시스템프로그래밍(가) 과제1

소프트웨어학부 20192800 권대현

1. 개발 환경
   - 운영체제: Windows 11 Home
   - 하위 시스템: GNU/Linux 5.15.146.1-microsoft-standard-WSL2 x86_64
   - 리눅스 버전: Ubuntu 22.04.2 LTS
2. 소스코드 설명
   - 20192800.h

```
void signed_char(char*, uint8_t);
void ASCII_codes(char*, uint8_t);
void unsigned_char(unsigned char*, uint8_t);
void signed_int(int*, uint32_t);
void unsigned_int(unsigned int*, uint32_t);
void signed_float(float*, int32_t);
void signed_double(double*, int64_t);
```

     - 20192800.h 파일에는 20192800.c에서 사용할 함수들에 대한 선언부가 저장되어 있다.
     - 구분하기 용이하도록 함수명은 저장할 자료형의 이름으로 작명하였다.
     - 파라미터는 2개로, 저장할 결과값 배열 포인터와 버퍼가 주어진다.
   - 20192800.c

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "20192800.h"

FILE *file;

//결과값을 각 배열에 저장하기 위한 인덱스 배열
int index_result[7] = { 0, };

int main()
{
    int bufsiz = 0;

    file = fopen("input", "r");

    //input의 버퍼 사이즈를 얻기 위한 while문
    while(!feof(file))
```

```
    {
        bufsiz++;
        fgetc(file);
    }

    //input 문자열을 담을 배열과 길이 선언
    int str_length = 0;
    char *str = (char*)malloc(bufsiz - 2);

    //파일 인덱스 초기화
    fseek(file, 0, SEEK_SET);

    //input 파일로부터 0과 1을 검사하고 str에 저장
    while(!feof(file))
    {
        char c = fgetc(file);
        if(c == '0' || c == '1')
            str[str_length++] = c;
    }

    //구분된 비트들을 각기 다른 자료형으로 출력하기 위한 결과값 배열 선언
    char *result_sc = (char*)malloc(str_length / 8);
    char *result_ac = (char*)malloc(str_length / 8);
    unsigned char *result_uc = (unsigned char*)malloc(str_length / 8);
    int *result_si = (int*)malloc(str_length / 32);
    unsigned int *result_ui = (unsigned int*)malloc(str_length / 32);
    float *result_sf = (float*)malloc(str_length / 32);
    double *result_sd = (double*)malloc(str_length / 64);

    //8비트(=1바이트)를 원소로 하는 2차원 문자 배열
    char **buffer;

    //2차원 배열 동적 할당
    buffer = (char**)malloc(sizeof(char*) * str_length / 8);
    for(int i = 0; i < str_length / 8; i++)
        buffer[i] = (char*)malloc(sizeof(char) * 8);

    //1바이트, 4바이트, 8바이트 버퍼 선언
    uint8_t buffer_onebyte;
    uint32_t buffer_fourbyte;
    uint64_t buffer_eightbyte;
    int buffer_index = 0;

    //0번 인덱스부터 input 문자열의 끝까지 탐색하여 8비트씩 잘라서 buffer 배열에
입력하는 for문
    for(int i = 0; i < str_length; i++)
    {
        if((i + 1) % 8 == 0)
```

```
        {
            int small_index = 0;
            for (int j = i - 7; j <= i; j++)
                buffer[buffer_index][small_index++] = str[j];
            buffer_index++;
        }
    }
    buffer_index--;

    //저장된 buffer 를 top 인덱스에서 0 번 인덱스로 스택처럼 뒤에서부터 탐색하는
for 문
    for(int i = buffer_index; i >= 0; i--)
    {
        //1 바이트일 때
        if((buffer_index - i + 1) % 1 == 0)
        {
            //비트로 표기된 문자열을 정수로 변환해주는 strtol
            buffer_onebyte = (uint8_t)strtol(buffer[i], NULL, 2);

            //각각 signed_char, ASCII_codes, unsigned_char 배열에 저장
            signed_char(result_sc, buffer_onebyte);
            ASCII_codes(result_ac, buffer_onebyte);
            unsigned_char(result_uc, buffer_onebyte);
        }
        //4 바이트일 때
        if((buffer_index - i + 1) % 4 == 0)
        {
            //32 비트(4 바이트)의 임시 버퍼 선언
            char buffer_temp[32] = "";

            //strcat 으로 비트로 표기된 문자열 이어 붙이기
            for(int j = i + 3; j >= i; j--)
                strcat(buffer_temp, buffer[j]);

            //문자열을 4 바이트 버퍼로 변환하여 할당
            buffer_fourbyte = (uint32_t)strtol(buffer_temp, NULL, 2);

            //signed_int, unsigned_int, sigend_float 배열에 저장
            signed_int(result_si, buffer_fourbyte);
            unsigned_int(result_ui, buffer_fourbyte);
            signed_float(result_sf, (int32_t)buffer_fourbyte);
        }
        //8 바이트일 때
        if((buffer_index - i + 1) % 8 == 0)
        {
            //64 비트(8 바이트)의 임시 버퍼 선언
            char buffer_temp[64] = "";
            for(int j = i + 7; j >= i; j--)
```

```
                strcat(buffer_temp, buffer[j]);

            //문자열을 8바이트 버퍼로 변환하여 할당
            buffer_eightbyte = (uint64_t)strtol(buffer_temp, NULL, 2);

            //signed_double 배열에 저장
            signed_double(result_sd, buffer_eightbyte);
        }
    }

    //버퍼 사이즈와 input 문자열 출력
    printf("BUFSIZ : %d\n\n", bufsiz);
    printf("input : %s\n\n", str);

    //저장된 결과 전부 출력
    printf("1. signed char : ");
    for(int i = 0; i < index_result[0]; i++)
        printf("%d ", result_sc[i]);
    printf("\n2. ASCII codes : ");
    for(int i = 0; i < index_result[1]; i++)
        printf("%c ", result_ac[i]);
    printf("\n3. unsigned char : ");
    for(int i = 0; i < index_result[2]; i++)
        printf("%ud ", result_uc[i]);
    printf("\n4. signed int : ");
    for(int i = 0; i < index_result[3]; i++)
        printf("%d ", (int32_t)result_si[i]);
    printf("\n5. unsigned int : ");
    for(int i = 0; i < index_result[4]; i++)
        printf("%ud ", (uint32_t)result_ui[i]);
    printf("\n6. signed float : ");
    for(int i = 0; i < index_result[5]; i++)
        printf("%.4f ", (float)result_sf[i]);
    printf("\n7. signed double : ");
    for(int i = 0; i < index_result[6]; i++)
        printf("%.4f ", (double)result_sd[i]);
    printf("\n");

    fclose(file);

    return 0;
}

//각 자료형에 알맞은 형변환과 함께 결과값 배열에 버퍼를 저장
void signed_char(char *str, uint8_t buffer)
{
    str[index_result[0]++] = (int8_t)buffer;
```

```
}

void ASCII_codes(char *str, uint8_t buffer)
{
    //만약 ASCII code 값을 벗어나면 .으로 저장
    if (buffer < 0 || buffer > 127)
        str[index_result[1]++] = '.';
    else
        str[index_result[1]++] = buffer;
}

void unsigned_char(unsigned char *str, uint8_t buffer)
{
    str[index_result[2]++] = buffer;
}

void signed_int(int *str, uint32_t buffer)
{
    str[index_result[3]++] = (int32_t)buffer;
}

void unsigned_int(unsigned int *str, uint32_t buffer)
{
    str[index_result[4]++] = (uint32_t)buffer;
}

void signed_float(float *str, int32_t buffer)
{
    str[index_result[5]++] = (float)buffer;
}

void signed_double(double *str, int64_t buffer)
{
    str[index_result[6]++] = (double)buffer;
}
```

- 20192800.c에는 과제1에서 요구하는 명세에 대한 구현 소스코드가 저장되어 있다.
- 소스코드의 대략적인 수행 절차는 다음과 같다.
  - "input"이라는 파일을 열고 파일의 끝까지 탐색하여 크기를 측정한다.
  - 파일의 크기를 바탕으로 문자열 배열을 동적 할당하고, 파일 내부에 있는 비트를 문자열에 저장한다.
  - 과제1에서 요구하는 자료형들의 출력을 위한 결과값 배열을 선언 및 할당

한다.

- ◆ Input 파일로부터 저장한 문자열 배열을 0번 인덱스부터 탐색하여 8비트, 즉 1바이트씩 끊어서 buffer 배열에 저장한다.
- ◆ 저장된 1바이트 Buffer 배열을 스택처럼 사용하기 위해 top 인덱스부터 거꾸로 탐색한다. index탐색이 각각 1, 4, 8일 때마다 if문에서 해당 바이트에 대한 변환과 결과값 저장을 수행한다.
- ◆ 마지막 printf 구간에서 모든 결과값을 출력한다.

■ Input의 비트에서 자료형으로 변환하여 저장하는 상세 내용은 다음과 같다.

- ◆ 먼저 input 파일로부터 비트 문자열을 담는 배열 str과 배열의 크기를 담는 str_length가 주어진다.
- ◆ 출력해야 하는 자료형은 1바이트 크기의 signed_char, ASCII_codes, unsigned_char와 4바이트 크기의 signed_int, unsigned_int, signed_float과 8바이트 크기의 signed_double로 총 7가지이다.
- ◆ 자료형의 결과값 출력을 위한 배열을 해당하는 자료형으로 선언하고 str_length를 바이트 크기로 나누는 방식으로 배열의 크기를 계산하여 동적 할당 해준다.
- ◆ 다음으로 1바이트를 단위로 하는 2차원 char형 배열 buffer를 선언한다. 이 역시 str_length를 통해 필요한 만큼의 크기만 메모리를 동적 할당해준다.
- ◆ 각 자료형에는 결과값 배열과 자료형 크기의 버퍼를 파라미터로 갖는 자료형 이름의 함수명을 가진 함수가 존재한다. 이 함수로 값을 넘겨주기 위한 1바이트, 4바이트, 8바이트의 고정 비트 크기를 갖는 버퍼를 선언한다. stdint.h 헤더파일에 저장된 uint8_t, uint32_t, uint64_t를 사용하였다.
- ◆ 비트값이 담긴 str배열을 8비트씩 끊어서 buffer에 저장한다.
- ◆ Input과 달리 메모리, 즉 buffer에 저장될 때에는 빅 엔디안으로 저장해야 하므로, buffer의 top 인덱스에서 0번 인덱스로 스택처럼 거꾸로 탐색하면 올바르게 비트 구획이 가능하다.
- ◆ Buffer를 탐색하며 탐색 횟수를 따로 저장하고, 만약 탐색 횟수가 각각 1, 4, 8에 해당하면 이를 검출하는 if문 내에서 비트 문자열을 정수로의 형변환과 결과값 저장을 위한 함수 호출이 이루어진다.
  - 4, 8바이트의 경우 1바이트의 buffer를 이어 붙여야 하므로, 그 개수만큼 다시 for문을 돌아서 strcat을 통해 임시 buffer에 넣어준다.
- ◆ 마지막으로 각 자료형의 함수가 호출되면 함수 내부에서 결과값을 인덱스에 해당하는 배열 위치에 저장하고 자료형에 맞게 형변환이 이루어진다.

■ 모든 작업이 끝나면 input의 버퍼 사이즈, input의 내용, 변환된 자료형의 결과값을 순서대로 출력한다. 출력은 공지의 "과제 1 입출력 예"와 동일하게 진행된다.

- Makefile

```
CC=gcc
CFLAGS=-I.
20192800: 20192800.c
    $(CC) -o 20192800 20192800.c $(CFLAGS)
```

  - Makefile 파일에는 소스코드를 컴파일하기 위한 정보와 명령어가 저장되어 있다.
  - 먼저 컴파일러의 종류를 담을 CC 변수를 선언하여 GCC를 사용하도록 지정했다.
  - CFLAGS 변수를 선언하여 -I. 옵션을 사용하도록 했다. 이 옵션은 컴파일할 때 헤더 파일을 검색할 위치를 현재 폴더에서 찾도록 한다.
  - Makefile을 통해 만들어진 실행 파일의 이름을 20192800로 설정하고 20192800.c 소스 코드를 사용하도록 명시했다.
  - 실제 빌드할 때의 명령어를 선언했던 변수를 이용하여 명시했다. gcc -o 20192800 20192800.c -I. 을 최종적으로 수행할 것이다.

3. 문제점 및 해결 방법
   - Input 비트의 메모리 저장 방식 이해 문제
     - 처음에 과제 명세서를 봤을 때 막연하게 input의 비트를 1바이트, 4바이트 등으로 끊어서 출력하면 된다고 생각했다. 그러나 공지의 "과제 1 입출력 예"와 비교했을 때 해당하는 값이 출력되지 않았고, 과제에 대한 이해가 틀렸다는 것을 알 수 있었다.
     - 결정적으로 4/9일 시스템프로그래밍 수업에서 교수님의 과제 설명을 통해 확실하게 이해할 수 있었다. Int형 변수가 1을 가졌을 때, 출력 상으론 리틀 엔디안을 따르지만, 메모리에 저장될 때에는 빅 엔디안을 따라서 1바이트 단위로 반대의 순서를 갖는다는 것을 확인했다.
   - 비트 문자열의 형변환 문제
     - Input을 통해 1바이트의 비트 문자열을 char 형으로 관리를 했지만, 이를 어떻게 전달하고 변환시켜야 할지 고민이 되었다.
       - 인터넷을 찾아본 결과 고정 비트 수의 자료형이 stdint.h에 선언되어 있음을 알 수 있었고 해당 자료형으로 버퍼를 선언하고 할당하여 해결할 수 있었다.
     - 또한, 처음에는 비트 쉬프트 연산과 OR 연산을 통해 버퍼에 넘겨주려 했으나, 원활히 이뤄지지 않는 문제가 있었다.
       - 이는 인터넷 검색을 통해 strtol이라는 정수 변환 함수를 대신 사용하여 해결할 수 있었다.

- 쓰레기 값 출력 문제
  - input값을 str에 저장할 때에나, 임시 버퍼를 이용하여 1바이트를 끊어서 저장할 때 자료형의 출력이 올바르게 이뤄지지 않는 것을 확인했다.
  - Printf를 통해 값을 출력해보니 쓰레기 값이 섞여있는 문제를 확인할 수 있었고, 문자열을 명확히 초기화하지 않아서 생긴 문제로 확인되어 이내 해결했다.