

## C 포트폴리오

20190717 조나영

## 머리말

이 글은 C 언어의 문자와 문자열로 시작하여 변수의 유효범위, 구조체와 공용체, 함수와 포인터, 파일처리까지 노트를 정리한 글이다. 필자가 강의를 들으며 노트정리한 글과 강의교재의 합작으로 필자가 생각하기에 중요해서 알아두면 좋겠다고 생각한 부분만 핵심적으로 뽑아서 넣은 글이다.

필자가 생각하기에 이 글은 핵심부분만 뽑아서 넣었기에 C 언어 공부를 한 후 나중에 한번 훑어보는 식으로 살펴보면 좋을 것이다. 아니면 다른 C 언어교재와 같이 비교해보면서 공부를 하는 식으로 참고서 느낌으로 사용하면 좋다.

마지막으로 내용구성에 대해 설명하겠다

page | 3

### 1주차

#### 핵심내용

- 문자와 문자열
- 다양한 문자 임출력
- 문자열 관련 함수( strcmp(), strcpy() )

핵심내용 정리

#### 문자와 문자열

문자 : 영어의 알파벳, 한글의 한 글자 같이 하나의 글자로 되어있는 것 (작은따옴표)

ex) 'A', '가'

문자열 : 문자가 모여 있는 것 즉, 문자의 모임 (큰 따옴표)

ex) "안녕하세요!", "hello world"

C언어에서는 문자와 문자열의 타입을 char형으로 해준다. ex) char c = 'A';

세부 내용

page | 6

#### <연습 문제>

1. 문자열을 입력 받아 3 글자씩 잘라 각각 세 라인에 출력하시오.

```
#include <stdio.h>

void main() {
    char a[10];
    int cn = 1, i=0;
    printf("문자열을 입력하세요 >> ");
    scanf_s("%s", a, 10);

    while (a[i]) {
        printf("%c", a[i]);
        if (cn % 3 == 0) {
            puts("");
        }
        cn++;
        i++;
    }
}
```

출력 -----

문자열을 입력하세요 >> 가나다라마바사아  
가나다  
라마바

연습 문제

## 강 의 계 획 서

아시아 직업교육 허브대학

2020 학년도 1학기	전공 컴퓨터정보공학과(사회맞춤형 지능형 컴퓨팅과정)	학부	컴퓨터공학부	
과 목 명	C에플리케이션구현(2016003-PE)			
강의실 과 강의시간	월:5(3-217),6(3-217),7(3-217),8(3-217)	학점	3	
교과분류	이론/실습	시수	4	
담당 교수	강환수 + 연구실 : 2호관-706 + 전 화 : 02-2610-1941 + E-MAIL : hskang@dongyang.ac.kr + 면담가능기간 : 월 11시~12시 화 14시~17시			
학과 교육목표				
과목 개요	본 과목은 프로그래밍 언어 중 가장 널리 사용되고 있는 C언어를 학습하는 과목으로 C++, JAVA 등과 같은 언어의 기반이 된다. 본 과목에서는 지난 학기에서 배운 시스템프로그래밍1에 이어 C언어의 기본 구조 및 문법 체계 그리고 응용 프로그래밍 기법 등을 다룬다.  C언어에 대한 학습은 Windows상에서 이루어지며, 기본적인 이론 설명 후 실습문제를 프로그래밍하며 숙지하는 형태로 수업이 진행된다.			
학습목표 및 성취수준	대학 교육목표와 학과 교육목표를 달성하기 위하여 이 과목을 수강함으로써 학습자는 C언어의 문법 전반과 응용 프로그램 기법을 알 수 있다. 직전 학기의 수강으로 인한 C언어의 기초부터 함수, 포인터 등의 내용 이해를 바탕으로하여 이번 학기에는 지난 학기 내용의 전체적인 복습과 함께 C언어 전체를 학습하고, 특히 응용 능력을 배양하여 프로그래밍으로 문제를 해결하는 능력을 익히게 된다.			
	도서명	저자	출판사	비고
주교재	Perfect C	강환수, 강환일, 이동규	인피니티북스	
수업시 사용도구	Visual C++			
평가방법	중간고사 30%, 기말고사 30%, 과제물 및 퀴즈 20%, 출석 20%			
수강안내	C 언어를 활용하여 응용프로그램을 구현할 수 있다.			
1 주차	[개강일(3/16)]			
학습주제	강의 소개 및 전 학기 강의 내용 복습: C언어 기초 및 조건문과 반복문 복습			
목표및 내용	C언어 기초 통합개발환경 테스트 기초적인 코드 실습			

## 강 의 계 획 서

아시아 직업교육 허브대학

미리읽어오기	교재 1~5장
과제,시험,기타	수업 중에 제시함
<b>2 주차</b>	<b>[2주]</b>
학습주제	C언어 기초 문법
목표및 내용	변수와 상수 연산자 l-value와 r-value
미리읽어오기	교재 1~5장
과제,시험,기타	수업 중에 제시함
<b>3 주차</b>	<b>[3주]</b>
학습주제	조건문
목표및 내용	6장 조건문 학습
미리읽어오기	교재 6장
과제,시험,기타	수업 중에 제시함
<b>4 주차</b>	<b>[4주]</b>
학습주제	반복문
목표및 내용	7장 반복문 학습
미리읽어오기	교재 7장
과제,시험,기타	수업 중에 제시함
<b>5 주차</b>	<b>[5주]</b>
학습주제	포인터
목표및 내용	8장 포인터 학습 단일포인터 다중포인터 여러가지 포인터
미리읽어오기	교재 8장
과제,시험,기타	수업 중에 제시함
<b>6 주차</b>	<b>[6주]</b>
학습주제	배열
목표및 내용	9장 배열
미리읽어오기	교재 9장
과제,시험,기타	수업 중에 제시함

## 강 의 계 획 서

아시아 직업교육 허브대학

<b>7 주차</b>	<b>[7주]</b>
학습주제	함수
목표및 내용	10장 함수
미리읽어오기	교재 10장
과제,시험,기타	수업 중에 제시함
<b>8 주차</b>	<b>[중간고사]</b>
학습주제	중간고사
목표및 내용	중간고사
미리읽어오기	
과제,시험,기타	
<b>9 주차</b>	<b>[9주]</b>
학습주제	문자열
목표및 내용	11장 문자열
미리읽어오기	교재 11장
과제,시험,기타	수업 중에 제시함
<b>10 주차</b>	<b>[10주]</b>
학습주제	변수 유효범위
목표및 내용	12장 변수 유효범위
미리읽어오기	교재 12장
과제,시험,기타	수업 중에 제시함
<b>11 주차</b>	<b>[11주]</b>
학습주제	구조체
목표및 내용	13장 구조체
미리읽어오기	교재 13장
과제,시험,기타	수업 중에 제시함
<b>12 주차</b>	<b>[12주]</b>
학습주제	함수와 포인터 활용
목표및 내용	14장 함수와 포인터활용
미리읽어오기	교재 14장
과제,시험,기타	수업 중에 제시함

## 강 의 계 획 서

아시아 직업교육 허브대학

<b>13 주차</b>	<b>[13주]</b>
학습주제	파일처리
목표및 내용	15장 파일처리
미리읽어오기	교재 15장
과제,시험,기타	수업 중에 제시함
<b>14 주차</b>	<b>[14주]</b>
학습주제	항상심화강좌(동적할당)
목표및 내용	16장 동적할당
미리읽어오기	교재 16장
과제,시험,기타	수업 중에 제시함
<b>15 주차</b>	<b>[기말고사]</b>
학습주제	기말고사
목표및 내용	기말고사
미리읽어오기	
과제,시험,기타	..
<b>수업지원 안내</b>	<p>장애학생을 위한 별도의 수강 지원을 받을 수 있습니다.</p> <p>언어가 문제가 되는 학생은 글로 된 과제 안내, 확대문자 시험지 제공 등의 지원을 드립니다.</p>

## 1주차

### 핵심내용

- 문자와 문자열
- 다양한 문자 입출력
- 문자열 관련 함수( strcmp(), strcpy() )

### 문자와 문자열

문자 : 영어의 알파벳, 한글의 한 글자 같이 하나의 글자로 되어있는 것 (작은따옴표)

ex) 'A', '가'

문자열 : 문자가 모여 있는 것 즉, 문자의 모임 (큰 따옴표)

ex) "안녕하세요!", "hello world"

C언어에서는 문자와 문자열의 타입을 char형으로 해준다. ex) char c = 'A';

cf) java는 문자열의 타입이 String형!

그 대신 문자열은 ch[]와 같이 문자배열을 만들어서 사용한다. ex) char c[] = "Hello world"

문자열은 문자열의 마지막을 의미하는 NULL문자 '\0'가 마지막에 저장되어야 한다. (시작~\0)

즉, 문자배열의 크기는 문자열+1 로 해줘야 한다는 의미이다.

만약 문자배열의 크기가 문자열을 넘어가는 큰 수로 지정했다면, 문자열과 '\0'문자가 넣고 남은 부분은 '\0'으로 채워진다.

#### • 배열선언시 초기화 방법

1. char ch[] = {'h', 'e', 'l', 'l', 'o', '\0'};
2. char ch[] = "hello"; //상수인 리터널 문자열인 경우 알아서 '\0'을 가져감
3. char ch[6] = "hello";

#### • printf()를 이용한 문자,문자열 출력

char ch = 'a'; //문자선언

char c[] = "hello"; //문자열 선언

printf("%s %c", c, ch); //문자열 출력 (형식제어문자 %s(문자열일경우), %c(문자일 경우)를 이용)

----- 출력 -----

hello a

#### • 문자열 상수를 문자 포인터에 저장하는 방식 (문자를 수정할 수 없음)

int i = 0;

char \*java = "java";

printf("%s ", java); //출력방법 1

i = 0;

while (java[i]) //출력방법 2

```
printf("%c", java[i++]);
printf(" ");
```

----- 출력 -----

java java

- 'W0'문자로 문자열 분리

```
char c[] = "C++ Python";
c[3] = 'W0'; //c안에는 C++W0PythonW0으로 넣어져 있음
printf("%sWn%sWn", c, (c+4));
```

----- 출력 -----

C++  
Python

## 다양한 문자 입출력

getchar()	문자의 입력에 사용 라인버퍼링(line buffering) 방식을 사용하여 입력을 다하고 enter키를 눌러야 입력이 실행됨 즉각적인 입력요구 시스템에 맞지 않음
putchar()	문자의 출력에 사용 getchar()와 같은 방식으로 실행됨.
getche() _getche()	버퍼를 사용하지 않고 문자를 입력하는 함수 문자 하나를 입력하면 바로 실행됨. => 연속해서 두 번 나옴 <conio.h>를 삽입해야함 입력한 문자는 수정할 수 없음
getch() _getch()	버퍼를 사용하지 않고 문자를 입력하는 함수 입력한 문자가 화면에 보이지 않음. <conio.h>를 삽입해야함 입력한 문자는 수정할 수 없음
scanf("%s", str) scanf("%c", &ch)	문자 입력에 사용, 버퍼를 이용 enter키를 눌러야 실행 누르면 바로 표시 가능 입력 수정 가능 문자열 입력의 경우 충분한 공간의 문자 배열이 있어야 함
printf()	문자 출력에 사용
get(char *buffer) get_s(char *buffer, size_f sizebuffer)	문자열을 입력 받아 buffer에 저장하고 입력 받은 첫 문자의 주소값 반환 (라인버퍼링 방식 사용) 마지막에 입력된 'Wn'이 'W0'으로 교체되어 인자인 배열에 저장 처리속도가 빠름
puts(const char * str)	마지막 'W0'문자를 개행 문자인 'Wn'으로 대체해 출력 일반적으로 정수값 0반환



	오류나면 EOF(파일의 끝 의미:-1)반환 처리속도가 빠름
--	-------------------------------------

\* 라인버퍼링(line buffering) 방식 : 입력된 문자가 임시 저장소인 버퍼에 저장되었다가 enter키를 만나면 버퍼에서 문자를 읽는 방식

## 문자열 관련 함수 [strxxx()]

문자 배열 관련 함수들은 <stdio.h>에 정의되어 있다.

함수원형	설명
<code>void *memchr(const void *str, int c, size_t n)</code>	메모리 str에서 n 바이트까지 문자 c를 찾아 그 위치를 반환
<code>int memcmp(const void *str1, const void *str2, size_t n)</code>	메모리 str1과 str2를 첫 n 바이트를 비교 검색하여 같으면 0, 다르면 음수 또는 양수 반환
<code>void *memcpy(void *dest, const void *src, size_t n)</code>	포인터 src 위치에서 dest에 n 바이트를 복사한 후 dest 위치 반환
<code>void *memmove(void *dest, const void *src, size_t n)</code>	포인터 src 위치에서 dest에 n 바이트를 복사한 후 dest 위치 반환
<code>void *memset(void *str, int c, size_t n)</code>	포인터 str 위치에서부터 n 바이트까지 문자 c를 지정한 후 str 위치 반환
<code>size_t strlen(const char *str)</code>	포인터 str 위치에서부터 널 문자를 제외한 문자열의 길이 반환

\* Perfect C의 p.509 참고

다양한 문자열 처리에 관련한 함수들은 <string.h>에 정의되어 있다.

### - strcmp() : 두 문자열을 비교하는 함수

`int strcmp(const char *s1, const char *s2);`

⇒ 같은 위치의 문자를 앞부터 비교 (아스키코드값으로)

⇒ 앞이 크면 양수, 같으면 0, 뒤가 크면 음수

`Int strncmp(const char *s1, const char *s2, size_t maxn);`

⇒ 최대 n까지만 비교하여 결과를 나타냄

Ex) `strcmp("naver", "naveR");` //대문자가 소문자보다 값이 작음 => 양수

### - strcpy() : 문자열을 복사하는 함수

`char *strcpy(char *dest, const char *source);`

⇒ dest에 null문자를 포함한 source를 복사하여 그 복사된 문자열 반환

⇒ dest는 수정되지만, source는 수정되지 않음

`char *strncpy(char *dest, const char *source, size_t maxn);`

⇒ n개의 문자를 복사하여 복사된 문자열 반환

`errno_t strcpy_s(char *dest, size_t sizedest, const char *source);`

`errno_t strncpy_s(char *dest, size_t sizedest, const char *source, size_t maxn);`

⇒ 여기서 sizedest는 dest의 크기를 입력한다.

⇒ Errno\_t는 정수형이고 반환값은 오류번호로 성공하면 0으로 반환

Ex) `char dest[80] = "java"; char source[80] = "c is language";  
printf("%s %n", strcpy(dest, source));` //결과 : c is language

## &lt;연습 문제&gt;

1. 문자열을 입력 받아 3 글자씩 잘라 각각 새 라인에 출력하시오.

```
#include <stdio.h>

void main() {
    char a[10];
    int cn = 1, i=0;
    printf("문자열을 입력하세요 >> ");
    scanf_s("%s", a, 10);

    while (a[i]) {
        printf("%c", a[i]);
        if (cn % 3 == 0) {
            puts("");
        }
        cn++;
        i++;
    }
}
```

----- 출력 -----

문자열을 입력하세요 >> 가나다라마바사아  
가나다  
라마바  
사아

2. 입력 받은 문자열 중 알파벳 아닌 것 개수 구하기.

```
#include <stdio.h>

void main() {
    char a[20];
    int count = 0;
    printf("문자열을 입력하세요\n");
    scanf_s("%s", a, 20);
    while(a[i]) {
        if ((a[i] >= 'A' && a[i] <= 'Z') || (a[i] >= 'a' && a[i] <= 'z')) {
            i++;
            continue;}
        else {
            i++;
            count++;}
    }
    printf("문자열 중 알파벳 아닌 것 개수 : %d\n", count); }
```

문자열을 입력하세요  
Hello안녕하세요  
문자열 중 알파벳이 아닌 것 개수 : 5

## 2주차

### 핵심내용

- 문자열 관련함수
- 여러 개의 문자열 처리
- 명령행 인자
- 전역변수와 지역변수
- 정적 변수와 레지스터 변수

### 문자열 관련함수

- **strcat()** : 하나의 문자열 뒤에 다른 하나의 문자열(null문자까지)을 연결할 때 사용

`char * strcat(char * dest, const char * source);`

⇒ dest에 source를 연결해 저장하며, 연결된 문자열을 반환하고 source는 수정할 수 없음

`char * strncat(char * dest, const char * source, size_t maxn);`

⇒ n개 크기만큼 연결하여 저장함.

⇒ maxn이 문자열 길이보다 크면 null 문자까지 연결

`errno_t strcat_s(char * dest, size_t sizedest, const char * source);`

`errno_t strncat_s(char * dest, size_t sizedest, const char * source, size_t maxn);`

⇒ sizedest는 dest의 크기의미

ex) `char a[] = "java";`

`char s[] = " is good!";`

`printf("%s\\n", strcat(a, s));` // 결과값 : java is good!

- **strtok()** : 문자열에서 구분자인 문자를 여러 개 지정하여 토큰을 추출함

`char * strtok(char * str, const char * delim);`

⇒ str에서 delim를 구성하는 구분자를 기준으로 순서대로 토큰을 추출하여 반환함

⇒ delim은 수정될 수 없음

`char * strtok_s(char * str, const char * delim, char ** context);`

⇒ context는 함수 호출에 사용되는 위치정보를 위한 인자

ex) `char str1[] = "C and C++";` //문자열

`char* delimiter = " ";` //구분자

`char* ptoken = strtok(str1, delimiter);` //첫 토큰을 추출

`while (ptoken != NULL) {` //ptoken이 NULL이면 더 이상 분리할 토큰이 없음.

`printf("%s\\t", ptoken);`

`ptoken = strtok(NULL, delimiter);` //다음 토큰을 반환

`}` // 결과 값 : C          and          C++

- **strlen()** : NULL문자를 제외한 문자열 길이 반환
- **strlwr()** : 모두 소문자로 변환하여 반환
- **strupr()** : 모두 대문자로 변환하여 반환

ex) char str[] = "JAVA\_HOME";  
 printf("%d, ", strlen("java")); //java의 길이 : 4  
 printf("%s, ", \_strlwr(str)); //모두 소문자로 변환  
 printf("%s\n", \_strupr(str)); //모두 대문자로 변환  
 //결과 값 : 4, java\_home, JAVA\_HOME

#### - 문자와 관련된 여러 함수 요약표

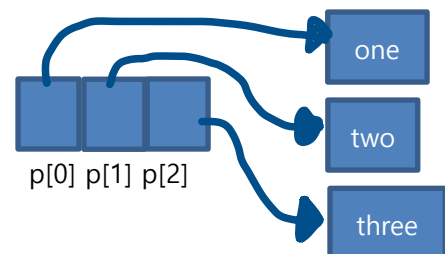
함수원형	설명
char * strlwr(char * str); errno_t _strlwr_s(char * str, size_t strsize); //Visual C++ 권장함수	문자열 str을 모두 소문자로 변환하고 변환한 문자열을 반환하므로 str은 상수이면 오류가 발생하며, errno_t는 정수형의 오류번호이며, size_t도 정수형으로 strsize는 str의 길이
char * strupr(char * str); errno_t _strupr_s(char * str, size_t strsize); //Visual C++ 권장함수	문자열 str을 모두 대문자로 변환하고 변환한 문자열을 반환하므로 str은 상수이면 오류가 발생하며, errno_t는 정수형의 오류번호이며, size_t도 정수형으로 strsize는 str의 길이
char * strpbrk(const char * str, const char * charset);	앞의 문자열 str에서 뒤 문자열 charset에 포함된 문자가 나타나는 처음 위치를 찾아 그 주소값을 반환하며, 만일 찾지 못하면 NULL 포인터를 반환
char * strstr(const char * str, const char * strsearch);	앞의 문자열 str에서 뒤 문자열 strsearch이 나타나는 처음 위치를 찾아 그 주소값을 반환하며, 만일 찾지 못하면 NULL 포인터를 반환
char * strchr(const char * str, char ch);	앞의 문자열 str에서 뒤 문자 ch가 나타나는 처음 위치를 찾아 그 주소값을 반환하며, 만일 찾지 못하면 NULL 포인터를 반환

\* Perfect C의 p.519 참고

### 여러 개의 문자열 처리

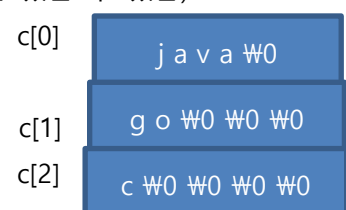
1. 문자 포인터 배열 (문자열 상수의 수정은 불가능)

```
char *p[] = {"one", "two", "three"};
printf("%s ", p[0]); printf("%s ", p[1]); printf("%s ", p[2]);
----- 출력 -----
one two three
```



2. 이차원 문자 배열 (문자열 수정가능 but 낭비되는 메모리공간 있을 수 있음)

```
char c[][5] = { "java", "go", "c" };
printf("%s ", c[0]); printf("%s ", c[1]); printf("%s ", c[2]);
----- 출력 -----
Java go c
```



## 명령행 인자

도스 창에서 프로그램이 저장된 폴더의 경로와 >를 합쳐 프롬프트라 하고, 명령어 프롬프트가 나타나는 이 줄에 명령어를 입력할 수 있으며 이 줄을 명령행이라 한다.

명령행에서 입력하는 문자열을 프로그램으로 전달하는 방법

```
main(int argc, char *argv[]);
```

```
// argc : 명령행에서 입력한 문자열의 수
```

```
// argv[] : 명령행에서 입력한 문자열을 전달 받는 문자 포인터 배열
```

Ex) 명령행에 실행파일 이름이 com이고 옵션으로 c c++

```
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;
}
```

----- 출력 -----

실행 명령행 인자(command line arguments) >>

argc = 3

argv[0] = com

argv[1] = c

argv[2] = c++

## 전역변수와 지역변수

\* 변수의 참조가 유효한 범위 : 변수의 유효 범위(scope)

- 지역 유효 범위(local scope) : 함수 or 블록 내부에서 선언

- 전역 유효 범위(global scope)

# 하나의 파일에서만 변수 참조가능

# 프로젝트를 구성하는 모든 파일에서 변수 참조 가능

변수의 위치에 따라

지역 변수(내부변수, 자동변수)	전역 변수(외부변수)
함수 or 블록 내부에서 선언	함수 외부에서 선언되는 변수
선언된 함수 or 블록에서 사용 가능	프로젝트의 모든 함수나 블록에서 참조가능
선언 후 초기화를 하지 않으면 쓰레기값 저장	선언 후 자동으로 초기값, 자료형에 맞게 저장

<p>함수 or 블록에서 선언 문장이 실행되는 시점에서 메모리 할당됨</p> <p>함수 or 블록에서 종료되는 순간 메모리에서 자동으로 제거</p> <p>자료형 앞에 auto를 사용가능(auto는 생략가능)</p>	<p>함수나 블록에서 전역변수와 같은 이름으로 지역변수 선언 가능 (함수 내부나 블록에서 이름을 참조하면 지역변수로 인식)</p> <p>프로젝트의 다른 파일에서도 참조 가능 (extern을 사용)</p>
---	---

- \* 지역변수가 할당되는 메모리 영역 : 스택(stack)
- \* extern은 변수선언 문장 맨 앞에 extern을 넣는 구조로 사용됨 (새로운 변수 선언이 아니라, 이미 존재하는 전역변수의 유효범위를 확장하는 것)
- \* extern을 동일 파일에서도 사용해야하는 경우 : 전역변수의 위치가 변수를 참조하려는 위치보다 뒤에 있는 경우 사용해야함
- \* 전역변수 장점 : 어디에서든지 수정가능함 => 사용하기 편함
- \* 전역변수 단점 : 예상치 못한 값이 저장되면 어디서 어느 부분에서 수정되었는지 알기 어렵

## 정적 변수와 레지스터 변수

기억분류에 따라 (할당되는 메모리 영역 결정되고 메모리 할당과 제거 시기 결정)

기억분류 종류	전역	지역
auto	×	○
register	×	○
static	○	○
extern	○	×

\* Perfect C의 p.556 참고

cf) 내가 생각하는 기억분류의 구분

auto : 지역변수에 사용

extern : 전역변수에 사용 but, 다른 파일에서 참조될 때 사용

register : 지역변수이지만, 처리속도가 빠르게 되어 할 때 사용

static : 값을 유지해야 할 때 사용

<사용 방법>

기억분류 자료형 변수이름;

기억분류 자료형 변수이름 = 초기값; (extern을 제외한 3개의 기억분류 초기값 저장가능)

## 레지스터 변수

변수의 저장공간이 일반 메모리가 아니라 CPU내부의 레지스터에 할당되는 변수

지역변수에만 이용 가능 (함수나 블록이 시작되면서 레지스터에 값 저장, 끝나면 소멸되는 특성)

CPU내부의 레지스터라서 일반 메모리보다 빠르게 참조가능 (처리속도 ↑)

주소연산자 &를 사용하지 못함 (일반 메모리에 할당되는 변수가 아니기 때문)

레지스터는 한정되어 있으므로 레지스터가 모자라면 일반 지역변수로 할당됨

## 정적 변수

초기 생성이후 메모리에서 제거되지 않으므로 지속적으로 저장 값을 유지하거나 수정할 수 있음.  
프로그램 시작되면 메모리 할당하고, 종료되면 메모리에서 제거

초기값 지정하지 않으면 자동으로 자료형에 맞춰 저장

초기화는 상수로만 가능하며, 단 한번만 수행 (초기화가 된 정적변수는 더 이상 초기화되지 않는 특성)

### - 정적 지역변수

참조 범위 : 지역변수

변수의 할당과 제거 : 전역변수 특징 (프로그램이 끝나야 제거)

⇒ 특징 : 함수나 블록을 종료해도 메모리에서 제거되지 않고 계속 메모리에 유지관리됨 (즉, 이전에 호출되어 저장된 값을 유지하여 이번 호출에도 사용가능)

```
Ex) int main(void) {
    for (int count = 0; count < 3; count++)
        inc();
}

void inc(void) {
    static int sindex = 1; //정적 지역변수

    printf("정적 지역변수 sindex : %d\n", sindex++);
}
```

출력

정적 지역변수 sindex : 1

정적 지역변수 sindex : 2

정적 지역변수 sindex : 3

### - 정적 전역변수

참조범위 : 선언된 파일에만 한정 (extern에 의해 다른 파일에서 참조 불가능)

변수의 할당과 제거 : 전역변수 특징

<총 정리>

선언위치	상세 종류	키워드		유효범위	기억장소	생존기간
전역	전역 변수	참조선언	extern	프로그램 전역	메모리 (데이터 영역)	프로그램 실행 시간
	정적 전역변수	static		파일 내부		
지역	정적 지역변수	static		함수나 블록 내부	레지스터	함수 또는 블록 실행 시간
	레지스터 변수	register			메모리 (스택 영역)	
	자동 지역변수	auto (생략가능)				

\* Perfect C의 p.568 참고

## &lt;연습 문제&gt;

1. 입력 받은 문자열 중 맨 가운데 문자를 출력하시오.(길이가 짝수면 2 개 출력)

```
#include <stdio.h>

void main() {
    char a[10];
    printf("문자열을 입력하세요 >>");
    scanf_s("%s", a, 10);

    int len = strlen(a) / 2;
    if (strlen(a) % 2 == 0) {
        printf("%c %c\n", a[len-1], a[len]);
    }
    else {
        printf("%c\n", a[len]);
    }
}
```

----- 출력 -----  
 문자열을 입력하세요 >> c 언어 프로그래밍  
 프

2. 전역변수를 하나 선언하고, main 함수에서 그 전역변수에 숫자를 입력 받은 후, 1 에서 이 수까지의 합을 함수 호출을 통하여 구하시오.

(단, 합 구하는 함수는 인자가 없으며, 다른 파일에 있게 할 것; int Gsum(); )

```
#include <stdio.h>

int a;
int Gsum();
void main() {
    printf("숫자를 입력하세요\n");
    scanf_s("%d", &a);
    printf("총합 : %d", Gsum());
}
```

숫자를 입력하세요

4

10

```
-----
extern int a;
int Gsum() {
    int sum = 0;
    for (int i = 0; i <= a; i++) {
        sum += i;
    }
    return sum;
}
```

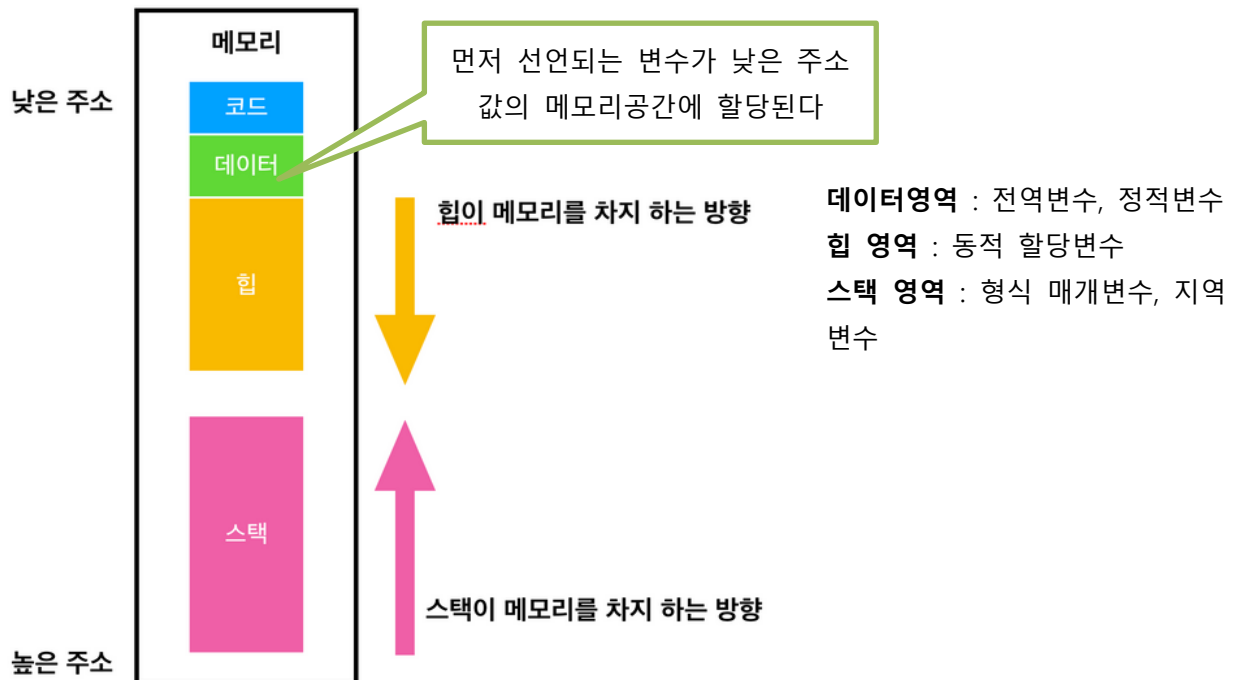


### 3주차

핵심 내용

- 메모리 영역과 변수 이용
- Visual studio 2019 사용법
- Ch.12 복습

#### 메모리 영역 (변수의 유효범위와 생존기간에 결정적 역할)



#### 변수의 이용

일반적으로 전역변수의 사용을 자제, 지역변수를 주로 이용

- 1) 실행속도를 개선하고자 할 때 => 레지스터 변수 이용
- 2) 함수나 블록내부에서 함수나 블록 종료되더라도 계속 값 저장하고 싶을 때  
=> 정적 지역변수 이용
- 3) 해당 파일 내부에서만 변수 공유하고 싶을 때 => 정적 전역변수 이용
- 4) 프로그램 모든 영역에서 값 공유하고 싶을 때 => 전역변수 이용

#### Visual studio 2019 사용법

솔루션 > 프로젝트 > 소스파일(c 파일)

실행하기 위해서 '디버그하지 않고 시작하기' 를 누르면 링크와 실행이 한번에 사용됨

'빌드 > 컴파일'을 사용하여 컴파일을 할 수 있음 (빌드 성공이 나와야 컴파일성공한 것임)

프로젝트 하부에 debug를 누르면 오브젝트 파일이 있음.

실행파일은 소스파일의 이름으로 만들어지는 것이 아니라 프로젝트의 이름으로 실행파일이 만들어짐. (솔루션 하부의 debug에 들어가면 실행파일이 있음)

솔루션 아래의 프로젝트가 두 개 일 경우, 실행을 누르면 원래의 프로젝트가 실행이 됨.

새로운 프로젝트를 사용하기 위해서는 '시작 프로젝트로 설정'을 눌러서 활성화되게 만듦.

## 복습

이 부분을 잘 알아 두면 될 것 같음

### <변수의 종류>

지역, 전역	종류	키워드		기억장소	생존기간	유효범위
전역	전역변수	정의 시	일반선언	메모리  (데이터 영역)	영구적  (프로그램 종료 시 제거)	프로그램 전역
		참조선언 시	extern			정의된 파일 내부
	정적 전역변수	static				
지역	정적 지역변수	static		레지스터  메모리(스택 영역)	일시적(함수 블록 종료 시 제거)	정의된 함수나 블록 내부
	레지스터 변수	register				
	자동 지역변수	auto(생략가능)				

### <변수의 유효범위>

지역, 전역	종류	프로그램 시작	다른 함수에서의 이용(같은 파일)	함수(블록) 시작	함수(블록) 종료	다른 함수에서의 이용(다른 파일)	프로그램 종료
전역	전역변수	생성	O	O	O	O	제거
	정적 전역변수	생성	O	O	O	X	제거
지역	정적 지역변수	생성	X	O	O	X	제거
	레지스터 변수		X	생성	제거	X	
	자동 지역변수		X	생성	제거	X	

### <변수의 초기값>

지역, 전역	종류	초기값을 부여하지 않은 경우	초기값 저장
전역	전역변수	자료형에 따라 0이나 '\0' 또는 NULL 값이 저장됨.	프로그램 시작 시
	정적 전역변수		
지역	정적 지역변수		
	레지스터 변수	쓰레기값이 저장됨.	함수나 블록이 실행될 때마다
	자동 지역변수		

\* Perfect C의 p.568~p.569 참고

## &lt;연습문제&gt;

1. 변수가 메모리에서 제거되는 시점이 다른 것은? 답 : 4

① 전역변수 ② 정적 지역변수 ③ 정적 전역변수 ④ 자동 지역변수

2. 변수의 종류에 맞추어서 사용하는 키워드를 작성하라

① 전역변수 (참조선언 시)

② 정적 지역변수

③ 레지스터 변수

④ 자동 지역변수

답 : 1) extern 2) static 3) register 4) auto

3. 전역변수를 하나 선언하고 그 전역변수 값만큼 "Hello"를 여러 번 출력하는 함수를 작성하시오.

- 전역변수의 값 입력은 main()에서 함

```
#include <stdio.h>
```

```
int cnt;
```

```
void prn_hello();
```

```
int main() {
    printf("값을 입력하시오 %n");
    scanf_s("%d" &cnt);
    prn_hello();
}
```

```
-----
#include <stdio.h>
```

```
extern int cnt;
```

```
void prn_hello(){
    int i;
    for( i = 0; i < cnt; i++)
        puts("Hello");
}
```

```
----- 출력 -----
```

```
값을 입력하시오
```

```
5
```

```
Hello
```

```
Hello
```

```
Hello
```

```
Hello
```

```
Hello
```

## 4 주차

### 핵심 내용

- 구조체와 공용체
- 자료형 재정의

### 구조체와 공용체

구조체 : 연관성이 있는 서로 다른 개별적인 자료형의 변수들을 하나의 단위로 묶은 새로운 자료형 (정수나 문자, 실수나 포인터 그리고 이들의 배열 등을 묶어 하나의 자료형으로 이용)

- 유도 자료형 : 기존 자료형으로 새로이 만들어진 자료형

Ex) 학생정보 ( 이름, 학과, 성적, 학번, 주소 )

웹툰정보 ( 이름, 작가, 요일, 새로 올라온 날짜 )

- 1) 구조체 정의 (변수 선언에서 이용될 새로운 구조체 자료형을 정의) (유효범위 : 전역, 지역)  
cf) java에서 객체 생성하기 위해 class라는 틀을 만들듯이!

```
struct lecture { //하나의 자료형으로 쓰임
```

```
    char name[20]; //반드시 ;으로 종료, 초기값은 대입할 수 없음
```

```
    int credit;
```

```
    int hour; //구조체 멤버나 필드, 여기서 선언되는 이름은 모두 유일해야 함
```

```
}; //멤버 : 일반변수, 포인터변수, 배열, 다른 구조체 변수 및 구조체 포인터
```

- 2) 변수 선언

```
struct lecture data;
```

```
struct lecture data1, data2, data3;
```

- 구조체 정의와 변수 선언 동시에 가능 (권장안함)

```
struct account {
```

```
    char name[20];
```

```
    int actnum;
```

```
    double balance;
```

```
} myaccount; //구조체 태그 없는 구조체 변수 가능 (동일한 자료형 변수 선언 불가능)
```

- 초기화

```
struct lecture c = { "C 프로그래밍", "3", "4" }; //멤버의 순서대로 선언해줘야 함.
```

- 구조체형 변수에서 멤버 접근 (접근연산자 .)

```
data.name = "데이터베이스"; data.credit = "3"; // 구조체변수이름.멤버 으로 참조 가능
```

- 실제 구조체의 크기 >= 멤버의 크기 합

- 구조체는 다른 구조체의 멤버로 사용가능

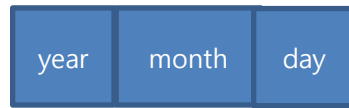
```
struct date {
```

```
    int year;
```

```
    int month;
```

```
    int day;
```

```
};
```



data라는 구조체

```
struct account {
```

```
    struct date open; // 구조체
```

```
    char name[12];
```

```
    int actnum;
```

```
    double balance;
```

```
};
```



account라는 구조체

```
struct account me = { {2018,3,9}, "홍길동", 1001, 300000 }; //변수 선언과 동시에 초기화
```

```
printf("[%d, %d, %d]\n", me.open.year, me.open.month, me.open.day); //접근연산자를 2번 사용
```

```
printf("%s %d %.2f\n", me.name, me.actnum, me.balance);
```

```
----- 출력 -----
```

```
[2018, 3, 9]
```

```
홍길동 1001 300000.00
```

- 동일한 구조체형의 변수는 대입문이 가능 (멤버마다 하나하나 대입할 필요 없이 한번에 가능)

```
struct date {
```

```
    int year;
```

```
    int month;
```

```
    int day;
```

```
};
```

```
struct date me = { 2020, 5, 25 };
```

```
struct date you;
```

```
you = me; //가능 you에 me의 값이 한번에 넣어짐
```

```
// if( you == me ) //오류 -> if( you.year == me.year )처럼 멤버 하나하나 비교해야 함.
```

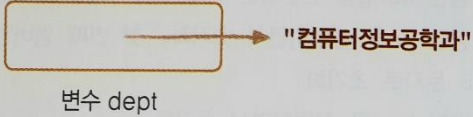
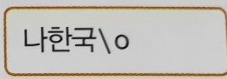
cf ) 포인터 char \*와 char []

char \* : 문자열 상수 주소 (문자열의 첫 문자 주소 저장)

char [] : 문자 하나 하나를 저장하고 마지막에 \0을 저장하여 사용

=> 상수 문자열 처리라면 포인터로 사용하고, 저장과 수정을 한다면 배열로 사용하는 편이 좋음

## &lt;비교표&gt;

char 포인터	char 배열
<code>char *dept; //학과 이름</code>	<code>char name[12]; //학생 이름</code>
<code>char *dept = "컴퓨터정보공학과";</code>	<code>char name[12] = "나한국";</code>
 <p>변수 dept</p>	 <p>변수 name[12]</p>
변수 dept는 포인터로 단순히 문자열 상수를 다루는 경우 효과적	변수 name은 배열로 12바이트 공간을 가지며 문자열을 저장하고 수정 등이 필요한 경우 효과적
<code>dept = "컴퓨터정보공학과";</code>	<code>name = "나한국"; //오류</code>
단지 문자열 상수의 첫 주소를 저장하므로 문자열 자체를 저장하거나 수정하는 것은 불가능하므로 다음 구문은 사용 불가능	문자열 자체를 저장하는 배열이므로 문자열의 저장 및 수정이 가능하고 문자열 자체를 저장하는 다음 구문 사용도 가능
<code>strcpy(dept, "컴퓨터정보공학과"); //오류</code>	<code>strcpy(name, "배상문");</code>
<code>scanf("%s", dept); //오류</code>	<code>scanf("%s", name);</code>

\* Perfect C의 p.592 참고

공용체 : 동일한 저장 장소에 여러 자료형을 저장하는 방법으로 마지막 자료형만 저장하고 참조 가능 (멤버에 한번에 한 종류만 저장가능)

cf) 공용체는 공용 차고지라 하면 여러 자료형은 여러가지 차라고 볼 수 있음.

- 공용체 선언 방법은 구조체 선언 방법과 동일 (struct -> union으로 바꿔 선언해줘야 함)
- 공용체의 변수의 크기는 멤버 중 가장 큰 자료형의 크기로 정해짐
- 공용체의 초기화 값은 공용체 정의 시 처음 선언한 멤버의 초기값으로만 저장 가능
- 공용체 변수로 멤버를 접근하기 위해 접근연산자 . 를 사용함

## 자료형 재정의

typedef : 이미 사용되는 자료유형을 다른 새로운 자료형 이름으로 재정의할 수 있도록 하는 키워드

자료형을 재정의하는 이유? 프로그램의 시스템 간 호환성과 편의성을 위해 필요함

Ex) visual C++에서의 int형은 4바이트

터보 C++에서의 int형은 2바이트

이 때 visual C++에서 작성한 프로그램이 터보 C++로 가서 실행하면 오류가 생김

⇒ 해결방안 : int형을 재정의하여 작성한 프로그램을 터보 C++로 가서 재정의한 문장에서 int형을 long형으로 바꿔서 사용하면 다른 소스 수정 필요없이 실행이 됨

## &lt;정의 방법&gt;

typedef 기존자료유형이름 새로운자료형1, 새로운자료형2 ... ;

Ex) typedef int myint;

- 구조체 정의할 때 'struct 변수이름'을 간단히 typedef으로 재정의할 수 있음

1) struct date {

int year;

int month;

int day;

};

typedef struct date date; //struct date라는 자료형이 data라는 자료형으로 재정의

2) typedef struct {

char title[30];

int date;

} web; // web은 변수가 아니라 구조체의 새로운 자료형임

web i = {"전자적독자시점", "20200527"}; // web자료형의 변수 i의 초기화

## &lt;연습문제&gt;

다음 내용을 참고로 구조체 struct food를 정의하고, 음식 2개를 선언하여 적당한 값을 초기화로 입력하고 출력하는 프로그램 작성하시오.

- 구조체 struct food 멤버 구성 : 음식이름, 가격, 수량, 유통기한
- 유통기한은 struct date로 구성 (멤버 구성 : 년, 월, 일)
- typedef로 자료형을 재정의 (struct date, struct food)

```
#include <stdio.h>
```

```
struct date {
```

```
    int year; //년
```

```
    int month; //월
```

```
    int day; //일
```

```
};
```

```
typedef struct date date;
```

```
int main(void) {
```

```
    struct food {
```

```
        char fname[30]; //음식이름
```

```
        int price; //가격
```

```
        int count; //수량
```

```
        date slife; //유통기한
```

```
    };
```

```
    typedef struct food food;
```

```
    food snd = { "샌드위치", 2400, 3, {2020,6,22} };
```

```
    food kim = { "김밥", 2200, 1, {2020,5,31} };
```

```
    printf("%10s %3s %s %7s\n", "음식이름", "가격", "수량", "유통기한" );
```

```
    printf("%10s %5d %5d ", snd.fname, snd.price, snd.count );
```

```
    printf(" %d. %d. %d\n", snd.slife.year, snd.slife.month, snd.slife.day);
```

```
    printf("%10s %5d %5d ", kim.fname, kim.price, kim.count );
```

```
    printf("%d. %d. %d\n", kim.slife.year, kim.slife.month, kim.slife.day);
```

```
    return 0;
```

```
}
```

```
----- 출력 -----
```

```
음식이름  가격  수량  유통기한
```

```
샌드위치  2400   3   2020. 6. 22
```

```
김밥      2200   1   2020. 5. 31
```



## 5주차

### 핵심 내용

- 구조체 포인터, 배열, 공용체 포인터
- 함수의 인자전달 방식
- 주소연산자 &

### 구조체 포인터 (구조체의 주소값 저장)

변수 선언은 일반 포인터 변수 선언과 동일 (사용 방법도 동일)

```
struct lecture {
    char name[20];
    int type;
    int credit;
    int hours;
};
typedef struct lecture lecture;
lecture os = {"운영체제", 2, 3, 4};
lecture *p = &os; //주소값 저장
```

- 포인터 변수의 구조체 멤버 접근 연산자 '->' (두 문자가 연결된 하나의 연산식, 사이에 공백 X)

Ex) p->type = os.type = (\*p).type //포인터 p가 가리키는 구조체 멤버 type

\*p.name // \*(p.name)이라는 의미, 여기서 p는 포인터이므로 오류

\*os.name // \*(os.name)이라는 의미, 이 경우 구조체 변수 os 멤버 name의 첫 문자임  
한글일 경우 실행 오류가 생김

\*p->name // \*(p->name)이라는 의미, 이 경우 구조체 포인터 p가 가리키는 구조체의 멤버  
name의 첫 문자임. 한글일 경우 실행 오류가 생김

Cf) 연산자 ->와 .은 우선순위 1위, 결합성은 좌에서 우

연산자 \*은 우선순위 2위, 결합성은 우에서 좌

### 공용체 포인터

공용체 포인터 변수로 멤버 접근하려면 접근연산자 -> 이용

```
union data{
    char ch;
    int cnt;
    double real;
} value, *p; //변수 value는 union data형, p는 union data포인터 형으로 선언
p = &value; //포인터 p에 value의 주소값 저장
p->ch = 'a'; //value.ch = 'a';와 동일한 문장
```

## 구조체 배열

동일한 구조체 변수가 여러 개 필요하면 구조체배열을 선언하여 이용 가능

lecture c[] = { //lecture라는 구조체정의를 내린 후 typedef를 이용하여 자료형 재정의 함.

```
    {"인간과 사회",2,2},
```

```
    {"경제학개론",3,3},
```

```
    {"자료구조",3,3}
```

```
};
```

```
int arysize = sizeof(c) / sizeof(lecture); //배열크기
```

```
lecture *p = c; //구조체배열이름은 구조체 포인터변수에 대입가능
```

```
for ( i = 0; i < arysize; i++)
```

```
    printf("%16s %5d %5d\n", c[i].name, c[i].credit, c[i].hours);
```

----- 출력 -----

```
인간과 사회      2      2
```

```
경제학개론      3      3
```

```
자료구조       3      3
```

## 함수의 인자전달 방식

1) 값에 의한 호출 (call by value) (함수에서 값의 전달)

함수 호출 시 실인자의 값이 형식 인자에 복사되어 저장된다는 의미

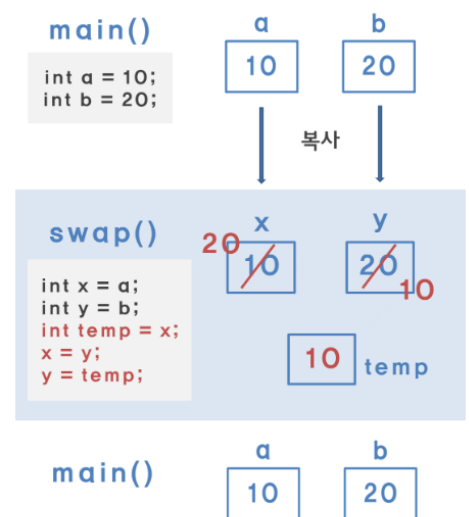
c언어의 함수인자 전달 방식 (함수 외부의 변수를 함수 내부에서 수정할 수 없다는 특징)

```
#include <stdio.h>

void swap(int x, int y);

void main() {
    int a = 10;
    int b = 20;
    printf("before swap() : a = %d, b = %d\n", a, b);
    swap(a, b);
    printf("after swap() : a = %d, b = %d\n", a, b);
}

void swap(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
}
```



----- 출력 -----

```
before swap() : a = 10, b = 20
```

```
after swap() : a = 10, b = 20
```

<https://medium.com/pocs/%ED%95%A8%EC%88%98%EC%9D%98-%EC%9D%B8%EC%9E%90-%EC%A0%84%EB%8B%AC-%EB%B0%A9%EC%8B%9D-7070cdf38645>

## 2) 참조에 의한 호출 (call by reference) (함수에서 주소의 전달)

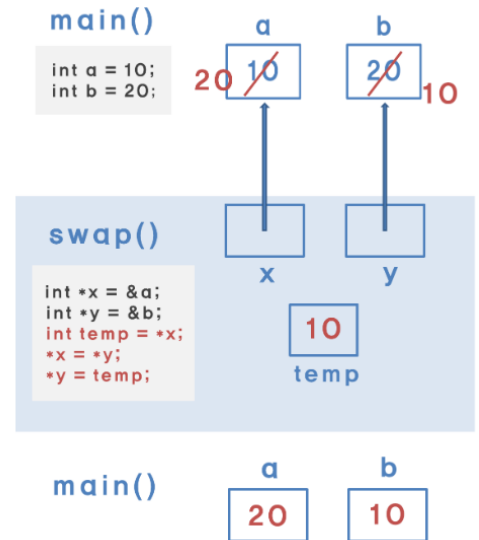
포인터를 매개변수로 사용하면 함수로 전달된 실인자의 주소를 이용하여 그 변수를 참조 가능

```
#include <stdio.h>

void swap(int *x, int *y);

void main() {
    int a = 10;
    int b = 20;
    printf("before swap() : a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("after swap() : a = %d, b = %d\n", a, b);
}

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}
```



출력

before swap() : a = 10, b = 20

after swap() : a = 20, b = 10

<https://medium.com/pocs/%ED%95%A8%EC%88%98%EC%9D%98-%EC%9D%B8%EC%9E%90-%EC%A0%84%EB%8B%AC-%EB%B0%A9%EC%8B%9D-7070cdf38645>

## 배열전달

함수의 매개변수로 배열 전달의 의미는 배열의 첫 원소를 참조 매개변수로 전달한다는 의미

- 매개변수로 배열, 배열크기를 전달

- 함수 내부에서 실인자로 전달될 배열의 배열크기 알 수 없음 → 두번째 인자로 배열크기

```
double sum(double ary[], int n);
```

```
void main() {
```

```
    double data[] = { 2.3, 3.4, 2.6, 5.3 }; //배열
```

```
    printf( "합 : %5.1f\n", sum(data,4);
```

```
}
```

```
double sum(double ary[], int n) {
```

```
    double total = 0.0;
```

```
    for (int i = 0; i < n; i++)
```

```
        total += ary[i];
```

```
    return total;}
```

출력

합 : 13.6

### 다양한 배열원소 참조 방법

- 간접연산자를 사용한 배열원소 참조

```
int i, sum = 0;
int point[] = { 95, 34, 59, 29, 43, 23 };
int *address = point;
int aryLength = sizeof (point) / sizeof (int);

for (i=0; i<aryLength; i++){
    sum += *(point+i); // sum += *(address++);와 동일
    //sum += *(point++); 오류, point가 주소상수이기에 사용불가능
}
```

- 함수헤더에 배열을 인자로 기술하는 방법

```
int summary(int ary[] , int SIZE) { //int summary(int *ary, int SIZE)와 동일
    for ( int i = 0; i < SIZE; i++){
        sum += ary[i];
        //sum += *(ary + i);
        //sum += *ary++;
        //sum += (ary++); //모두 같은 뜻을 가진 코드
    }
}
```

### 배열 크기 계산 방법

sizeof(배열이름) / sizeof(배열원소) 의 결과가 배열 크기이다.

### 다차원 배열 전달

- 함수원형과 함수 정의의 헤더에서 첫 번째 대괄호 내부의 크기를 제외한 다른 모든 크기는 반드시 기술 ex) double sum(double data[][3], int, int);
- 매개변수 : 이차원배열, 행의 수, 열의 수
- 이차원 배열의 행의 수 : sizeof(x) / sizeof(x[0])
- 이차원 배열의 열의 수 : sizeof(x[0]) / sizeof(x[0][0])
- 이차원 배열의 전체 원소 수 : sizeof(x) / sizeof(x[0][0])

cf) sizeof(x) : 배열 전체의 바이트 수

sizeof(x[0]) : 1행의 바이트 수

sizeof(x[0][0]) : 첫 원소의 바이트 수

### 가변인자

함수에서 인자의 수와 자료형이 결정되지 않은 함수 인자 방식

- 가변 인자가 있는 함수머리

함수의 가변 인자는 처음 또는 앞 부분의 매개변수는 정해져 있으나, 이후 매개변수 수와 각각 자료형이 고정적이지 않고 변하는 인자 (중간이후부터 마지막에 위치)

가변인자 ... 시작 전 첫 고정 매개변수는 이후 가변인자를 처리하는데 필요한 정보를 지정하는데 사용

Int printf(const char \*\_Format, ...); // 함수 정의 시 가변인자의 매개변수는 ...로 기술

//함수 사용

printf("%d%d", 3, 4); //인자가 2개

printf("%d%d%f%f", 2, 6, 4.5, 6.3); //인자가 4개

- 가변인자가 있는 함수 구현 (<stdarg.h>가 필요)

#### 1) 가변인자 선언

가변인자로 처리할 변수 하나를 만드는 일

Ex) va\_list argp; //자료형 va\_list는 stdarg.h에 정의되어있음

#### 2) 가변인자 처리 시작

선언된 변수에서 마지막 고정 인자를 지정해 가변 인자의 시작 위치를 알리는 방법

Ex) va\_start(argp, mun); // va\_start(가변인자변수, 가변인자\_이전\_첫\_고정자)

#### 3) 가변인자 얻기

가변인자 각각의 자료형을 지정하여 가변인자를 반환하는 절차

Ex) total += va\_arg(argp, int); //va\_arg(가변인자변수, 반환될\_자료형)

#### 4) 가변인자 처리 종료

가변 인자에 대한 처리를 끝내는 단계

Ex) va\_end(argp);

## 주소연산자 & (참조에 의한 호출)

```
int m = 0, n = 0, sum = 0;
```

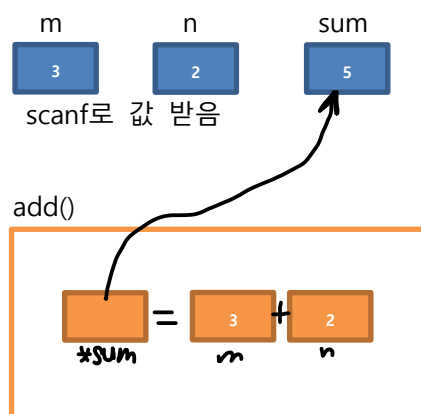
```
scanf("%d %d", &m, &n);
```

```
add(&sum, m, n);
```

```
void add(int *sum, int a, int b){
```

```
    *sum = a + b;
```

```
}
```



## &lt;연습 문제&gt;

1. 값에 대한 호출과 참조에 대한 호출을 설명하시오.

값에 대한 호출은 함수 호출 시 실인자의 값이 형식 인자에 복사되어 저장되어 사용하며, 실제 변수 안에 있는 값은 변동이 없다.

참조에 대한 호출은 포인터를 매개변수로 사용하여 함수로 전달된 실인자의 주소를 이용하여 그 변수를 참조함. 변수 안에 있는 값이 변동이 생긴다.

2. 다음 프로그램은 이차원 배열 값을 모두 더해주는 프로그램이다. 가로 안에 알맞은 코드를 쓰시오

- 출력 : 이차원 배열 합 : 78

```
#include <stdio.h>
```

```
int sum (double data[][3], int, int);
```

```
int main(void) {
```

```
    int ary[][3] = { { 1, 3, 5 }, { 2, 4, 6 }, { 7, 8, 9 }, {10, 11, 12} }; //4x3
```

```
    int row = sizeof(ary) / sizeof(ary[0]);
```

```
    int col = _____(1)_____;
```

```
    printf("이차원 배열 합 : %d", _____(2)_____);
```

```
    return 0;
```

```
}
```

```
int sum (double data[][3], int rowsize, int colsize) {
```

```
    int total = 0;
```

```
    for ( int i = 0; i < ____ (3) ____; i++)
```

```
        for( int j = 0; j < ____ (4) ____; j++)
```

```
            _____(5)_____;
```

```
    return 0;
```

```
}
```

답

(1) sizeof(ary[0]) / sizeof(ary[0][0])

(2) sum(ary, row, col)

(3) rowsize

(4) colsize

(5) total += data[i][j];

## 6주차

### 핵심 내용

- 키워드 const
- 복소수를 위한 구조체
- 함수 포인터와 함수 포인터 배열
- Void 포인터
- 파일 기초

### 주소값 반환 (함수의 결과를 포인터로 반환)

```
int * add(int *, int, int); //함수 원형
```

```
int m = 0, n = 0, sum = 0;
```

```
scanf("%d %d", &m, &n);
```

```
printf("두 정수 합 : %d\n", *add(&sum, m, n)); //함수 앞에 *을 쓴 이유 : 반환 값이 포인터라서
```

```
int * add(int *psum, int a, int b){
```

```
    *psum = a + b;
```

```
    return psum;
```

```
}
```

```
----- 출력 -----
```

```
3 7
```

```
두 정수 합 : 10
```

### 키워드 const

포인터를 매개변수로 사용하면 수정된 결과를 받을 수 있어 편하지만, 때때로 매개변수 값이 원하지 않는 방향으로 수정될 가능성이 있음. 이때 미리 예방하는 방법이 const를 사용하는 것임

- 인자인 포인터 변수가 가리키는 내용을 수정할 수 없게 함. (변수 -> 상수)
- const의 위치는 자료형 앞 or 포인터변수 앞

```
ex) void min(double *result, const double *a, double const *b){
```

```
    *result = *a + *b;
```

```
    //*a += 1 //오류
```

```
}
```

cf) const의 위치가 \*과 포인터 변수 사이 => 포인터변수 자체가 상수

ex) int\* const pi = &i; //포인터 pi에 저장되는 초기 주소 값을 더 이상 수정 불가능 (상수)

### 복소수를 위한 구조체

cf) 복소수 : 실수의 개념을 확장한 수 ex)  $a+bi$  //a는 실수부, b는 허수부

$a+bi$ 의 켤레 복소수 :  $a-bi$

$a-bi$ 의 켤레 복소수 :  $a+bi$

```

struct complex {
    double real; //실수
    double img; //허수
};

typedef struct complex complex; //구조체 정의

```

1) 인자와 반환형으로 구조체 사용 (값에 의한 호출 방식)

```

complex comp = { 2.4, 5.3 };
complex pcomp;
pcomp = paircomplex1(comp); //main
-----
complex paircomplex1(complex com) { //구조체 자체를 인자로 하는 함수
    com.img = - com.img;
    return com; //반환값도 구조체 자체
}

```

2) 참조에 의한 호출 방식

```

complex comp = { 2.4, 5.3 };
complex pcomp;
paircomplex2(&pcomp); //main
-----
void paircomplex2(complex *com){
    com->img = -com->img;
}

```

## 함수 포인터

동일한 구조의 함수의 주소 값을 저장하는 포인터 변수 (반환형,인자목록의 수,자료형이 일치)  
하나의 함수 이름으로 필요에 따라 여러 함수 사용하기 위해 사용

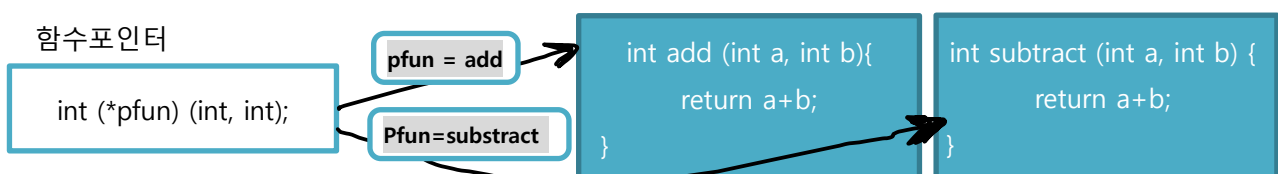
Ex) int (\*pfun) (int, int); //반환자료형 (\*함수 포인터이름) (자료형1, 자료형2, ...);

pfun = add; //변수 pfun에 함수 add의 주소값 대입 가능

pfun = &add; //가능

pfun = subtract; //만약 subtract가 add의 반환형고 인자목록이 동일하다면 가능

//pfun =subtract() //함수호출로 대입해서 오류 생김



cf) 함수 포인터에서 \*함수포인터이름의 괄호는 중요함!!

void \*pf (double \*, double, double); //함수 포인터가 아니라, 함수 원형으로 선언된 것임



- 함수 포인터를 이용한 함수 호출

void add(double \*, double, double); //함수원형

```
int main(void){
    //함수 포인터 선언
    void (*p) (double*, double, double);

    double m, n, result = 0;
    printf("+를 수행할 실수 2개 입력 >>");
    scanf("%lf %lf", &m, &n);

    p = add; //
    p(&result, m, n); // add(&result, m, n), (*p)(&result, m, n)와 동일
    printf("더하기 : %lf + %lf = %lf", m, n, result);

    return 0;
}

void add(double *re, double x, double y){
    *re = x + y;
}
```

----- 출력 -----

+를 수행할 실수 2개 입력 >> 3.2 4.5

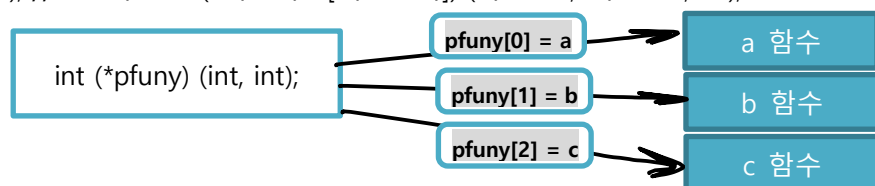
더하기 : 3.200000 + 4.500000 = 7.700000

## 함수 포인터 배열

함수 포인터가 배열의 원소로 여러 개의 함수 포인터를 선언 (함수 포인터가 원소인 배열)

Ex) int (\*pfuny[3]) (int, int); //반환자료형 (\*배열이름[배열크기]) (자료형1, 자료형2, ... );

```
pfuny[0] = a;
pfuny[1] = b;
pfuny[2] = c;
```



- 포인터배열을 선언하면서 주소 값을 초기화하는 문장

```
int (*pfuny[3]) (int, int) = { a, b, c };
```

## void포인터

자료형을 무시하고 주소값만을 다루는 포인터

대상에 상관없이 모든 자료형의 주소를 저장가능한 만능 포인터(일반포인터,배열,구조체,함수주소)

cf) 주소값 : 참조를 시작하는 주소(자료형을 알아야 참조할 범위와 내용을 해석할 방법 알수있음)

```
ex) int m = 10;
    char ch = 'A';
    void *p; //void포인터 p선언
    p = &m; //m의 주소값만 저장
    p = &ch; //ch의 주소값만 저장
```

- void포인터는 모든 주소 저장 가능하나, 가리키는 변수를 참조하거나 수정이 불가능  
⇒ 변수를 참조하기 위해서는 자료형 변환이 필요

```
int m = 10;
double x = 3.4;
```

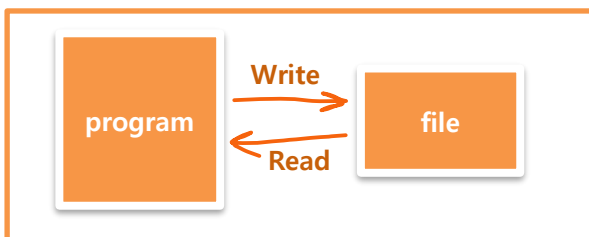
```
void *cp = &m;
int n = *(int *)cp; //int *로 변환
```

```
cp = &x;
int y = *(double *)cp; //double *로 변환
```

```
//함수일 경우
cp = sprint;
((void(*) (void)) cp) (); //결과값 : sprint의 값이 나오게 됨
```

## 파일 기초

메모장과 워드프로세서로만 파일을 만들었음 -> 워드프로세서없이 프로그램만으로 파일 생성가능



프로그램 출력 : 콘솔, 입력 : 키보드

IF) 프로그램 출력을 파일에 하면 파일생성  
프로그램 입력을 파일에 하면 파일입력

- 파일의 필요성  
원래 프로그램이 종료되면 그 안에 있는 모든 자료가 날라가게 된다.  
날라가는 자료를 프로그램이 종료하더라도 계속 저장하고 싶을 때 파일을 이용하게 된다.  
cf) 파일이 저장되어 있는 장소는 보조기억장치인 디스크이다.

• 파일 유형

텍스트 파일(text file)	이진파일(binary file)
<p>메모장 같은 편집기로 작성 문자 기반의 파일</p> <p>내용이 아스키코드같은 문자코드값으로 저장 실수와 정수 같은 내용도 문자형식으로 변환되어 저장</p> <p>텍스트편집기를 이용하여 내용을 볼 수 있고, 수정가능</p>	<p>실행파일, 그림파일, 음악파일, 동영상파일</p> <p>컴퓨터 내부 형식으로 저장되는 파일</p> <p>자료는 메모리 자료내용에서 어떠한 변화를 거치지 않고 그대로 파일에 기록</p> <p>입출력속도는 text file보다 빠름</p> <p>이진파일을 볼 수 있는 특수한 프로그램</p>

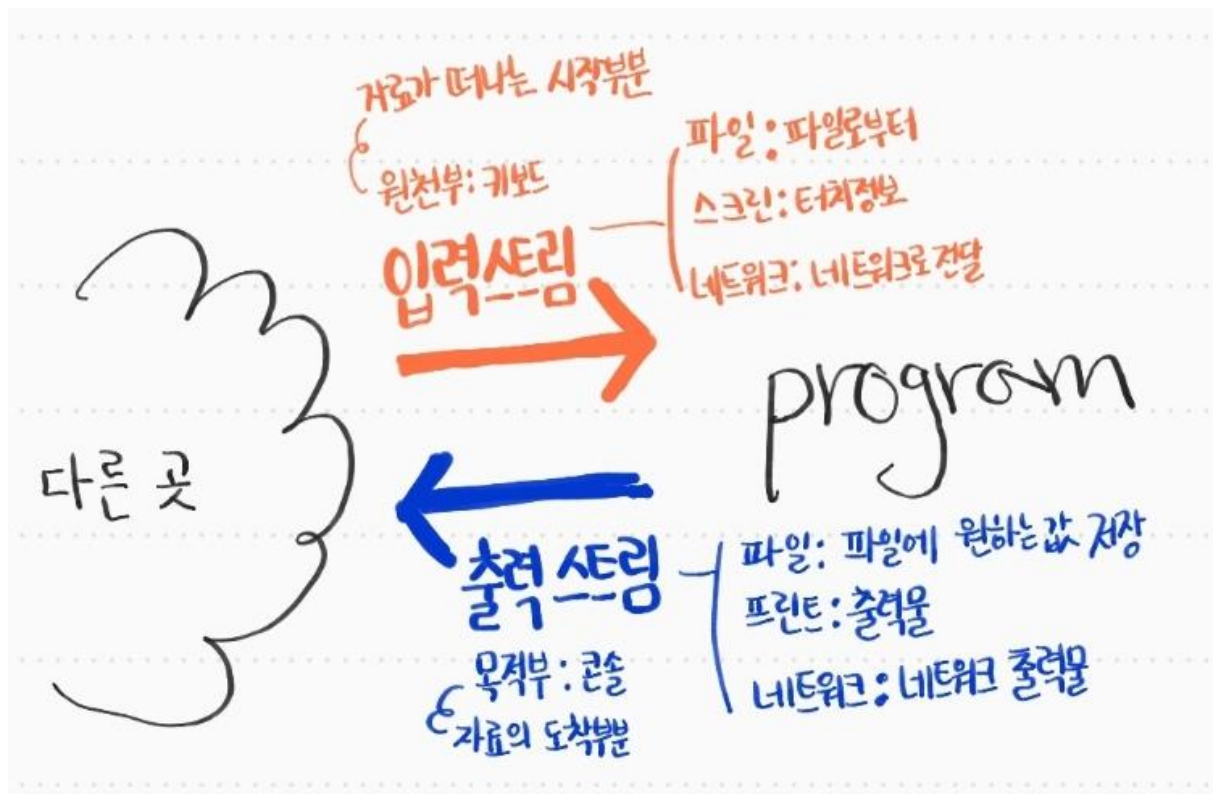
• 입출력 스트림 (입출력 시 이동경로)

cf) 자료의 입력과 출력은 자료의 이동이라 수 있음 (자료 이동하기 위해 이동경로 필요)

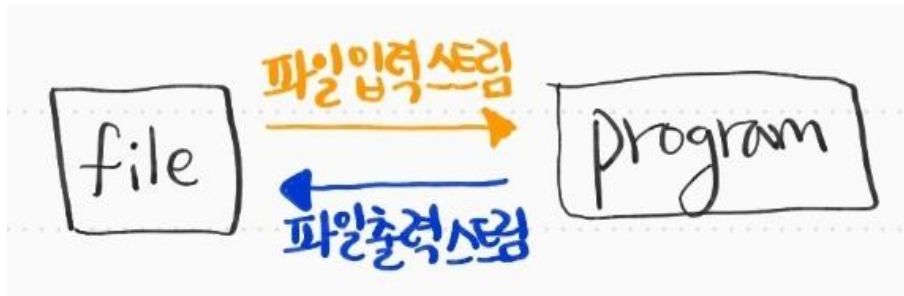
■ 표준입력스트림, 표준출력스트림



■ 입력스트림, 출력스트림



- 파일 스트림 (보조기억장치의 파일과 프로그램을 연결하는 전송경로)  
파일 스트림을 만들기 위해서 특정한 파일이름과 파일모드 필요



#### - 파일 스트림 열기

fopen() 또는 fopen\_s() : 프로그램에서 특정 파일과 파일 스트림 연결하기 위해 사용  
<함수 원형>

FILE \* fopen (const char \* \_Filename, const char \* \_Mode);

- ⇒ FILE은 <stdio.h>에 정의되어 있는 구조체유형 ( 반환값 : FILE의 포인터유형 )
- ⇒ 인자 : 파일이름, 파일열기모드
- ⇒ 파일 스트림 연결 성공 : 파일 포인터 반환
- ⇒ 파일 스트림 연결 실패 : NULL 반환

Erno\_t fopen\_s (FILE \*\*\_File, const char \* \_Filename, cons char \* \_Mode);

- ⇒ 인자 : 파일 포인터의 주소값, 파일 이름, 파일열기모드
- ⇒ 스트림 연결에 성공하면 \_File에 파일 포인터가 저장되고, 0이 반환됨.
- ⇒ 스트림 연결에 실패하면 양수를 반환

#### cf) 파일열기 종류

모드	기능	파일 없음	파일 있음	기존 파일 보호 (파일로 출력 시)
r	읽기(=입력)	에러	기존 파일 이용	에러
r+	읽기/쓰기(=출력)	에러	기존 파일 이용	겹쳐 써짐
w	쓰기	새로 생성	새로 생성	지워짐(새로 생성)
w+	쓰기/읽기	새로 생성	새로 생성	지워짐(새로 생성)
a	쓰기(덧붙이기)	새로 생성	기존 파일 이용	뒤쪽에 써짐
a+	쓰기, 읽기	새로 생성	기존 파일 이용	뒤쪽에 써짐

<https://desire-with-passion.tistory.com/169>

## &lt;연습 문제&gt;

## 1. void 포인터에 대해 설명하시오

대상에 상관없이 모든 자료형의 주소를 저장가능한 만능 포인터이다. 즉, 주소값만 저장하는 포인터라는 의미이다.

## 2. 함수 포인터 배열을 이용하여 값과 연산기호를 입력하면 그 결과값이 수행하는 프로그램을 작성하시오.

- 배열의 크기가 3인 함수 포인터 배열 pfun
- int add(int, int); int mult(int, int); int subtr(int, int);
- 출력 : 4 + 5 = 9

```
#include <stdio.h>
```

```
int add(int, int);
```

```
int mult(int, int);
```

```
int subtr(int, int);
```

```
void main(){
```

```
    int a, b;
```

```
    char op;
```

```
    int (*pfun[3]) (int, int);
```

```
    pfun[0] = add;
```

```
    pfun[1] = subtr;
```

```
    pfun[2] = mult;
```

```
    printf("두 정수와 연산기호를 입력하시오(정수 정수 연산기호) >> ");
```

```
    scanf_s("%d %d %c", &a, &b, &op);
```

```
    switch(op) {
```

```
        case '+': printf("%d + %d = %d", a, b, pfun[0](a, b)); break;
```

```
        case '-': printf("%d - %d = %d", a, b, pfun[1](a, b)); break;
```

```
        case '*': printf("%d * %d = %d", a, b, pfun[2](a, b)); break;
```

```
    }
```

```
}
```

```
int add(int a, int b){
```

```
    return a+b;
```

```
}
```

```
int subtr(int a, int b){
```

```
    return a-b;
```

```
}
```

```
int mult(int a, int b){  
    return a*b;  
}
```

### 3. 텍스트 파일과 이진파일에 대해 설명하시오

텍스트 파일은 문자 기반의 파일로 내용이 아스키코드 같은 문자 코드값으로 저장된다.

이진 파일은 동영상, 이미지파일같이 각각 목적에 알맞은 자료가 이진 형태로 저장되는 파일이다.

### 4. 스트림은 무슨 의미인가?

스트림은 이동경로라고 볼 수 있다. 자료가 입출력을 하기 위해서는 자료의 이동이 필요한 데 이 이동하는 경로를 스트림이라고 한다.

## 7주차

### 핵심 내용

- 파일 스트림
- 텍스트 파일의 입출력 함수
- 이진 파일의 입출력 함수

#### - 파일 스트림 닫기

`fclose()` : `fopen()`으로 연결한 파일 스트림을 닫는 기능을 수행

\*\* 파일 스트림을 연결한 후 파일 처리 끝나면 파일 포인터를 인자로 해서 `fclose()`를 반드시 호출  
<함수원형>

```
int fclose(FILE *_File);
```

- ⇒ 파일 스트림 연결에 할당된 자원을 반납
- ⇒ 파일과 메모리 사이에 있던 버퍼내용 모두 지우는 역할 수행
- ⇒ 성공 : 0 실패 : EOF(양수) 반환

### 텍스트파일의 입출력함수

**`fprintf()` : text파일에 정보를 쓰기위해 사용**

<함수원형>

```
int fprintf(FILE *_File, const char *_Format, ...);
```

- ⇒ 인자 : 이용될 파일, 입출력 제어 문자열, 여러 개의 출력될 변수 또는 상수

**`fscanf()` 또는 `fscanf_s()` : text파일에 정보를 읽기 위해 사용**

<함수원형>

```
int fscanf(FILE *_File, const char *_Format, ...);
```

```
int fscanf_s(FILE *_File, const char *_Format, ...);
```

cf) 인자 중 이용될 파일을 이용하여 표준 입력, 표준 출력 가능함.

```
fprintf(stdout, "%6s%16s%10sWn", "번호", "이름", "중간"); //표준출력 (=printf)
```

<표준 파일 종류>

표준 파일	키워드	장치
표준입력	stdin	키보드
표준출력	stdout	모니터 화면
표준에러	stderr	모니터 화면

cf) Perfect C p.690-691 (파일 스트림 열고 닫고, text파일 수정부분 간단하게!)

실습예제 15-2

쓰기 모드로 파일을 열어 표준 입력으로 받은 학생이름, 중간점수를 파일에 기록하고 다음  
다시 읽기 모드로 그 파일을 열어 기록된 내용 읽어와 표준출력으로 출력하는 프로그램

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

Int main(){
    char fname[] = "grade.txt"; //사용할 파일 이름
    FILE *f;                    //파일 포인터
    char name[30];              //학생이름
    int point, cnt;             //중간점수와 번호

    if(fopen_s(&f, fname, "w") != 0) { //파일 스트림에 쓰기 모드로 연결이 성공하지 않으면
        printf("파일이 열리지 않습니다\n");
        exit(1);
    }; //exit(1)로 종료

    printf("이름과 중간성적을 입력하세요! > > ");
    scanf("%s %d", name, &point);

    //파일 "grade.txt"에 쓰기
    fprintf(f, "%d %s %d\n", ++cnt, name, point);
    fclose(f); //파일스트림 닫음

    if(fopen_s(&f, fname, "r") != 0) { //파일 스트림에 읽기 모드로 연결이 성공하지 않으면
        printf("파일이 열리지 않습니다\n");
        exit(1);
    }; //exit(1)로 종료

    //파일 "grade.txt"에 읽기
    fscanf(f, "%d %s %d\n", &cnt, name, &point);

    //표준출력에 쓰기
    printf(stdout, "%6s%16s%10s\n", "번호", "이름", "중간");
    printf(stdout, "%5d%18s%8d\n", cnt, name, point);
    fclose(f); //파일스트림 닫음
    return 0; }

```

----- 출력 -----

이름과 중간성적을 입력하세요! > > 김소라 70

번호	이름	중간
1	김소라	70



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

&lt;함수원형&gt;

char \* fgets(char \* \_Buf, int \_MaxCount, FILE \* \_File);

- ⇒ 인자 : 문자열이 저장될 문자포인터, 입력할 문자의 최대수, 저장될 파일
- ⇒ 파일로부터 문자열을 개행문자(\n)까지 읽어 마지막 개행문자를 '\0'으로 바꾸어 입력버퍼문자열에 저장

**fputs() : 파일로부터 한 행의 문자열을 출력하는 함수**

&lt;함수원형&gt;

int fputs(char \* \_Buf, FILE \* \_File);

- ⇒ 인자 : 출력될 문자열이 저장된 문자포인터, 출력될 파일

**feof() : 파일 스트림의 EOF(End of File) 표시를 검사하는 함수**

- 파일에서 읽어낼 때 "끝까지 읽어라"하면 파일의 끝이 어디인지 체크
- 표준입력을 받을 때도 "여기가 끝이야"하고 인지하고 싶을 때

&lt;함수원형&gt;

int feof (FILE \* \_File);

- ⇒ 끝이면 0이 아닌 값
- ⇒ 끝이 아니면 0을 반환

**ferror() : 파일처리에서 오류가 발생했는지 검사하는 함수**

&lt;함수원형&gt;

int ferror (FILE \* \_File);

- ⇒ 오류 발생 : 0이 아닌값
- ⇒ 오류 발생하지 않음 : 0

cf) 여러 줄의 표준입력을 처리 방식 : while( !feof(stdin) ) { ... } 구문 이용

표준입력에서 입력 종료하려면 EOF의미하는 ctrl+Z를 새로운 행의 처음에 누름

**fgetc() 또는 getc() : 파일로부터 문자 하나를 입력받는 함수**

&lt;함수원형&gt;

int fgetc(FILE \* \_File);

int getc(FILE \* \_File);

- ⇒ getc()를 이용한 매크로 정의 : getchar()

**fputc() 또는 putc() : 파일로부터 문자 하나를 입력받는 함수**

&lt;함수원형&gt;

int fputc(int \_Ch, FILE \* \_File);

int putc(int \_Ch, FILE \* \_File); //인자 : 문자, 파일

- ⇒ putc()를 이용한 매크로 정의 : putchar()

**<파일 내용을 표준출력으로 그대로 출력>**

도스 명령어 type 같이 파일 내용을 그대로 콘솔에 출력 가능 cf) Perfect C p.699-702

**이진 파일의 입출력 함수**

이진파일은 C언어의 자료형을 모두 유지하면서 바이트 단위로 저장되는 파일 (블록 단위)

**fwrite() : 바이트 단위로 원하는 블록을 파일에 출력하기 위한 함수 (저장)**

<함수원형>

```
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *f);
```

- ⇒ 인자 : 출력될 자료의 주소값, 자료항목의 byte크기, 항목갯수, 출력될 파일 포인터
- ⇒ ptr이 가리키는 메모리에서 size만큼 n개를 파일 f에 쓰는 함수
- ⇒ 반환값 : 출력된 항목의 개수

**fread() : 이진 파일에 저장되어 있는 자료를 입력하는 함수 (읽어옴)**

<함수원형>

```
size_t fread(void *dstbuf, size_t size, size_t n, FILE *f);
```

- ⇒ 인자 : 입력된 자료의 주소값, 자료항목의 byte크기, 항목갯수, 입력된 파일 포인터
- ⇒ 파일 f에서 n개만큼 메모리 dstbuf에서 읽어오는 함수
- ⇒ 반환값 : 입력된 항목의 개수

cf) fprintf()와 fscanf(), fscanf\_s()는 자료의 입출력을 text모드처리(파일에 아스키코드값으로 저장)  
fwrite()는 메모리상에 있는 것을 그대로 저장 -> fread()로 읽어야만 함.

cf) 이진파일에서 파일열기모드종류

모드	의미	파일이 없으면	파일이 있으면
"rb"	읽기 전용(read)	NULL 반환	정상 동작
"wb"	쓰기 전용(write)	새로 생성	기존 내용 삭제
"ab"	추가 쓰기(append)	새로 생성	기존 내용 뒤에 추가
"rb+"	읽기와 쓰기	NULL 반환	정상 동작
"wb+"	읽기와 쓰기	새로 생성	기존 내용 삭제
"ab+"	추가를 위한 읽기와 쓰기	새로 생성	기존 내용 뒤에 추가

## &lt;연습 문제&gt;

1. 다음은 구조체 변수 하나를 파일에 출력하는 프로그램을 작성하라

- 구조체 struct company 멤버 구조 : 이름, 사원수
- 파일 이름 : company.bin

```
#include <stdio.h>
```

```
int main() {
    struct company {
        char name[10]; //이름
        int employee; //사원수
    };
    typedef struct company company;

    char fname[] = "company.bin";
    FILE *f;
    fopen_s(&f, fname, "wb");

    company kakao = { "(주)kakao", 1000 };
    fwrite(&kakao, sizeof(company), 1, f);
    fclose(f);

    return 0;
}
```

2. 위 코드에서 구현한 company.bin을 읽어 표준출력하는 프로그램이다. 빈 부분 완성하시오

```
#include <stdio.h>
```

```
int main() {
    struct company {
        char name[10]; //이름
        int employee; //사원수
    };
    typedef struct company company;

    char fname[] = "company.bin";
    FILE *f;
    fopen_s(_____(1)_____);

    company kakao;
```

```
fread(_____(2)_____);  
printf(___ (3) ___, "%18s%8d", kakao.name, kakao.employee);  
_____(4)_____;  
  
return 0;  
}
```

답

- (1) &f, fname, "rb"
- (2) &kakao, sizeof(company), 1, f
- (3) stdout
- (4) fclose(f)

## 맺음말

필자는 지난 1 학기의 반의 여정을 하나의 포트폴리오로 정리하는 과정에서 “내가 이렇게 많은 개념들을 배웠었구나”라는 생각과 “이러한 개념들에 대해 정리하는 시간을 갖게 되어서 다행이다”라는 생각이 들었다. 이러한 부분에 있어 이 포트폴리오 과제를 수행하게 된 것에 감사함을 표한다.

이 포트폴리오가 완성됨에 따라 “더 잘할 수 있을 거 같은데”라는 아쉬움, “드디어 완성했다”라는 뿌듯함, “내가 이 글을 적었다”라는 성취감까지 여러 복잡한 감정들이 올라와서 무언가 가슴이 벅찼다. 이 포트폴리오를 적기위해 몇 밤을 샀는지 잘 모르겠다. 이렇게 많이 공들여서 적어본 것이 거의 처음이었기에 더 가슴이 벅찼는지도 모른다.

이 글을 마치면서 앞으로 한번쯤은 이런 포트폴리오를 만들어야 겠다는 생각이 들었다. 많은 시간을 들이고 여러 자료들을 찾으며 머리가 터질 듯이 아프겠지만 이러한 경험을 해야 c 언어에 대한 정보가 머리 속에 쉽게 사라지지 않고, 차곡차곡 쌓아질 것이다. 이러한 정보가 쌓이고 쌓이다 보면 결국에는 c 언어에 잘 알고, c 언어 프로그램을 하는 사람이 될 수 있지 않을까? 라는 자그마한 소망이 있다.

