본 내용은 구현하지 않아도 어떤 불이익도 없다.
- **본 내용을 정확히 구현하면, 코드 점수의 20%에 해당하는 보너스 점수를 받는다.**

2.3. Result message

- 에러가 발생한 경우, 표준 에러(`stderr`)에 "`Error occurred!\n`" 란 문자열을 출력한다. '`\n`'은 개행 문자이다.
  (이 개행 문자를 복잡하게 생각하지 말 것. 과제의 대상이 아님!)
- 파일이 정상적으로 처리된 경우, 표준 출력(`stdout`)에 다음과 같은 문자열을 출력한다. (개행 문자가 존재함을 확인할 것. 여기서 ⏎는 개행 문자를 뜻함.)

```
Input:<INPUT_FILE_NAME> Output:<OUTPUT_FILE_NAME> Size:<FILE_LENGTH>⏎

/* printf format */
printf("Input:%s Output:%s Size:%lld\n", input, output, length);

/* A possible output */
Input:hello.txt Output:hello.txt.out Size:192⏎
```

2.4. Returns and error handling

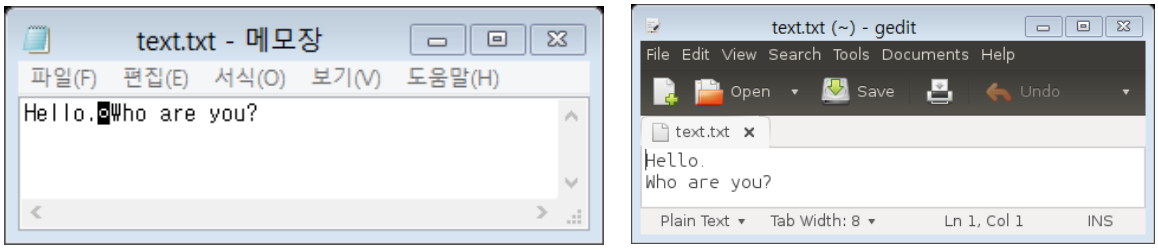
프로그램 수행 중, 다음과 같은 상황이 발생하면 에러이다. 에러가 발생하면, 각 해당 에러에 해당하는 값을 반환한 뒤 종료한다. 각 에러는 파일 `convert.h`에 선언되었다. 프로그램이 정상적으로 수행된 경우, 0을 반환한다.

- 입력 파일이 존재하지 않음 → `(E_INPUT_NOT_EXIST)`
- 입력 파일을 이용할 권한이 없음 → `(E_INPUT_NOT_ALLOWED)`
- 출력 파일을 생성할 수 없음
  - 같은 이름을 가진 파일이 이미 존재함 → `(E_OUTPUT_EXIST)`
  - 해당 위치에 파일을 생성할 권한이 없음 → `(E_OUTPUT_NOT_ALLOWED)`

3. Background

3.1. Newline

전통적인 이유로, 각 시스템은 독자적인 방식으로 개행 문자를 표현한다. Unix의 경우, <LF>만으로 개행 문자를 표현하고, Windows의 경우, <CR><LF>로 개행 문자를 표현한다. 따라서, 양쪽의 시스템에서 같은 내용을 작성해 저장해도, 두 파일을 비교하면 파일의 내용이 일치하지 않는다. 대부분의 텍스트 에디터는 이 차이를 자동으로 보정하지만, Windows의 메모장 같은 프로그램은 이를 수행하지 않아서, 다음 그림과 같이 Unix에서 생성한 파일이 제대로 표현되지 않는다.



[좌측은 같은 파일을 Windows 메모장으로 열었을 때, 우측은 Linux의 `gedit`으로 열었을 때]

파일을 바이트 단위로 확인하면 더 정확히 알 수 있다. `od`는 파일을 바이트 단위로, 특정 포맷으로 출력해주는 도구로 문자(char)나, 16진수(hex) 등으로 출력할 수 있다. `od -c` 명령어를 사용하면 해당 파일을 문자형으로 보여주는데, 같은 내용의 파일이라도 파일의 포맷에 따라 개행 문자에 해당하는 부분이 <LF>와 <CR><LF>로 다름을 확인할 수 있다. 아래 그림에서 보이는 것처럼, 같은 내용을 표현하는 두 파일의 파일 크기는 서로 다를 수 있다.

```
minsl@salmon:~$ od -c text.txt
0000000   H   e   l   l   o   .  \n   W   h   o       a   r   e       y
0000020   o   u   ?
0000023
minsl@salmon:~$ od -c text-win.txt
0000000   H   e   l   l   o   .  \r  \n   W   h   o       a   r   e
0000020   y   o   u   ?
0000024
```

프로그램을 구현한 후, 검증하기 위해 위의 도구를 사용하거나, 생성된 파일을 Windows 메모장에서 열었을 경우 정상적으로 표현되는지 확인하는 방법 등을 사용할 수 있다.
(참고: http://en.wikipedia.org/wiki/Newline)

3.2. Errno

시스템 콜 수행 중 에러가 발생한 경우, 반환 값은 -1이고, `errno`란 곳에 에러 번호가 저장된다. 이 `errno`를 사용하기 위해선 프로그램 내부에 `<errno.h>` 헤더 파일을 포함시킨다. 이 `errno`와 시스템 콜의 매뉴얼(man page)에 언급된 발생할 수 있는 에러를 비교하여 정확한 원인을 찾을 수 있다.
(참고: `$ man 3 errno`)

4. Skeleton codes

이번 과제 수행을 위해 다음 4개의 파일이 주어진다:

`Makefile:`	GNU `make` 도구를 위해 필요한 파일
`convert.h:`	에러 규격과 함수가 선언된 헤더 파일
`convert.c:`	제시된 `convert()` 함수를 구현할 `C` 파일
`main.c:`	쉬운 테스트를 위해 만들어진 `main()` 함수가 포함된 `C` 파일.

GNU `make` 도구를 이용하여 다음 명령을 호출함으로써 실행 파일을 만들 수 있다. 만들어진 실행 파일의 이름은 `convert`로, 다음과 같이 실행시킨다.

```
$ make
$ ./convert
```

GNU `make` 도구는 큰 프로그램을 만들 때 유용하게 사용되는데, 프로그램을 제작하기 위해 어떤 코드를 (재)컴파일해야 하는지 결정해준다. 일단 `Makefile`이 준비되면, 어떤 소스 코드를 변경하던지 셸에서 단순히 `make`란 명령을 실행시키는 것으로 재컴파일이 필요한 모든 파일을 알아서 찾아 다시 컴파일한다. 본 과제를 예로 들면, `convert.c` 혹은 `main.c`를 고친 다음 셸에서 `make`를 실행하는 것으로 실행 파일을 업데이트할 수 있다.

5. Restrictions

- 파일을 다루기 위해 리눅스 시스템 콜을 이용한다.
  - 수업에서 배우지 않은 시스템 콜을 쓸 수 있다. 이 경우엔 보고서에 왜 해당 시스템 콜을 사용했는지 설명한다.
- `malloc()`, `calloc()`, `free()` 함수를 제외한 다른 라이브러리 함수는 **사용할 수 없다**.

필요하다면, `convert.c` 안에 직접 구현하여 사용한다. [i.e., PA #0에서 구현한 함수]
    - 쉬운 디버깅을 하기 위해 라이브러리 함수를 사용할 수 있다. 다만, 제출한 파일엔 라이브러리 함수가 포함되지 않아야 한다. (주석 등으로 처리해도 괜찮음)
- 어떤 자원을 동적으로 할당 받았다면, 프로그램 종료 전에 반드시 해제해야 한다.
    - 여기서 자원이란 파일이나 메모리를 뜻한다.

6. Hand in instruction

- 작성한 코드 상단의 주석에 이름과 학번을 작성한다.
- `convert.c` 파일의 이름을 "`학번.c`" 로 바꾼다. (e.g., `2008311920.c`)
- 본 과제 수행 시 구현 방법과 디자인을 설명하는 보고서를 PDF 포맷으로 작성하여 "`학번.pdf`" 이란 이름을 붙인다. (가능하면 PDF가 가장 좋지만, 대중적인 문서 포맷은 다른 포맷도 괜찮음)
- 과제 제출은 아이캠퍼스로 한다. 제출 양식은 C 파일과 문서 파일을 "학번.(zip/tar.gz/...)" 으로 압축하여 제출한다.

7. Logistics

- 본 과제는 혼자 수행한다.
- 과제 제출 시간은 메일 도착 시간을 기준으로 하며, 과제를 지연 제출하면 기한 직후부터 매 8시간마다 점수를 10%씩 추가로 감점한다.
- 다른 사람의 과제를 copy할 경우, 개입한 사람 전부 해당 과제에 대해 0점 처리되고, 교수님께 보고되며, **성적 산정에 불이익이 있다**. 또한, copy가 두 차례 이상 적발될 경우 **F 학점이 부여될 수 있다**.

Have fun!

박종원, 담당 조교
임베디드소프트웨어연구실