C애플리케이션 구현

학과	학번	이름
컴퓨터정보공학과	20191797	박별이

머리말

1학년 2학기에 C프로그래밍을 배우면서 1학년 1학기에 배웠던 자바와 언어가 비슷한 듯 다른 C언어의 생소한 개념들을 배우면서 어려움을 느꼈습니다. 하지만 Perfect C 교재를 수차례 읽으면서 C언어에 대해 이해하는 과정을 거치고 실제로 어느 부분에 사용하는지에 대한 호기심, 다양한 예제를 접해보는 등 더 깊이 있게 배우고 싶다고 느꼈습니다. 프로그래밍 언어에서 C언어는 기초이자 가장 중요한 언어이기 때문에 컴퓨터공학도로서 필수적으로 알아야 하는 부분이라고 생각했습니다.

전 세계적인 코로나19 감염으로 이론 과목인 C애플리케이션 구현을 모두 온라인 강의로 진행하였기 때문에 대면 수업이 아니라서 이해하는데 다소 어려운 부분이 있었으나 지난 학기와 마찬가지로 Perfect C 교재와 구글링을 통해 스스로 생각하면서 개념 이해를 하다 보니 C 언어 지식을 넓힐 수 있었습니다.

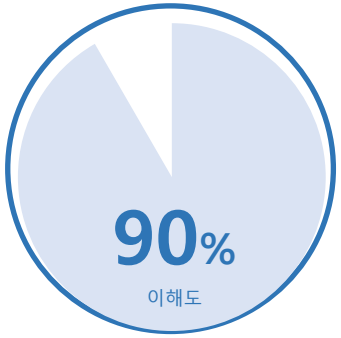
그래서 이번 C애플리케이션 구현 포트폴리오에서는 이번 학기에 배운 내용을 전체적으로 정리하면서 공부하면서 이해도가 떨어졌던 부분과 LAB에 있는 예제 표본을 스스로 해설을 하면서 PPT로 만들기 위해 노력할 것입니다. 학창시절부터 이론을 외우는 것도 중요하지만, 그와 관련된 문제를 풀면서 개념을 적용하는 것이 효과적이라고 생각했기 때문에 예제 코드에 관해 설명하면서 이 포트폴리오를 작성할 계획입니다.

학습 계획표

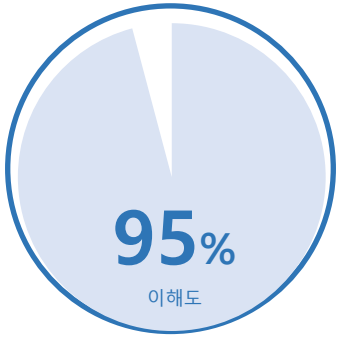
학습 기간	학습 내용
2020-03-16 ~ 2020-03-29	11. 문자와 문자열
2020-03-23 ~ 2020-04-05	12. 변수 유효범위
2020-04-06 ~ 2020-04-19	13. 구조체와 공용체
2020-04-13 ~ 2020-04-26	14. 함수와 포인터 활용
2020-04-20 ~ 2020-05-03	15. 파일 처리(15.3 이진파일 입출력까지)

목차

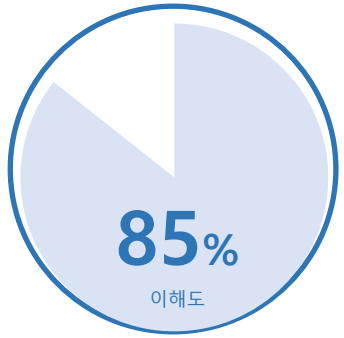
11. 문자와 문자열



12. 변수 유효범위



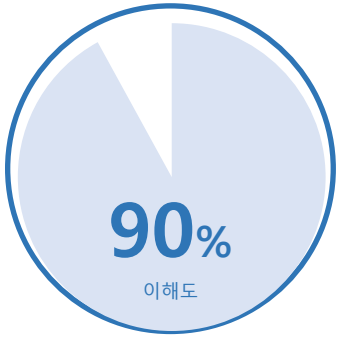
13. 구조체와 공용체



목차

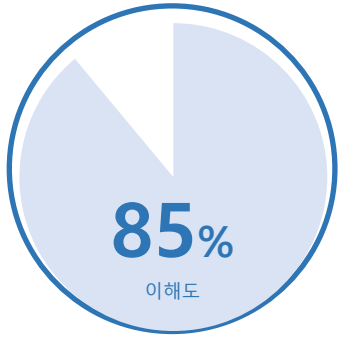
14. 함수와 포인터 활용

- 14.1 함수의 인자전달 방식 정리
- 14.1 함수의 인자전달 방식 실습예제 14-7
- 14.1 함수의 인자전달 방식 Lab14-1
- 14.2 포인터 전달과 반환 Lab14-2
- 14.3 함수 포인터와 void 포인터 Lab14-3



15. 파일 처리

- 15.1 파일 기초 Lab15-1
- 15.2 텍스트 파일 입출력 정리
- 15.2 텍스트 파일 입출력 Lab15-2
- 15.3 이진 파일 입출력 Lab15-3



11.1 문자와 문자열 Lab11-1

```c
// lineprint.c:
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main()
{
    char s[100];
    //문자배열 s에 표준입력한 한 행을 저장
    gets(s);

    //문자배열에 저장된 한 행을 출력
    char* p = s;
    while (*p)
        printf("%c", *p++);
    printf("₩n");

    return 0;
}
```

```
한 행을 표준입력으로 받아 문자 하나 하나를 그대로 출력
한 행을 표준입력으로 받아 문자 하나 하나를 그대로 출력
```

C언어에서 문자열 저장 자료형이 X
-> char형 변수에 문자를 저장하는 문자 배열을 사용함

문자 배열에 입력한 모든 문자 뒤엔 '₩0' 문자로 저장됨
-> 배열크기는 저장될 문자 수보다 1이 커야 함

gets()는 한 행의 문자열 입력에 유용한 함수

[enter] 키 누를 때까지 한 행을 버퍼에 저장 한 후 입력처리
-> 마지막에 입력된 '₩n'이 '₩0'으로 교체되어 배열에 저장

puts()는 한 행의 문자열 출력에 유용한 함수

gets()와 반대로 문자열의 마지막에 저장된 '₩0'를 ₩n'으로 교체하여 버퍼에 전송 -> 모니터에 출력함
오류가 발생하면 EOF(End Of File)를 반환

12. 포인터 변수 p에 배열 s를 저장
13. while문에서 p가 0이 아닐 때까지
14. p를 출력하고 끝난 후 1을 더한다.
15. 한 줄 [enter]을 한다.

11.2 문자열 관련 함수 정리

함수 strlen() = 문자열의 **길이**를 반환하는 함수
함수 memcpy() = 문자배열의 **복사**를 위한 함수
함수 memchr() = 문자배열에서 문자 이후의 **문자열을 찾**는 함수

문자열 비교와 복사, 그리고 문자열 연결 등과 같은 다양한 문자열 처리는
헤더파일 string.h에 함수원형으로 선언된 라이브러리 함수로 제공

함수 strcmp() = 인자인 두 문자열을 **사전 상의 순서로 비교**하는 함수
함수 strncmp() = 두 문자를 비교할 문자의 **최대 수를 지정**하는 함수

함수 strcpy() = 문자열을 **복사**하는 함수, 앞 인자 문자열에 뒤 인자 문자열을 복사
함수 strncpy() = 문자열을 복사하는 함수, **복사되는 최대 문자 수를 마지막 인자로 지정**하는 함수

함수 strcat() = 앞 문자열에 뒤 문자열의 null 문자까지 연결하여, **앞의 문자열 주소를 반환**하는 함수

함수 strtok() = 문자열에서 **구분자인 문자**를 여러 개 지정하여 **토큰을 추출**하는 함수
->결과를 저장한 변수가 NULL -> 더 이상 분리할 토큰이 없는 경우

함수 strlen() = NULL 문자를 제외한 **문자열 길이를 반환**하는 함수
함수 strlwr() = 인자를 모두 **소문자**로 변환하여 반환
함수 strupr() = 인자를 모두 **대문자**로 변환하여 반환

11.2 문자열 관련 함수 Lab11-2

```
1   // strreverse.c:
2   #include <stdio.h>
3   #include <string.h>
4
5   void reverse(char str[]);
6
7   int main(void)
8   {
9       char s[50];
10      memcpy(s, "C Programming!", strlen("C Programming!") + 1);
11      printf("%s₩n", s);
12
13      reverse(s);
14      printf("%s₩n", s);
15
16      return 0;
17  }
18
19  void reverse(char str[])
20  {
21      for (int i = 0, j = strlen(str) - 1; i < j; i++, j--)
22      {
23          char c = str[i];
24          str[i] = str[j];
25          str[j] = c;
26      }
27  }
```

함수 원형 선언

함수 호출

함수 정의

```
C Programming!
!gnimmargorP C
```

문자의 배열 관련 함수는 헤더파일 string.h에 함수원형이 정의되어 있으므로 적어야 사용이 가능

void ***memcpy**(void *dest, const void * src, size_t n)는 문자배열의 복사를 위한 함수
-> "C Programming!" 위치에서 char형 배열인 s에 strlen("C Programming!")+1 바이트를 복사한 후 s위치 반환

reverse()는 문자열 배열을 역순으로 저장하는 함수
-> for문 초기값은 int형 변수 i가 0이고 j는 14-1=13
조건문은 변수 i가 j보다 작을 때까지
증감문은 변수 i는 1씩 증가, j는 1씩 감소

23. char형 변수에 str[0] (즉, 'C')를 저장하고
24. str[0]에 str[13] (즉, '!')를 저장하고
25. str[13]에 c에 저장했던 'C'를 저장하면
문자열 배열을 역순으로 저장할 수 있다.

11.3 여러 문자열 처리 실습예제 11-14

```c
// file: commandarg.c
#include <stdio.h>

int main(int argc, char* argv[])
{
    int i = 0;

    printf("실행 명령행 인자(command line arguments) >>₩n");
    printf("argc = %d₩n", argc);
    for (i = 0; i < argc; i++)
        printf("argv[%d] = %s₩n", i, argv[i]);

    return 0;
}
```

명령행 인자 사용 방법
= 명령행에서 입력하는 문자열을 프로그램으로 전달하는 방법

main() 함수에서 두개의 인자 argc, argv
-> (int argc, char * argv[])로 기술해야 함

매개변수 argc = 명령행에서 입력한 문자열의 수
매개변수 argv[] = 명령행에서 입력한 문자열을 전달받는 문자 포인터 배열

주의할 점 = 실행 프로그램 이름도 하나의 명령행 인자에 포함

```
실행 명령행 인자(command line arguments) >>
argc = 1
argv[0] = C:₩Users₩oa₩source₩repos₩Ch11₩Debug₩Ch11.exe
```

11.3 여러 문자열 처리 Lab11-3

```
1  // file: strprocess.c
2  #include <stdio.h>
3
4  int main(void)
5  {
6      char str1[] = "JAVA";
7      char str2[] = "C#";
8      char str3[] = "C++";
9
10     char* pstr[] = { str1, str2, str3 };
11
12     //각각의 3개 문자열 출력
13     printf("%s ", pstr[0]);
14     printf("%s ", pstr[1]);
15     printf("%s₩n", pstr[2]);
16
17     //문자 출력
18     printf("%c %c %c₩n", str1[0], str2[1], str3[2]);
19     printf("%c %c %c₩n", pstr[0][1], pstr[1][1], pstr[2][1]);
20
21     return 0;
22 }
```

```
JAVA C# C++
J # +
A # +
```

여러 개의 문자열을 처리하려면
-> 문자 포인터 배열을 이용 or 문자의 이차원 배열을 이용

6~8. char형 문자열 배열에 각각 값을 입력
10. **문자 포인터 배열**을 이용하여 pstr[]에 위의 문자열 배열을 넣는다.
-> 장점 : pstr[0][3] = 'A'이지만 pstr[1][3]의 값은 없는 것 처럼 각각의 문자열 저장을 위한 최적의 공간 사용
단점 : 문자열 상수의 수정은 불가능

문자의 이차원 배열에서는 모든 열 수가 동일하게 메모리에 할당됨
-> 장점 : 문자열을 수정할 수 O
단점 : 문자열의 길이가 서로 다른 경우에는 '₩0'이 뒤에 들어가 낭비되는 메모리 공간O

13~15. 문자 포인터 배열을 이용하여 각 문자를 출력하려면 형식 제어 문자 %s를 사용하여 콘솔창 1번째 줄이 나옴
18. %c를 사용하여 str1[0][0], str2[1][1], str3[2][2]을 출력하면 콘솔창의 2번째 줄이 나옴
19. %c를 사용하여 문자 포인터 배열을 출력하면 콘솔창의 3번째 줄이 나옴

12.1 전역변수와 지역변수 Lab12-1

```c
//file fibonacci.c
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>

//전역변수
int count;
//함수원형
void fibonacci(int prev_number, int number);

int main(void) {
    //자동 지역변수
    auto prev_number = 0, number = 1;

    printf("피보나츠를 몇 개 구할까요?(3 이상) >> ");
    //전역변수를 표준입력으로 저장
    scanf("%d", &count);
    if (count <= 2)
        return 0;

    printf("1 ");
    fibonacci(prev_number, number);
    printf("₩n");
}
```

```
피보나츠를 몇 개 구할까요?(3 이상) >> 20
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

변수 참조가 유효한 범위를 변수의 유효 범위(scope)
-> 전역(외부) 변수의 유효범위는 프로젝트(전체 프로그램)과 파일
선언 -> 자동으로 초기값이 자료형에 맞는 0으로 지정
함수, 블록에서 전역변수와 같은 이름으로 지역변수 선언 가능
-> 함수 내부에서 이름 참조 -> 지역변수로 인식(전역변수X)
extern을 이용하여 다른 파일에서 선언된 전역변수임을 선언
-> 프로젝트 다른 파일에서 참조O
단점 : 예상하지 못한 값이 저장
-> 프로그램 어느 부분에서 수정되었는지 알기 어렵다.

-> 지역 변수의 유효범위는 함수 또는 블록
함수의 매개변수 = 함수 전체에서 사용 가능한 지역변수
지역변수 선언 후 -> 초기화X -> 쓰레기값 저장
할당되는 메모리 영역 = 스택(stack)
선언 부분 -> 자동 생성, 종료 부분 -> 자동 제거

함수 fibonacci(int prev_number, int number)
-> 이전 두 정수를 인자로 자기 자신을 호출하는 재귀함수

16. scanf()로 int형 전역변수 count에 사용자가 입력한 값을 저장
17~18. scanf()로 받은 값이 2보다 작거나 같으면 리턴함
21. 함수 fibonacci()를 호출함

12.1 전역변수와 지역변수 Lab12-1

```
25  void fibonacci(int prev_number, int number)
26  {
27      //정적 지역변수 i
28      static int i = 1;
29
30      //전역변수 count와 함수의 정적 지역변수를 비교
31      while (i++ < count)
32      {
33          //지역변수
34          int next_num = prev_number + number;
35          prev_number = number;
36          number = next_num;
37          printf("%d ", next_num);
38          fibonacci(prev_number, number);
39      }
40  }
```

피보나츠의 수 = 1, 1로 시작하여 이전 두 수를 더한 수

정적 지역변수 i
-> 프로그램이 실행되는 동안 계속해서 유지할 수 있는 변수
전역변수 대신 쓴 까닭은?
-> 변수에 접근하는 영역이 하나의 함수로 제한되기 때문

while조건문으로 count(사용자가 입력한 수)보다 i가 작을 경우 실행
34. while문 내에서 사용하는 지역변수로 int형 next_num 사용
이전 두 숫자를 더한 값을 넣음
35. 이전 숫자에 현재 숫자를 넣고
36. 현재 숫자에 다음 숫자를 넣음
37. 다음숫자를 출력하고 한 칸 띄어쓰기를 함
38. 함수fibonacci()를 다시 호출함

```
피보나츠를 몇 개 구할까요?(3 이상) >> 20
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

실행 결과를 보면 1 + 1 = 2, 1 + 2 = 3, 2+ 3 = 5로 정상적으로 실행하였음을 알 수 있다.

12.2 정적 변수와 레지스터 변수 Lab12-2

```c
//file: static.c
#include <stdio.h>

void process();

int main()
{
    process();
    process();
    process();

    return 0;
}

void process()
{
    //정적 변수
    static int sx;
    //지역 변수
    int x = 1;

    printf("%d %d₩n", x, sx);

    x += 3;
    sx += x + 3;
}
```

```
1 0
1 7
1 14
```

정적 지역변수
-> 함수를 종료해도 메모리에서 제거X
-> 유지 관리됨

정적 전역변수
->extern에 의해 다른 파일에서 참조 불가능
프로그램 크고 복잡
-> 부작용의 위험성이 존재

8~10. 함수 process()를 3번 호출하여 실행

18. 초기값을 지정하지 않았기 때문에 sx는 0

20. x는 지역변수이기 때문에 실행할 때마다 값을 1로 갖음

22. x와 sx의 값을 출력

24. x는 3을 더하지만 다시 실행하면 1

25. 메모리가 제거되지 않기 때문에 항상 값이 더해짐

기억부류 종료	전역	지역
auto	X	O
register	X	O
static	O	O
extern	O	X

extern은 컴파일러에 변수가 이미 어딘가에 존재
-> 이제 사용하겠다는 것을 알리는 구문에 사용
사용 범위는 전역 or 지역 한정

나머지 3개는 기억부류의 변수선언에서 초기값 저장O

auto는 지역변수 선언에 사용 -> 생략O
-> 지금까지 함수에 선언된 모든 변수가 auto 생략

register는 일반 메모리X -> CPU내부의 레지스터에 할당
-> 처리 속도 빠름 -> 반복문의 횟수 제어 효과적
register는 일반 메모리X -> 주소연산자 & 사용 불가능

static은 전역, 지역변수로 사용가능
-> 프로그램 시작 -> 메모리 할당, 종료 ->메모리 제거
초기생성 이후 메모리에서 제거X -> 저장값 유지, 수정O
초기값 지정X -> 자료형에 따라 0, '₩0', NULL 저장

12.3 메모리 영역과 변수 이용 정리

메모리 영역	변수의 종류
데이터 영역 -> 메모리 주소가 낮은 값 ~> 높은 값 으로 저장 장소 할당 프로그램이 시작되는 시점에 정해진 크기대로 고정된 메모리영역이 확보	전역변수 정적변수
힙 영역 -> 프로그램이 실행되면서 영역 크기가 계속적으로 변함 데이터 메모리 주소가 낮은값 ~> 높은값 사용X 공간이 동적으로 할당	동적할당되는 변수
스택 영역 -> 프로그램이 실행되면서 영역 크기가 계속적으로 변함 메모리 주소가 높은 값 ~> 낮은 값 으로 저장 장소 할당 함수 호출과 종료 -> 높은 주소 ~> 낮은 주소 메모리 할당 -> 제거 작업	함수 호출에 의한 형식 매개변수 함수 내부의 지역변수

변수 이용 기준(정리)

1. 실행속도 개선 -> 레지스터 변수 이용
2. 함수, 블록 내부에서 함수, 블록이 종료
   -> 계속적으로 메모리에 값 저장
   => 정적 지역변수 이용
3. 해당 파일 내부에서만 변수 공유 => 정적 전역변수
4. 프로그램 모든 영역 -> 전역변수
   가능하면 전역변수의 사용 줄이면-> 프로그램 이해 높여
   -> 발생하는 프로그램 문제 줄여

12.3 메모리 영역과 변수 이용 Lab12-3

```c
// file: bank.c
#include <stdio.h>

//전역변수
int total = 10000;

//입금 함수원형
void save(int);
//출금 함수원형
void withdraw(int);

int main(void)
{
    printf(" 입금액  출금액    총입금액    총출금액      잔고₩n");
    printf("================================================₩n");
    printf("%46d₩n", total);
    save(50000);
    withdraw(30000);
    save(60000);
    withdraw(20000);
    printf("================================================₩n");

    return 0;
}
```

함수가 종료되어도 정적 지역변수이기 때문에 메모리에 저장되어있어 수정 가능

```c
//입금액을 매개변수로 사용
void save(int money)
{
    //총입금액이 저장되는 정적 지역변수
    static int amount;
    total += money;
    amount += money;
    printf("%7d %17d %20d₩n", money, amount, total);
}

//출금액을 매개변수로 사용
void withdraw(int money)
{
    //총출금액이 저장되는 정적 지역변수
    static int amount;
    total -= money;
    amount += money;
    printf("%15d %20d %9d₩n", money, amount, total);
}
```

```
 입금액  출금액    총입금액    총출금액      잔고
================================================
                                          10000
 50000                 50000                60000
         30000                 30000        30000
 60000                110000                90000
         20000                 50000        70000
================================================
```

13.1 구조체와 공용체 정리

구조체 = 정수, 문자, 실수, 포인터, 배열등 연관성O 서로 다른 개별적인 자료형의 변수들을 하나의 단위로 묶은 새로운 자료형

-> 연관된 멤버로 구성되는 통합 자료형 => 대표적인 유도 자료형

-> 사용하려면 먼저 **구조체 틀을 정의**해야 함

구조체 하나 하나의 항목은 **구조체 멤버 or 필드**

멤버 선언 구문에는 **반드시 세미콜론**을 해야 함

구조체 멤버로는 다른 구조체 변수 및 구조체 포인터도 허용함

구조체 내의 변수에서 **초기값을 입력X**
-> **자료형에 따라 기본값** 0, 0.0, '₩0' 등으로 저장

선언된 구조체형 변수는 **접근 연산자 .**를 사용하여 멤버 참조 가능
ex) seoul.name, seoul.place.latitude 등

실제 구조체의 크기 >= 멤버의 크기의 합

구조체의 **동등비교**를 하려면 구조체 멤버 하나하나를 비교해야 함
ex) if(seoul.place.latitude == newyork.place.latitude)

공용체 = 서로 다른 자료형의 값을 동일한 저장공간에 저장하는 자료형

-> **공용체 변수의 크기 = 멤버 중 가장 큰 자료형의 크기**

공용체의 초기화 값은 정의시 **처음 선언한 멤버의 초기값으로만** 저장이 가능

공용체 변수로 멤버를 접근하기 위해서 구조체와 같이
접근 연산자 .을 사용

13.1 구조체와 공용체 Lab13-1

```
[서울] 위도= 37.3 경도= 126.6
[뉴욕] 위도= 40.8 경도= 73.9
```

```
1   // file: structcity.c
2   #include <stdio.h>
3   #include <string.h>
4
5   //지구 위치 구조체
6   struct position
7   {
8       double latitude;    //위도
9       double longitude;   //경도
10  };
11
12  int main(void)
13  {
14      //도시 정보 구조체
15      struct city
16      {
17          char* name;             //이름
18          struct position place;  //위치
19      };
20      struct city seoul, newyork;
21
22      seoul.name = "서울";
23      seoul.place.latitude = 37.33;
24      seoul.place.longitude = 126.58;
25
26      newyork.name = "뉴욕";
27      newyork.place.latitude = 40.8;
28      newyork.place.longitude = 73.9;
29
30      printf("[%s] 위도= %.1f 경도= %.1f₩n",
31          seoul.name, seoul.place.latitude, seoul.place.longitude);
32      printf("[%s] 위도= %.1f 경도= %.1f₩n",
33          newyork.name, newyork.place.latitude, newyork.place.longitude);
34
35      return 0;
36  }
```

전역 구조체

지역 구조체

6~10. 전역 구조체를 정의와 변수 선언을 함께 함
멤버는 초기값 지정을 하지 않았기 때문에 0.0이 들어감

15~19. 구조체 멤버로 다른 구조체 변수를 허용하기 때문에
위치를 place로 하여 멤버로 사용함

20. 구조체 struct city형인 seoul과 newyork를 선언함

22~24. seoul에 접근 연산자 .를 사용하여 멤버를 참조하여 초기화 함

26~28. newyork에 접근 연산자 .를 사용하여 멤버를 참조하여 초기화 함

30~31. 접근 연산자 .를 사용하여 첫번째 콘솔창과 같이 출력함

32~33. 접근 연산자 .를 사용하여 첫번째 콘솔창과 같이 출력함

13.2 자료형 재정의 Lab13-2

```c
// file: typemovie.c
#include <stdio.h>

int main(void)
{
    //영화 정보 구조체
    typedef struct movie
    {
        char* title;    //영화제목
        int attendance; //관객수
    } movie;

    movie assassination;

    assassination.title = "암살";
    assassination.attendance = 12700000;

    printf("[%s] 관객수: %d₩n", assassination.title, assassination.attendance);

    return 0;
}
```

```
[암살] 관객수: 12700000
```

typedef = 이미 사용되는 자료 유형을 다른 새로운 자료형 이름으로 재정의 할 수 있도록 하는 키워드
-> 프로그램의 시스템 간 호환성과 편의성을 위해 필요
일반 변수와 같이 그 사용 범위를 제한함

옆의 예제는 구조체 자료형을 재정의해서 사용하였는데
구조체 정의 자체를 typedef와 함께 처리함
-> 이 경우 구조체 태그이름이 생략 가능

이 방법 외에 일반적인 구조체가 정의된 상태에서 typedef 사용하여
구조체 struct movie를 movie로 재정의 할 수도 있음
ex) typedef struct movie movie;

13. movie 자료형으로 assassination을 선언

15~16. assassination에 접근연산자 .를 사용하여 초기값을 저장

18. assassination에 저장한 멤버를 접근연산자 .를 사용하여 출력

13.3 구조체와 공용체의 포인터와 배열 Lab13-3

```
   제목      감독    관객수
========================
[    명량] 김한민 17613000
[국제시장] 윤제균 14257000
[  베테랑] 류승완 13383000
```

```c
// file: structmovie.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

int main(void)
{
    //영화 정보 구조체
    typedef struct movie
    {
        char* title;        //영화제목
        int attendance;     //관객수
        char director[20];  //감독
    } movie;

    movie box[] = {
    { "명량", 17613000, "김한민" },
    { "국제시장", 14257000, "윤제균" },
    { "베테랑", 13383000} };

    //영화 베테랑의 감독을 류승완으로 저장
    strcpy(box[2].director, "류승완");

    printf("   제목      감독    관객수₩n");
    printf("========================₩n");
    for (int i = 0; i < 3; i++)
        printf("[%8s] %6s %d₩n",
            box[i].title, box[i].director, box[i].attendance);

    return 0;
}
```

구조체 포인터 = 구조체의 주소값을 저장할 수 있는 변수
-> 멤버 접근 연산자 ->는 **p->title**과 같이 사용함
포인터 p가 가리키는 구조체 변수의 멤버 title을 접근하는 연산식

p->title뿐만 아니라**(*p).title**로도 사용 가능

연산식 *p.title은 접근 연산자(.)가 간접연산자(*)보다 우선순위가 빠름
-> *(p.title)과 같은 연산식

연산자 ->와 .은 우선순위 1위이고 결합성은 좌에서 우
**연산자 ***은 우선순위 2위이고 결합성은 우에서 좌

공용체 포인터도 구조체 포인터와 접근 연산자가 동일함

구조체 배열 = **동일한 구조체 변수가 여러 개** 필요할 때 선언하여 이용O

9~14. 구조체 struct movie를 정의하면서 typedef를 이용하여
동시에 자료형 movie도 정의

16~19. 구조체 movie의 box[] 배열 선언하면서 바로 초기값을 대입함

22. 초기값을 입력하지 않은 box[2].director에 문자열을 복사하는 함수
strcpy()를 사용하여 "류승완"을 저장

26. for문을 이용하여 배열 box[]에 저장된 값을 출력

14.1 함수의 인자전달 방식 정리

C 언어는 함수의 인자 전달 방식
= 기본적으로 **값에 의한 호출(call by value)** 방식

값에 의한 호출 방식
= 함수 호출 시 실인자의 값이 형식인자에 복사되어 저장된다는 뜻
-> **함수 외부의 변수를 함수 내부에서 수정할 수 X**

포인터를 매개변수로 사용
-> 함수로 전달된 **실인자의 주소**를 이용 -> **해당 변수 참조O**
-> 함수에서 주소의 호출 => **참조에 의한 호출(call by reference)**

함수의 매개변수로 배열 전달
= 배열의 첫 원소를 참조 매개변수로 전달하는 것

배열크기에 관계없이 배열 원소의 합 구하는 함수를 만들려면
-> 배열의 크기도 하나의 인자로 사용해야 함
ex) sum(data, 5); -> double sum(double ary[], int n)

함수 원형의 매개변수 -> 배열 이름 생략O ex) double[]
함수 호출 -> 배열 인자는 반드시 배열이름으로 기술 ex) sum(data, 5)

배열 point에서 간접연산자 사용 배열원소의 접근 방법 = *(point + i)
배열 point를 가리키는 포인터변수 address를 선언하여
배열원소의 접근 방법 = *(address++)
But, 배열 point는 주소 상수이기 때문에 *(point++)는 불가능

함수헤더에 int ary[] == int *ary
함수정의에서 배열원소의 참조방법
-> ary[i], *(ary +i), *ary++, *(ary++)이 가능함

배열크기 계산 방법
-> 배열크기(배열원소의 수) = seizeof(배열이름) / sizeof(배열원소)
ex) int arraysize = sizeof(data) / sizeof(data[0]);

다차원 배열을 인자로 이용하는 경우
-> 함수원형, 함수 정의의 헤더에서 첫번째 대괄호를 제외한 다른 모든 크기는 반드시 기술

다차원 배열에서 sizeof(x)는 배열 전체의 바이트 수,
sizeof(x[0])는 1행의 바이트 수, sizeof(x[0][0])는 1번째 원소의 바이트 수

14.1 함수의 인자전달 방식 실습예제 14-7

```
평균 3.40
```

```c
// file: vararg.c
#include <stdio.h>
#include <stdarg.h>

double avg(int count, ...); //int count 이후는 가변인자 ...

int main(void)
{
    printf("평균 %.2f₩n", avg(5, 1.2, 2.1, 3.6, 4.3, 5.8));

    return 0;
}

//가변인자 ... 시작 전 첫 고정 매개변수는
//이후의 가변인자를 처리하는데 필요한 정보를 지정
//여기서에서는 가변인자의 수를 지정
double avg(int numagrs, ...)
{
    //가변인자 변수 선언
    va_list argp;

    //numargs 이후의 가변인자 처리 시작
    va_start(argp, numagrs);

    double total = 0; //합이 저장될 변수
    for (int i = 0; i < numagrs; i++)
        //지정하는 double 형으로 가변인자 하나 하나를 반환
        total += va_arg(argp, double);

    //가변인자 처리 종료
    va_end(argp);

    return total / numagrs;
}
```

가변인자
= 함수에서 인자의 수와 자료형이 결정X 함수 인자 방식
-> 함수 정의 시 **가변인자의 매개변수는 ...**으로 기술함
반드시 **앞** 부분에는 매개변수가 **고정적**
마지막 부분에는 **고정적이면 X**
구현하기 위해 헤더파일 **stdarg.h** 필요

가변인자 선언 = 처리할 변수를 하나 만드는 것

가변인자 처리 시작 = 위의 선언된 변수에서 마지막 고정인자를 지정
가변 인자의 시작 위치를 알림

가변인자 얻기 = 가변인자 각각의 자료형 지정- > 가변인자를 반환
-> 이때 함수 va_arg는 헤더파일에 정의된 매크로 함수

가변인자 처리 종료 = 가변인자에 대한 처리를 끝냄
-> 이때 함수 va_end는 헤더파일에 정의된 매크로 함수

14.1 함수의 인자전달 방식 Lab14-1

```
2 4 6 8 10
```

```c
// file: aryprocess.c
#include <stdio.h>

void aryprocess(int* ary, int SIZE);

int main(void)
{
    int data[] = { 1, 3, 5, 7, 9 };

    int aryLength = sizeof(data) / sizeof(int);
    aryprocess(data, aryLength);
    for (int i = 0; i < aryLength; i++)
        printf("%d ", *(data + i));
    printf("₩n");

    return 0;
}

void aryprocess(int* ary, int SIZE)
{
    for (int i = 0; i < SIZE; i++)
        (*ary++)++; //(*(ary++))++;
        // ++(*(ary++)); ++(*ary++); ++*ary++; //동일한 기능
}
```

4. 함수 aryprocess()는 int형 포인터를 매개변수로 사용

8. int형 data[] 배열에 상수를 넣음

10. 배열 크기를 계산하기 위해 sizeof(배열이름)/ sizeof(배열원소)

11. 함수 aryprocess()의 매개변수로 data와 aryLength를 넣음

12~13. for문을 이용하여 aryLength보다 작을 때까지 *(data + i)를 출력

19~24. 함수 aryprocess()는 인자로 사용된 배열의 내부 원소를
for문을 이용하여 모두 1씩 증가시킴
-> 함수로 전달된 실인자의 주소를 이용
-> 해당 변수를 참조할 수 있기때문에 배열 data[]값을 바꿈

14.2 포인터 전달과 반환 Lab14-2

```
제목: 절대자바, 저자: 강환수, ISBN: 123987
제목: 파이썬웹프로그래밍, 저자: 김석훈, ISBN: 2398765
```

```c
1  //file: bookreference.c
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <stdio.h>
4  #include <string.h>
5
6  typedef struct book
7  {
8      char  title[50];
9      char  author[50];
10     int   ISBN;
11 } book;
12
13 void print(book* b);
14
15 int main()
16 {
17     book python = { "파이썬웹프로그래밍", "김석훈", 2398765 };
18     book java;
19     strcpy(java.title, "절대자바");
20     strcpy(java.author, "강환수");
21     java.ISBN = 123987;
22     print(&java);
23     print(&python);
24
25     return 0;
26 }
27
28 void print(book* b)
29 {
30     printf("제목: %s, ", b->title);
31     printf("저자: %s, ", b->author);
32     printf("ISBN: %d₩n", b->ISBN);
33 }
```

함수를 호출하는 방법 : 주소연산자 &를 사용

함수의 결과를 포인터로 반환할 때 *add(&sum, m, n)으로 호출
-> int * add(int *psum, int a, intb) { *psum = a + b; return psum; }으로
반환 타입을 int으로 하고 반환값속에 값을 넣으면 사용할 수 있음

지역변수 주소값의 반환 -> 문제 발생 가능성O(컴파일 경고)

const = 수정을 원하지 않는 함수의 인자 앞에 쓰는 키워드

6~11. 구조체 struct book을 자료형 book으로 정의

13. 함수 print()는 인자가 (book *b)으로
    구조체를 포인터로 받아 책 정보를 출력

17. book 자료형으로 python을 선언하고 초기화
18. java 자료형으로 python을 선언함
19~20. 함수 strcpy()로 java의 멤버에 값을 입력
21. 함수를 사용하지 않고 접근 연산자를 이용해 값을 입력
22~23. 함수 print()에 주소연산자 &를 사용하여 호출함

28. 함수 print()의 인자가 book* b로 함수 내부에는 printf() 구문으로
    각각의 제목, 저자, ISBN을 출력함
30~32. 구조체 포인터를 인자로 사용하여 멤버 접근 연산자 ->를 이용함

14.3 함수 포인터와 void 포인터 Lab14-3

```c
1   // file: funpointer.c
2   #include <stdio.h>
3
4   int add(int a, int b);
5   int mult(int a, int b);
6   int subt(int a, int b);
7
8   int main(void)
9   {
10      int(*pfunary[3])(int, int);
11      pfunary[0] = add;
12      pfunary[1] = mult;
13      pfunary[2] = subt;
```

```
* 결과: 15
+ 결과: 8
- 결과: -2
```

포인터 장점 : 다른 변수를 참조하여 읽거나 쓰는 것 가능O
-> 하나의 함수 이름으로 필요에 따라 여러 함수를 사용 -> 편리

함수 포인터 = 함수의 주소값을 저장하는 포인터 변수

void (*pf) (double*, double); 처럼 변수이름 앞에 *이 있으면
반드시 소괄호를 사용해야 한다.

pf = &add; 처럼 주소연산자 사용 가능
pf = add(); 함수호출 대입 불가능

pf = add;로 함수 포인터 변수에 함수 add()의 주소값이 저장
-> 함수 호출 pf(&result, m, n);으로 add(&result, m, n);을 대체함

함수 포인터 배열 = 함수 포인터가 원소인 배열

4~6. 반환타입이 int형이고 매개변수가 int a, int b인 **함수 원형**

10. 반환타입이 int형이고 인자목록이(int, int)인 **함수 포인터 배열** *pfunary[3]의 선언

11~13. **함수 3개를 각각의 배열원소에 저장**

14.3 함수 포인터와 void 포인터 Lab14-3

```
15      char* ops = "*+-";
16      char op;
17      while (op = *ops++)
18          switch (op) {
19          case '+': printf("%c 결과: %d\n", op, pfunary[0](3, 5));
20              break;
21          case '-': printf("%c 결과: %d\n", op, pfunary[2](3, 5));
22              break;
23          case '*': printf("%c 결과: %d\n", op, pfunary[1](3, 5));
24              break;
25          }
26
27      return 0;
28  }
29
30  int add(int a, int b)
31  {
32      return a + b;
33  }
34  int mult(int a, int b)
35  {
36      return a * b;
37  }
38  int subt(int a, int b)
39  {
40      return a - b;
41  }
```

```
* 결과: 15
+ 결과: 8
- 결과: -2
```

15. char형 포인터 변수 ops에 "*+-"대입

17. while 조건문은 char형 op에 포인터변수 *ops를 대입하고
수행할 때마다 포인터변수 ops를 1씩 증가 (* -> + -> - 순)

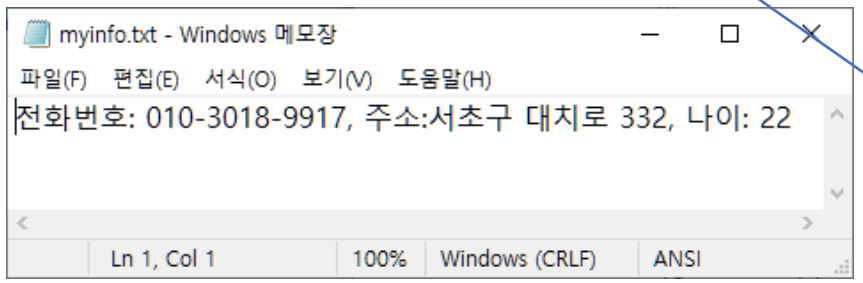
18~25. switch-case문으로 op에 저장된 값이 case와 일치할 때
printf()로 결과를 출력
if) pfunary[0](3,5)에서 pfunary[0]은 add이므로 add함수에
들어가서 3과 5를 더하면 리턴값으로 8이 출력

30~41. 반환타입이 int형인 add(덧셈), mult(곱셈), subt(뺄셈)
을 수행하고 return하는 함수이다.

void 포인터 = 자료형 무시, 주소값만 다루는 포인터(만능 포인터)
-> 모든 주소를 저장O But, 가리키는 변수 참조, 수정X
=> 자료형 정보 없이 임시로 주소만 저장하는 포인터
변수 참조 -> 자료형 변환 필요

15.1 파일 기초 Lab15-1

```c
// file: basicfileio.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h> //for exit()

int main()
{
    FILE* f; //파일 포인터

    //파일 열기 함수 fopen()과 fopen_s()
    if ((f = fopen("myinfo.txt", "w")) == NULL)
        //if (fopen_s(&f, "myinfo.txt", "w") != 0)
    {
        printf("파일이 열리지 않습니다.₩n");
        exit(1);
    };
```




```
전화번호: 010-3018-9917, 주소:서초구 대치로 332, 나이: 22
프로젝트 폴더에서 파일 myinfo.txt를 메모장으로 열어 보세요.
```

파일은 2가지 유형
1. 텍스트 파일 = 문자 기반의 파일, 문자 코드값(아스키 코드) 저장
2. 이진 파일 = 그림, 동영상, 실행 파일 등 각각의 목적에 알맞은 자료
   컴퓨터 내부 형식으로 저장되는 파일
   메모리 자료 내용에서 변환 거치지X -> 파일에 기록
   -> 입출력 속도도 텍스트 파일에 비해 빠름

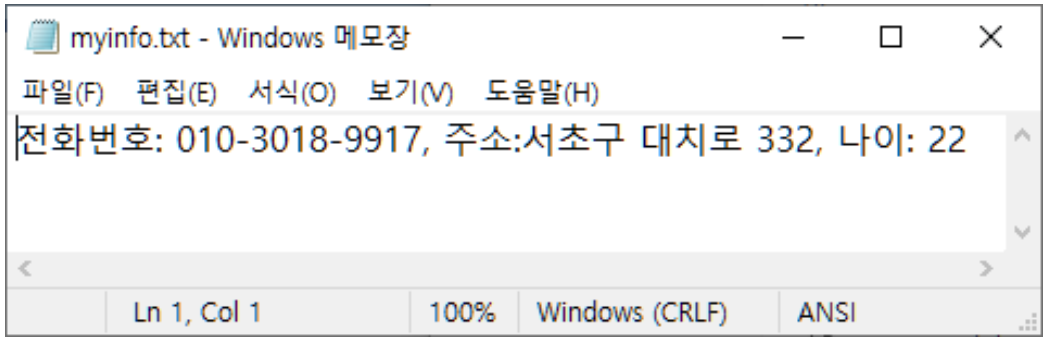
자료 입력과 출력 = 자료의 이동 -> 이동 경로 필요
입출력 스트림 = 입출력 시 이동 통로
키보드 (자료이동 경로)~> 프로그램 = 표준입력 스트림 ex)함수 scanf()
프로그램 ~> 모니터의 콘솔 = 표준출력 스트림 ex)함수 printf()
다른 곳 (입력 스트림)~> 프로그램 (출력 스트림)~> 다른 곳

파일 스트림 = 보조기억장치의 파일과 프로그램을 연결하는 전송경로
->파일 입력 스트림, 파일 출력 스트림

파일 스트림 열기 -> 함수 fopen() or fopen_s() 사용
FILE = 헤더 파일 stdio.h에 정의되어 있는 구조체 유형
   파일을 표현하는 C 언어의 유도 자료형
함수 fopen() = 인자가 파일이름과 파일열기 모드, 파일스트림 연결 성공
   -> 파일 포인터를 반환, 실패 -> NULL 반환
함수 fopen_s() = 파일 스트림 연결 성공 -> 정수 0 반환,
   실패 -> 양수 반환
ex) fopen_s(파일 포인터의 주소값, 파일 이름, 파일열기 종류인 모드)

15.1 파일 기초 Lab15-1

```c
// file: basicfileio.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h> //for exit()

int main()
{
    FILE* f; //파일 포인터

    //파일 열기 함수 fopen()과 fopen_s()
    if ((f = fopen("myinfo.txt", "w")) == NULL)
        //if (fopen_s(&f, "myinfo.txt", "w") != 0)
    {
        printf("파일이 열리지 않습니다.₩n");
        exit(1);
    };

    //파일에 쓰려는 자료
    char tel[15] = "010-3018-9917";
    char add[30] = "서초구 대치로 332";
    int age = 22;
    //파일"basic.txt"에쓰기
    fprintf(f, "전화번호: %s, 주소:%s, 나이: %d₩n", tel, add, age);

    //파일 닫기
    fclose(f);
    //표준출력 콘솔에쓰기
    printf("전화번호: %s, 주소:%s, 나이: %d₩n", tel, add, age);
    puts("프로젝트 폴더에서 파일 myinfo.txt를 메모장으로 열어 보세요.");

    return 0;
}
```




```
전화번호: 010-3018-9917, 주소:서초구 대치로 332, 나이: 22
프로젝트 폴더에서 파일 myinfo.txt를 메모장으로 열어 보세요.
```

파일열기 종류인 모드가 텍스트 파일
-> 읽기모드 r = 읽기O, 쓰기X
쓰기모드 w = 읽기X, 쓰기O
추가모드 a = 읽기X, 파일 중간에 쓰기X, 추가적으로 쓰는 것만O

함수 **fclose()** = fopen()으로 연결한 파일 스트림 닫는 기능
-> 파일 처리가 모두 끝나면 호출하여 반드시 파일을 닫음

함수 **fprintf()** = 파일에 서식화된 문자열을 출력하는 함수

파일이 쓰기모드가 X -> 함수 exit()로 종료 -> 헤더파일 stdlib.h가 필요
함수 **exit()** = 함수 강제종료 기능, 인자값 == 0 -> 정상종료, 이외 비정상

함수 **puts()** = 한 행의 문자열 출력에 유용한 함수

15.2 텍스트 파일 입출력 정리

표준파일	키워드	장치(device)
표준입력	stdin	키보드
표준출력	stdout	모니터 화면
표준에러	stderr	모니터 화면

함수 scanf_s() = 표준입력을 빈칸으로 구분 -> 빈칸X 이름을 입력
-> scanf()와 다르게 형식제어문자 %s를 사용하는
문자열 입력에 저장되는 버퍼
+버퍼의 크기인 인자를 하나 더 삽입

함수 fscanf_s() = fscnaf()와 다르게 저장되는 버퍼의 크기인
인자가 하나 추가

함수 fgets() = 파일로부터 **한 행의 문자열을 입력**받는 함수

함수 fputs() = 파일로 **한 행의 문자열을 출력**하는 함수

함수 feof() = 파일 스트림의 **EOF(End OF File) 표시를 검사**하는 함수
-> 읽기 작업이 이전 부분 읽으면 -> 0 반환
단순히 파일 지시자가 파일 끝 -> 결과 = 0

함수 ferror() = 파일 처리에서 **오류가 발생했는지 검사**하는 함수
-> 이전 파일 처리 오류가 발생 -> 0이 아닌 값 반환

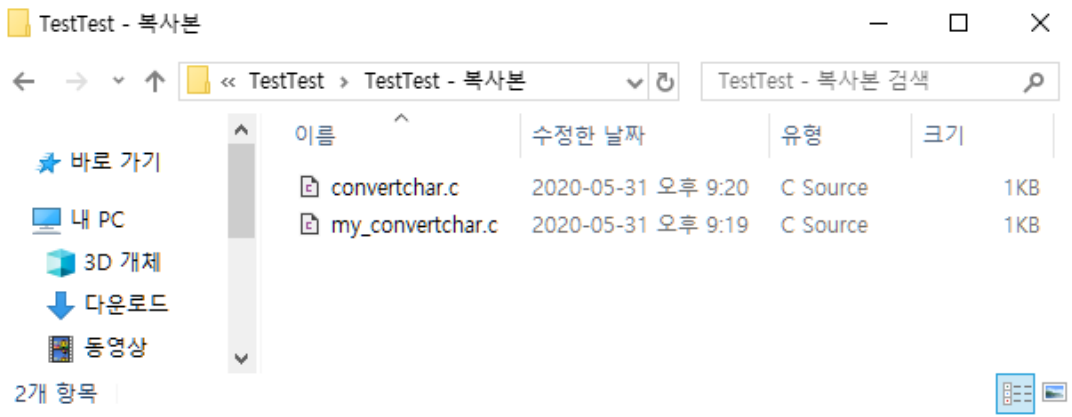
함수 fgetc(), getc() = 파일로부터 **문자 하나를 입력** 받는 함수

함수 fputc(), putc() = **문자 하나를 파일로 출력**하는 함수

15.2 텍스트 파일 입출력 Lab15-2

```c
// file: convertchar.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main(void)
{
    FILE* f1, * f2;
    if ((f1 = fopen("convertchar.c", "r")) == NULL) {
        printf("cannot open this file\n");
        exit(1);
    }
    if ((f2 = fopen("my_convertchar.c", "w")) == NULL) {
        printf("cannot open this file\n");
        fclose(f1);
        exit(1);
    }

    char a;
    while ((a = getc(f1)) != EOF)
    {
        if (isalpha(a))
            if (islower(a))
                a = toupper(a);
            else if (isupper(a))
                a = tolower(a);
        putc(a, f2);
    }

    fclose(f1);
    fclose(f2);
    printf("File my_convertchar.c is created!!!\n");

    return 0;
}
```

```
File my_convertchar.c is created!!!
```



10~18. if조건문으로 함수 fopen()로 파일스트림 연결 실패하였을 때 NULL 반환이면 내부 문장들을 수행
"r"은 읽기모드이고 "w"는 쓰기모드
함수 exit()는 함수 강제종료 기능으로 헤더파일 stdlib.h가 필요

21. 함수 getc()는 파일로부터 문자 하나를 읽어 그 문자가 EOF가 아니면 계속 변환하여 출력

23~27. 현재 파일의 알파벳에서 대문자 -> 소문자 소문자 -> 대문자
28. 함수 putc()는 문자 하나를 파일로 출력하는 함수

31~32. 함수 fclose()는 fopen()으로 연결한 파일 스트림 닫는 기능

33. my_convertchar.c 파일이 생성되었음을 알려주는 출력문

15.3 이진 파일 입출력 Lab15-3

```c
//file: studentinfo.c
#include <stdio.h>

//구조체 자료형 student 정의
typedef struct student
{
    char dept[40];  //학과
    char name[20];  //이름
    int snum;       //학번
} student;

int main()
{
    student mylab[] = {
        { "컴퓨터정보공학과", "김하늘", 201698657},
        { "컴퓨터정보공학과", "백규정", 201648762 },
        { "컴퓨터소프트웨어공학과", "김효주", 201665287 } };

    FILE* f;
    char fname[] = "student.bin";
    fopen_s(&f, fname, "wb");
    int size = sizeof(mylab) / sizeof(student);
    fwrite(mylab, sizeof(student), size, f);
    fclose(f);
```

```
        컴퓨터정보공학과    김하늘   201698657
        컴퓨터정보공학과    백규정   201648762
컴퓨터소프트웨어공학과    김효주   201665287
```

함수 fprintf(), fscanf(), fscanf_s() = 자료의 입출력을 텍스트 모드로 처리

함수 fwrite(), fread() = 이진 모드로 블록 단위 입출력을 처리
이진파일은 C 언어의 자료형을 모두 유지
-> 바이트 단위로 저장되는 파일

함수 fwrite() = 바이트 단위로 원하는 블록을 파일에 출력하기 위한 함수
-> 출력된 자료는 함수 fread()로 입력해야 자료유형 유지O

fwrite(출력될 자료의 주소값, 출력될 자료 항목의 바이트 크기, 출력될 항목의 개수, 출력될 파일 포인터)

fread(저장될 버퍼 자료의 주소값, 입력될 자료 항목의 바이트 크기, 입력될 항목의 개수, 입력 파일 포인터)

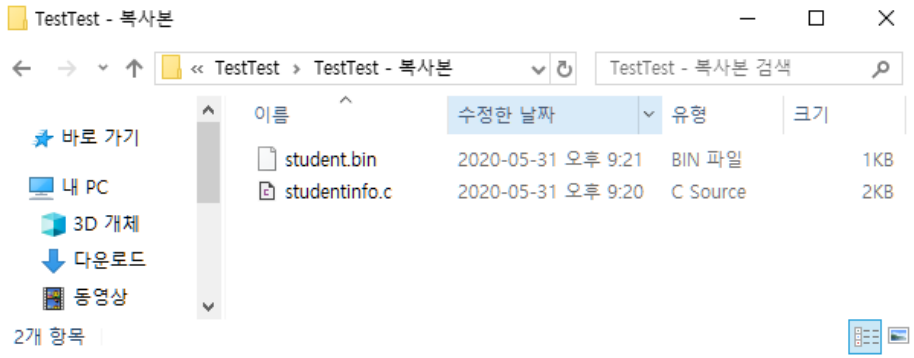
5~10. 학생의 정보를 표현하는 구조체 struct student를 정의하면서 typedef를 이용하여 동시에 자료형 student도 정의

14~17. 자료형 student 배열 mylab[]에 학생 3명의 정보를 초기화 저장

20. char형 배열 fname[]에 파일 이름을 넣음
21. student.bin을 열기모드로 파일을 열음
22. 출력될 항목의 개수를 구하는 식
23. 함수 fwrite()로 바이트 단위로 원하는 블록을 파일에 출력
24. 함수 fclose()는 fopen_s()로 연결한 파일 스트림 닫는 기능

15.3 이진 파일 입출력 Lab15-3

```
26      //다시 읽기 위해 오픈
27      fopen_s(&f, fname, "rb");
28      //파일에서 구조체 배열 모두를 한번에 읽어 다시 저장된 배열을 출력
29      student lab[10]; //다시 파일의 내용을 저장할 배열 선언
30      //파일 f에서 sizeof(student) 크기로 size 수만큼 읽어 lab에 저장
31      fread(lab, sizeof(student), size, f);
32      for (int i = 0; i < size; i++)
33          fprintf(stdout, "%24s%10s%12d\n", lab[i].dept, lab[i].name, lab[i].snum);
34      fclose(f);
35
36      /*
37      //파일에서 구조체 하나씩 읽어 읽을 내용이 있으면 출력
38      student one;
39      while (fread(&one, sizeof(student), 1, f))
40          fprintf(stdout, "%24s%10s%12d\n", one.dept, one.name, one.snum);
41      fclose(f);
42      */
43
44      return 0;
45  }
```




```
        컴퓨터정보공학과       김하늘    201698657
        컴퓨터정보공학과       백규정    201648762
  컴퓨터소프트웨어공학과       김효주    201665287
```

27. 함수 fopen_s()로 파일을 열음

31. 함수 fread()로 배열을 모두 입력

32. for문을 이용해 함수 fprintf()를 사용하여 파일에 서식화된 문자열을 출력

33. 배열 lab[]에 저장된 학생 정보를 접근 연산자.를 사용하여 다시 콘솔에 출력

34. 함수 fclose()로 fopen_s()로 연결한 파일 스트림을 닫음

맺는말

C애플리케이션 구현 중간고사 대체 포트폴리오를 작성하면서 그동안 온라인 수업에서 들었던 내용을 체계적으로 정리할 기회를 얻게 되어서 교수님께 감사했습니다. 직접 얼굴을 보며 수업을 듣는 것이 아니어서 실제 강의보다 집중이 어려워 다시 Perfect C 교재를 천천히 읽으면서 개념들을 제 것으로 만드는 과정이 매우 뿌듯했습니다.

수업에 사용했던 교재뿐만 아니라 시중에 나와 있는 책을 몇 번 더 읽고 코드를 직접 작성한다면 C 언어를 완전히 이해하고 원하는 코드를 짤 수 있을 것 같습니다.

15.3 이진 파일 입출력까지 포트폴리오를 모두 작성하였기 때문에 이 이후에는 15.4 파일접근처리와 16 동적 메모리와 전처리 부분을 이어서 작성하고 수업을 들으며 어려워서 풀지 못하였던 각 단원의 마지막에 있는 프로그래밍 연습 부분을 풀어서 깃허브에 올리는 것을 목표로 삼아 공부할 것입니다.

C애플리케이션 구현

중간고사 대체 자기주도학습 포트폴리오

끝까지 읽어주셔서 감사합니다!