

# C 애플리케이션 구현

---



---

컴퓨터정보공학과

학번 : 20192635

작성자: 김한수



---

# 목차

- 머리말
- 강의 계획서
- CHAPTER 11 문자와 문자열
  - 11-1 문자와 문자열
  - 11-2 문자열 관련 함수
  - 11-3 여러 문자열 처리
- CHAPTER 12 변수 유효범위
  - 12-1 전역변수와 지역변수
  - 12-2 정적 변수와 레지스터 변수
  - 12-3 메모리 영역과 변수 이용
- CHAPTER 13 구조체와 공용체
  - 13-1 구조체와 공용체
  - 13-2 자료형 재정의
  - 13-3 구조체와 공용체의 포인터와 배열
- CHAPTER 14 함수와 포인터 활용
  - 14-1 함수의 인자전달 방식
  - 14-2 포인터 전달과 반환
  - 14-3 함수 포인터와 void 포인터
- CHAPTER 15 파일 처리
  - 15-1 파일 기초
  - 15-2 텍스트 파일 입출력

---

## 머리말

포트폴리오 작성을 마치며.....

포트폴리오를 작성하는 경험이 처음이라 낯설고 조금 힘들었지만 좋은 경험이 되었습니다. 개인적으로 C 애플리케이션구현 중간고사 대체과제로 학생 포트폴리오를 작성하면서 복습한다는 느낌으로 시작하였습니다. 처음에는 가볍게 시작하였지만 쉽지만은 않은 과정이었습니다. 그래도 노력한 만큼 이 과정 속에서 얻어가는 것이 분명히 있다고 느끼기에 충분히 가치가 있는 시간이었다고 생각합니다.

개인적으로는 자료구조, 함수들을 어떤 경우에 사용하는지와 코드들의 전체적인 흐름에 대한 시야를 넓히는데 큰 도움이 되었습니다. 단순히 "이런 자료형, 구조가 있구나"보다는 몇 단계 더 나아가서 이것을 쓸 방안들을 미리 생각하면서 이번 포트폴리오를 작성한 것도 큰 도움이 되었습니다.

마치며, 앞으로 프로젝트를 몇 가지 진행하게 될 예정에 있습니다. 프로젝트 과정 속에서 C 애플리케이션구현 강의에서 학습한 내용들을 활용하면서 느낄 성취감과 또 경험 속에서 성장할 나 자신을 기대하며 이런 기회를 마련해주신 강환수 교수님께 감사의 말을 전합니다. 감사합니다.

컴퓨터정보공학과  
20192635 김한수

본 포트폴리오는 Perfert C 교재의 내용을 먼저 읽고 개인적으로 요약 정리를 한 후 예제를 풀어보는 나름대로의 형식으로 작성되었으니 참고 부탁드립니다. 감사합니다.

|              |  |                           |        |        |
|--------------|--|---------------------------|--------|--------|
| 2020 학년도 1학기 | 전공   | 컴퓨터정보공학과(사회맞춤형 지능형 컴퓨팅과정) | 학부     | 컴퓨터공학부 |
| 과 목 명        | C애플리케이션구현(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<br>+ 연구실 : 2호관-706<br>+ 전 화 : 02-2610-1941<br>+ E-MAIL : hskang@dongyang.ac.kr<br>+ 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.<br>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.<br>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<br>C언어 기초 및 조건문과 반복문 복습  |                           |        |        |
| 목표및 내용       | C언어 기초<br>통합개발환경 테스트<br>기초적인 코드 실습   |                           |        |        |

|             |   |
|-------------|---|
| 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<br>연산자<br>f-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<br>단일포인터<br>다중포인터<br>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                               |

|              |               |
|--------------|---------------|
| <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     |

|                |   |
|----------------|---|
| <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> | 장애학생을 위한 별도의 수강 지원을 받을 수 있습니다.<br>언어가 문제가 되는 학생은 글로 된 과제 안내, 확대문자 시험지 제공 등의 지원을 드립니다. |

# CHAPTER 11

## 문자와 문자열

---



## 11-1 문자와 문자열

### \*문자와 문자열의 개념

#### 문자

- 영어의 알파벳이나 한글의 한 글자를 작은 따옴표로 둘러싸서 'A', '김'과 같이 표기함.
- 작은 따옴표에 의해 표기된 문자를 **문자 상수**라 함.
- \*\*C 언어에서 저장공간 1 바이트인 자료형 char로 지원함.

#### 문자열

- 문자의 모임인 일련의 문자를 **문자열(string)**이라 함.
- 큰 따옴표로 둘러싸서 "python"으로 표기함.
- 문자 하나도 큰 따옴표로 둘러싸면 문자열 상수. Ex) "A"
- **문자열을 작은 따옴표로 둘러싸면 오류 발생** Ex) 'ABC'

### \*문자와 문자열의 선언

C 언어에서는 **char** 형 변수에 문자를 저장함.

문자열을 저장하려면 **문자 배열**인 **char[]**를 사용해야함

#### \*주의\*

문자열의 마지막을 의미하는 NULL 문자 'w0'가 마지막에 저장되어야 함!!

>> 크기 지정 시 문자수 +1 만큼의 크기를 지정해주는 것이 중요!

#### 문자와 문자열 선언 방법

```
//문자 선언과 출력
char ch = 'A';
//문자열 선언 방법1
char java[] = { 'J', 'A', 'V', 'A', 'w0' };
```

```
//문자열 선언 방법2
char c[] = "C language";    //크기를 생략하는 것이 간편
//문자열 선언 방법3
char csharp[5] = "C#";
```

## + printf()를 사용한 문자와 문자열 출력위한 형식제어문자

문자 (Char) : %c

문자열 (Char[]) : %s

### 실습예제 11-1

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

int main(void)
{
    //문자 선언과 출력
    char ch = 'A';
    printf("%c %d\n", ch, ch);

    //문자열 선언 방법1
    char java[] = { 'J', 'A', 'V', 'A', '\0' };
    printf("%s\n", java);
    //문자열 선언 방법2
    char c[] = "C language";    //크기를 생략하는 것이 간편
    printf("%s\n", c);
    //문자열 선언 방법3
    char csharp[5] = "C#";
    printf("%s\n", csharp);

    //문자 배열에서 문자 출력
    printf("%c%c\n", csharp[0], csharp[1]);

    return 0;
}
```

## 문자열을 구성하는 문자 참조

문자열을 처리하는 다른 방법은 문자열 상수를 문자 포인터에 저장하는 방식!

예제는 다음과 같다.

## 실습예제 11-2

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

int main(void)
{
    char* java = "java";
    printf("%s ", java);

    //문자 포인터가 가리키는 문자 이후를 하나 하나 출력
    int i = 0;
    while (java[i]) //while (java[i] != '\0')
        printf("%c", java[i++]);
    printf(" ");

    i = 0;
    while (*(java + i) != '\0') //java[i]는 *(java + i)와 같음
        printf("%c", *(java + i++));
    printf("\n");

    //수정 불가능, 실행오류 발생
    java[0] = 'J';

    return 0;
}
```

## ‘W0’ 문자에 의한 문자열 분리

### \*\*중요

함수 `printf()`에서 `%s` 는 문자 포인터가 가리키는 위치에서 NULL 문자(‘W0’)까지를 하나의 문자열로 인식함.

\*예제에서 중간에 널 문자를 넣음으로써 문자열이 분리되는 것을 확인함.

### 실습예제 11-3

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

int main(void)
{
    char c[] = "C C++ Java";
    printf("%sWn", c);
    c[5] = 'W0'; //널 문자에 의해 문자열 분리
    printf("%sWn%sWn", c, (c + 6));

    //문자 배열의 각 원소를 하나 하나 출력하는 방법
    c[5] = ' '; //널 문자를 빈 문자로 바꾸어 문자열 복원
    char* p = c;
    while (*p) /*(*p != 'W0')도 가능
        printf("%c", *p++);
    printf("Wn");

    return 0;
}
```

## 다양한 문자 입출력

### 버퍼처리 함수 getchar()

- 문자 입력을 위한 함수
- enter 키를 누르면 입력이 실행되는 라인 버퍼링(line buffering)방식을 사용함

### 함수 getche()

- getchar()와 달리 버퍼를 사용하지 않고 문자를 입력하는 함수
- 문자 하나하나를 바로 바로 입력할 수 있음
- 사용하기 위해서 헤더파일 conio.h 삽입 필요
- 입력된 문자는 바로 출력되니 주의!

### 함수 getch()

- 문자 입력을 위한 함수
- getche()와 동일하게 버퍼 사용 X
- conio.h 삽입 필요!
- getche()와 달리 입력한 문자가 화면에 보이지 않음!!

## scanf(), getchar(), getche(), getch() 비교

표 11-1 문자입력 함수 scanf(), getchar(), getche(), getch()의 비교

| 함수              | scanf("%c", &ch)  | getchar() | getche()<br>_getche() | getch()<br>_getch() |
|-----------------|-------------------|-----------|-----------------------|---------------------|
| 헤더파일            | stdio.h           |           | conio.h               |                     |
| 버퍼 이용           | 버퍼 이용함            |           | 버퍼 이용 안함              |                     |
| 반응              | [enter] 키를 눌러야 작동 |           | 문자 입력마다 반응            |                     |
| 입력 문자의 표시(echo) | 누르면 바로 표시         |           | 누르면 바로 표시             | 표시 안됨               |
| 입력문자 수정         | 가능                |           | 불가능                   |                     |

## 실습예제 11-4

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

int main(void)
{
    char ch;

    printf("문자를 계속 입력하고 Enter를 누르면 >>\n");
    while ((ch = getchar()) != 'q')
        putchar(ch);

    printf("\n문자를 누를 때마다 두 번 출력 >>\n");
    while ((ch = _getche()) != 'q')
        putchar(ch);

    printf("\n문자를 누르면 한 번 출력 >>\n");
    while ((ch = _getch()) != 'q')
        _putch(ch);
    printf("\n");

    return 0;
}
```

## 문자열 입력

### 문자배열 변수로 scanf()에서 입력

- scanf()는 공백으로 구분되는 하나의 문자열을 입력받을 수 있음!
- scanf("%s", 저장할 변수(str))로 사용
- 헤더파일 stdio.h 삽입

### 실습예제 11-5

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

int main(void)
{
    char name[20], dept[30]; //char *name, *dept; 실행 오류 발생

    printf("%s", "학과 입력 >> ");
    scanf("%s", dept);
    printf("%s", "이름 입력 >> ");
    scanf("%s", name);
    printf("출력: %10s %10s\n", dept, name);

    return 0;
}
```

## gets()와 puts()

- **gets()**와 **puts()**는 한 행의 문자열 입출력에 유용한 함수!!
- 헤더파일 **stdio.h** 삽입
- \*\* **gets()**는 마지막 [enter]키를 'w0'으로 대체하여 입력!
- \*\* **puts()**는 마지막 'w0'를 'wn'으로 대체하여 출력!
- \*\*\* **puts()**는 오류 발생 시 EOF(-1)를 반환

### 실습예제 11-6

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

int main(void)
{
    char line[101]; //char *line 으로는 오류발생

    printf("입력을 종료하려면 새로운 행에서 (ctrl + Z)를 누르십시오.wn");
    while (gets(line))
        puts(line);
    printf("wn");

    while (gets_s(line, 101))
        puts(line);
    printf("wn");

    return 0;
}
```



## LAB 한 행을 표준입력으로 받아 문자 하나 하나를 그대로 출력

### Lab 11-1

```
// 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;
}
```

## 11-2 문자열 관련 함수

### 다양한 문자열 라이브러리 함수

- 헤더파일 **string.h** 삽입!
- **size\_t** 는 비부호 정수(unsigned int type), **void \*** 는 아직 정해지지 않은 다양한 포인터를 의미

표 11-1 문자열 배열에 관한 다양한 함수

| 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                     |

### 실습예제 11-7

```
// file: memfun.c
#include <stdio.h>
#include <string.h>
int main(void)
{
    char src[50] = "https://www.visualstudio.com";
    char dst[50];
    printf("문자배열 src = %s\n", src);
    printf("문자열크기 strlen(src) = %d\n", strlen(src));
    memcpy(dst, src, strlen(src) + 1);
    printf("문자배열 dst = %s\n", dst);
    memcpy(src, "안녕하세요!", strlen("안녕하세요!") + 1);
    printf("문자배열 src = %s\n", src);
    char ch = ':';
    char* ret;
    ret = memchr(dst, ch, strlen(dst));
    printf("문자 %c 뒤에는 문자열 %s 이 있다.\n", ch, ret);
    return 0;
}
```

**\*\* 문자열 비교와 복사, 그리고 문자열 연결 등 다양한 문자열 처리는  
헤더파일 string.h 에 함수원형으로 선언된 라이브러리 함수로 제공됨!**

### 함수 strcmp(), strncmp()

- 두 문자열을 사전(lexicographically)상의 순서로 비교하는 함수
- **strncmp()**는 비교할 문자의 최대 수를 지정할 수 있음.
- **앞에서부터 아스키코드를 기준으로 순차적 비교 방식!**

문자열 비교 함수: 헤더파일 string.h 삽입

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

두 인자인 문자열에서 같은 위치의 문자를 앞에서부터 다를 때까지 비교하여 같으면 0을 반환하고, 앞이 크면 양수를, 뒤가 크면 음수를 반환한다.

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

두 인자 문자열을 같은 위치의 문자를 앞에서부터 다를 때까지 비교하나 최대 n까지만 비교하여 같으면 0을 반환하고, 앞이 크면 양수를, 뒤가 크면 음수를 반환한다.

### 실습예제 11-8

```
// file: strcmp.c
#include <stdio.h>
#include <string.h>
int main(void)
{
    char* s1 = "java";
    char* s2 = "java";
    printf("strcmp(%s, %s) = %d\n", s1, s2, strcmp(s1, s2));
    s1 = "java";
    s2 = "jav";
    printf("strcmp(%s, %s) = %d\n", s1, s2, strcmp(s1, s2));
    s1 = "jav";
    s2 = "java";
    printf("strcmp(%s, %s) = %d\n", s1, s2, strcmp(s1, s2));
    printf("strncmp(%s, %s, %d) = %d\n", s1, s2, 3, strncmp(s1, s2, 3));
    return 0;
}
```

### 문자열 복사와 연결

## 함수 strcpy(), strncpy()

- 문자열을 복사하는 함수

### 앞 인자 문자열에 뒤 인자 문자열을 복사함

- **strncpy()**는 복사할 최대 문자 수를 지정 가능!
- **\*\*문자열은 항상 마지막 NULL 문자까지 포함하므로 꼭 기억!!**

#### 문자열 복사 함수

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

- 앞 문자열 dest에 처음에 뒤 문자열 null 문자를 포함한 source 를 복사하여 그 복사된 문자열을 반환한다.
- 앞 문자열은 수정되지만 뒤 문자열은 수정될 수 없다.

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

- 앞 문자열 dest에 처음에 뒤 문자열 source에서 n개 문자를 복사하여 그 복사된 문자열을 반환한다.
- 만일 지정된 maxn이 source의 길이보다 같면 나머지는 모두 널 문자가 복사된다. 앞 문자열은 수정되지만 뒤 문자열은 수정될 수 없다.

```
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);
```

- 두 번째 인자인 sizedest는 정수형으로 dest의 크기를 입력한다.
- 반환형 errno\_t는 청수형이며 반환값은 오류번호로 성공하면 0을 반환한다.
- Visual C++에서는 앞으로 함수strcpy\_s()와 strncpy\_s()의 사용을 권장한다.

## 실습예제 11-9

```
// file: strcpy.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
int main(void)
{
    char dest[80] = "Java";
    char source[80] = "C is a language.";
    printf("%s\n", strcpy(dest, source));
    //printf("%d\n", strcpy_s(dest, 80, source));
    //printf("%s\n", dest);
    printf("%s\n", strncpy(dest, "C#", 2));

    printf("%s\n", strncpy(dest, "C#", 3));
    //printf("%d\n", strncpy_s(dest, 80, "C#", 3));
    //printf("%s\n", dest);

    return 0;
}
```

## 함수 strcat(), strncat()

- 문자열 연결 함수
- 앞 문자열에 뒤 문자열의 null 문자까지 연결
- 앞 문자열 주소를 반환
- \*\*앞 문자열은 뒤 문자열의 길이를 모두 수용할 수 있는 공간이 필요!!
- 공간 부족 시 문제를 예방하기 위해 **strncat()**으로 조절 가능!

문자열 연결 함수

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

• 앞 문자열 dest에 뒤 문자열 source를 연결(concatenate)해 저장하며, 이 연결된 문자열을 반환하고 뒤 문자열은 수정될 수 없다.

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

• 앞 문자열 dest에 뒤 문자열 source중에서 n개의 크기만큼을 연결(concatenate)해 저장하며, 이 연결된 문자열을 반환하고 뒤 문자열은 수정될 수 없다.  
• 지정한 maxn이 문자열 길이보다 크면 null 문자까지 연결한다.

```
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);
```

• 두 번째 인자인 sizedest는 정수형으로 dest의 크기를 입력한다.  
• 반환형 errno\_t는 정수형이며 반환값은 오류번호로 성공하면 0을 반환한다.  
• Visual C++에서는 앞으로 함수strcat\_s()와 strncat\_s()의 사용을 권장한다.

## 실습예제 11-10

```
// file: strcat.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
int main(void)
{
    char dest[80] = "C";

    printf("%s\n", strcat(dest, " is "));
    //printf("%d\n", strcat_s(dest, 80, " is "));
    //printf("%s\n", dest);
    printf("%s\n", strncat(dest, "a java", 2));
    //printf("%d\n", strncat_s(dest, 80, "a proce", 2));
    //printf("%s\n", dest);
    printf("%s\n", strcat(dest, "procedural "));
    printf("%s\n", strcat(dest, "language."));

    return 0;
}
```

## 함수 strtok()

- 문자열에서 구분자(delimiter)인 문자를 여러 개 지정하여 토큰을 추출
- \*\*\*앞 str 인자는 문자열 상수를 사용할 수 없음!!!

### \*사용방법\*

1. 문장 ptoken = strtok(str, delimiter); 으로 첫 토큰을 추출
  2. 결과를 저장한 ptoken 이 NULL 이면 더 이상 분리할 토큰이 없는 것!!
  3. while(ptoken != NULL)로 순차적 추출 가능
- \*두 번째 토큰부터는 ptoken(NULL, delimiter)로 출력

## 실습예제 11-11

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

int main(void)
{
    char str1[] = "C and C++ language are best!";
    char* delimiter = " ,Wt!";
    //char *next_token;

    printf("문자열 W"%sW"을 >>Wn", str1);
    printf("구분자[%s]를 이용하여 토큰을 추출 >>Wn", delimiter);
    char* ptoken = strtok(str1, delimiter);
    //ptoken = strtok_s(str, delimiter, &next_token);
    while (ptoken) //(ptoken != NULL)
    {
        printf("%sWn", ptoken);
        ptoken = strtok(NULL, delimiter); //다음 토큰을 반환
        //ptoken = strtok_s(NULL, delimiter, &next_token); //다음 토큰을 반환
    }

    return 0;
}
```

## 문자열의 길이와 위치 검색

- `strlen()`은 NULL 문자열 제외한 문자열 길이를 반환하는 함수
- `strlwr()`, `strupr()`는 각각 인자를 모두 소문자, 대문자로 변환된 문자열을 반환하는 함수

표 11-2 다양한 문자열 관련 함수

| 함수원형  | 설명  |
|---|---|
| <code>char * strlwr(char * str);</code><br><code>errno_t _strlwr_s(char * str, rsize_t rsize); //Visual C++ 권장함수</code> | 문자열 <code>str</code> 을 모두 소문자로 변환하고 변환한 문자열을 반환하므로 <code>str</code> 은 상수이면 오류가 발생하며, <code>errno_t</code> 는 정수형의 오류번호이며, <code>size_t</code> 도 정수형으로 <code>rsize</code> 는 <code>str</code> 의 길이 |
| <code>char * strupr(char * str);</code><br><code>errno_t _strupr_s(char * str, rsize_t rsize); //Visual C++ 권장함수</code> | 문자열 <code>str</code> 을 모두 대문자로 변환하고 변환한 문자열을 반환하므로 <code>str</code> 은 상수이면 오류가 발생하며, <code>errno_t</code> 는 정수형의 오류번호이며, <code>size_t</code> 도 정수형으로 <code>rsize</code> 는 <code>str</code> 의 길이 |
| <code>char * strpbrk(const char * str, const char * charset);</code>  | 앞의 문자열 <code>str</code> 에서 뒤 문자열 <code>charset</code> 에 포함된 문자가 나타나는 처음 위치를 찾아 그 주소값을 반환하며, 만일 찾지 못하면 NULL 포인터를 반환  |
| <code>char * strstr(const char * str, const char * strsearch);</code>   | 앞의 문자열 <code>str</code> 에서 뒤 문자열 <code>strsearch</code> 이 나타나는 처음 위치를 찾아 그 주소값을 반환하며, 만일 찾지 못하면 NULL 포인터를 반환  |
| <code>char * strchr(const char * str, char ch);</code>  | 앞의 문자열 <code>str</code> 에서 뒤 문자 <code>ch</code> 가 나타나는 처음 위치를 찾아 그 주소값을 반환하며, 만일 찾지 못하면 NULL 포인터를 반환  |

## 실습예제 11-12

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

int main(void)
{
    char str[] = "JAVA 2017 go c#";
    printf("%d\n", strlen("java"));           //java의 길이: 4
    printf("%s, ", _strlwr(str));             //모두 소문자로 변환
    printf("%s\n", _strupr(str));            //모두 대문자로 변환

    //문자열 VA가 시작되는 포인터 반환: VA 2013 GO C#
    printf("%s, ", strstr(str, "VA"));
    //문자 A가 처음 나타나는 포인터 반환: AVA 2013 GO C#
    printf("%s\n", strchr(str, 'A'));

    return 0;
}
```

## LAB 문자열을 역순으로 저장하는 함수 reverse() 구현

### Lab 11-2

```
// streverse.c:
#include <stdio.h>
#include <string.h>

void reverse(char str[]);

int main(void)
{
    char s[50];
    memcpy(s, "C Programming!", strlen("C Programming!") + 1);
    printf("%s\n", s);

    reverse(s);
    printf("%s\n", s);

    return 0;
}

void reverse(char str[])
{
    for (int i = 0, j = strlen(str) - 1; i < j; i++, j--)
    {
        char c = str[i];
        str[i] = str[j];
        str[j] = c;
    }
}
```



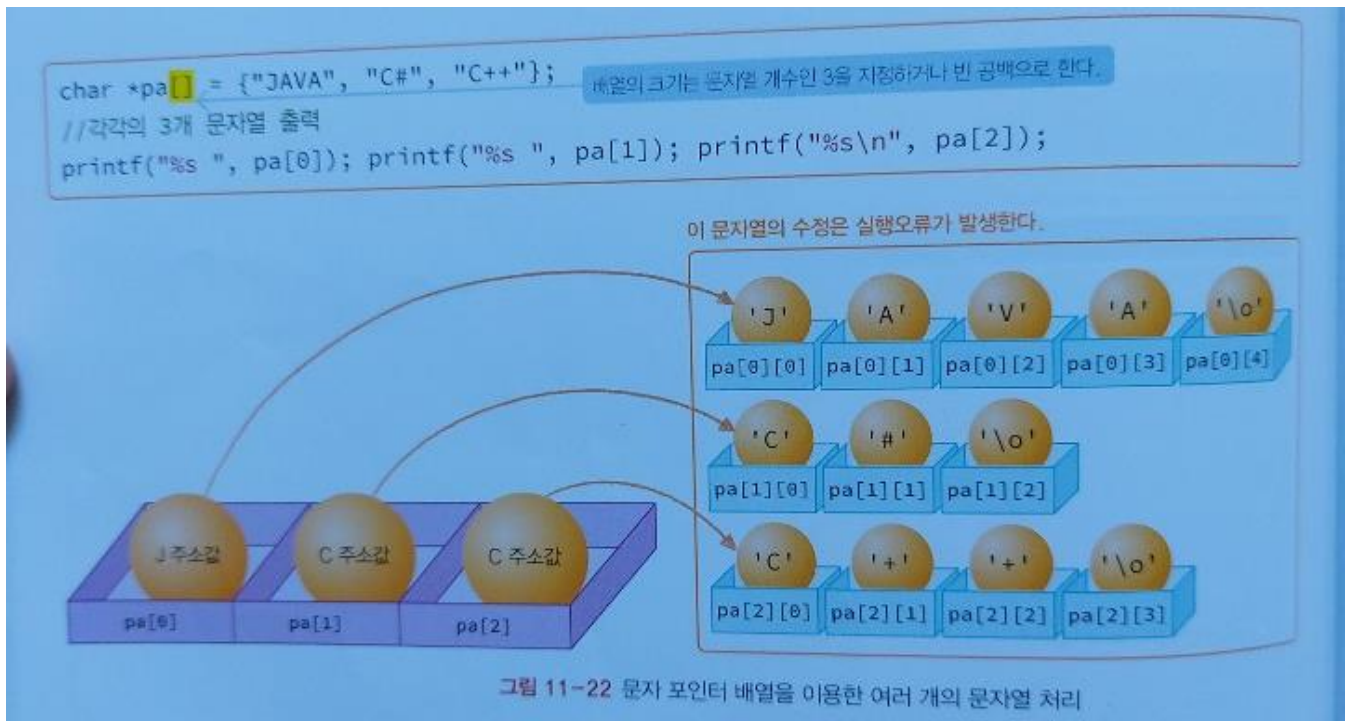
## 11-3 여러 문자열 처리

### 문자열 포인터 배열과 이차원 문자 배열

#### 문자 포인터 배열

- 여러 개의 문자열을 처리
- 문자 포인터 배열을 이용
- \*최적의 공간을 사용하는 장점
- \*문자열 상수의 수정 불가능 Ex) `pa[0][1] = 'v';` -> 오류발생

#### 처리 방법



## 이차원 문자 배열

- 여러 개의 문자열을 처리
- 이차원 배열 이용
- \*크기 지정 시 NULL 문자 고려 (가장 긴 문자열의 길이+1)

## 처리방법

```
char ca[][5] = {"JAVA", "C#", "C++"};
//각각의 3개 문자열 출력
printf("%s ", ca[0]); printf("%s ", ca[1]); printf("%s\n", ca[2]);
```

첫 번째(열) 크기는 문자열 갯수를 지정하거나  
한 글자씩 두며, 두 번째(행) 크기는 문자열  
중에서 가장 긴 문자열의 길이보다 1크게 지  
정한다.

이 문자열의 수정은 실행오류가 발생한다.

|          |          |          |          |          |
|----------|----------|----------|----------|----------|
| 'J'      | 'A'      | 'V'      | 'A'      | '\0'     |
| ca[0][0] | ca[0][1] | ca[0][2] | ca[0][3] | ca[0][4] |
| 'C'      | '#'      | '\0'     | '\0'     | '\0'     |
| ca[1][0] | ca[1][1] | ca[1][2] | ca[1][3] | ca[1][4] |
| 'C'      | '+'      | '+'      | '\0'     | '\0'     |
| ca[2][0] | ca[2][1] | ca[2][2] | ca[2][3] | ca[2][4] |

그림 11-23 이차원 문자배열을 이용한 여러 문자열 처리

## 실습예제 11-13

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

int main(void)
{
    //문자 선언과 출력
    char ch = 'A';
    printf("%c %d\n", ch, ch);
    //문자열 선언 방법1
    char java[] = { 'J', 'A', 'V', 'A', 'W\0' };
    printf("%s\n", java);
    //문자열 선언 방법2
    char c[] = "C language"; //크기를 생략하는 것이 간편
    printf("%s\n", c);
    //문자열 선언 방법3
    char csharp[5] = "C#";
    printf("%s\n", csharp);
    //문자 배열에서 문자 출력
    printf("%c%c\n", csharp[0], csharp[1]);
    return 0;
}
```

## 명령행 인자

### main(int argc, char \* argv[])

- 도스 창(command prompt)처럼 명령행에서 입력하는 문자열을 프로그램으로 전달하는 방법이 **명령행 인자(command line arguments)**를 사용하는 방법이다.
- 프로그램에서 명령행 인자는 main() 함수의 인자로 기술됨.
- 명령행 인자를 받는 방법 : main(void) -> main(int argc, char \* argv[])
- **\*\*실행 프로그램 이름도 하나의 명령행 인자에 포함되니 주의!**

예제의 실행결과를 보면 이해하기 쉬웠다.

예제를 실행하기 위해서 [프로젝트/{프로젝트이름} 속성] > [디버깅] > [명령인수 수정] 단계가 필요했다. 설정과 실행 결과는 다음과 같았다.

```
실행 명령행 인자(command line arguments) >>
argc = 4
argv[0] = D:\Visual Studio source\anyway\Debug\anyway.exe
argv[1] = C#
argv[2] = C++
argv[3] = Java
```

### 실습예제 11-14

```
// 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;
}
```

## LAB 여러 문자열 처리

### Lab 11-3

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

int main(void)
{
    char str1[] = "JAVA";
    char str2[] = "C#";
    char str3[] = "C++";

    char* pstr[] = { str1, str2, str3 };

    //각각의 3개 문자열 출력
    printf("%s ", pstr[0]);
    printf("%s ", pstr[1]);
    printf("%s\n", pstr[2]);

    //문자 출력
    printf("%c %c %c\n", str1[0], str2[1], str3[2]);
    printf("%c %c %c\n", pstr[0][1], pstr[1][1], pstr[2][1]);

    return 0;
}
```

# CHAPTER 12

## 변수 유효범위

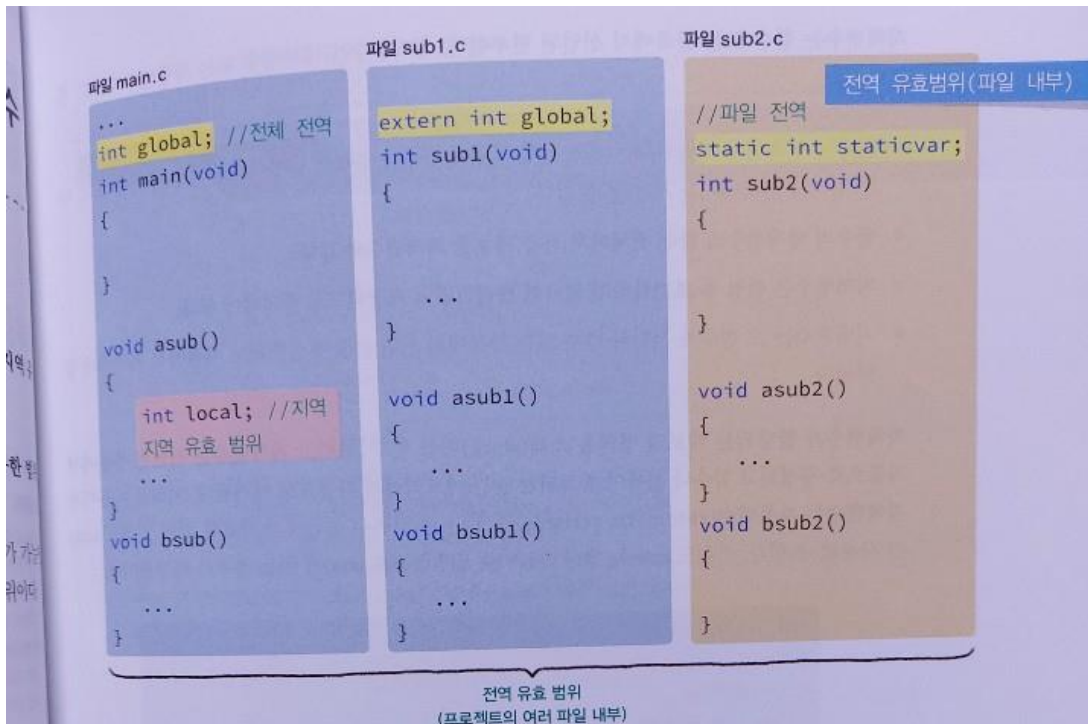
---

## 12-1 전역변수와 지역변수

### 변수 범위와 지역변수

#### 변수 scope

- 변수의 참조가 유효 범위를 **변수의 유효 범위(scope)**라 함
- **지역 유효(local scope)**와 **전역 유효(global scope)** 범위로 나눌 수 있음.
- 다른 파일에서 전역 변수를 사용하고 싶다면 미리 **extern** 으로 선언 필요!  
파일 내에서만 전역 변수를 사용하고 싶다면 **static** 으로 선언 필요!  
다음 이미지가 잘 설명해주고 있었음.



### 지역변수

- 함수 또는 블록에서 선언된 변수 (전역변수와는 **범위**에서 차이가 있음!)
- **\*\*지역변수는 선언 후 초기화하지 않으면 쓰레기값이 저장되므로 주의!**
- **\*\*선언 문장이 실행되는 시점부터 메모리에 할당됨**
- **\*\*지역변수는 함수나 블록이 종료 시 순간 메모리에서 자동으로 제거 됨!!!**  
>> 자동변수(automatic variable)라고도 부름
- 자료형 앞에 키워드 **auto** 를 붙여야하나 생략 가능하여 일반적으로 없음.

## 실습예제 12-1

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

void sub(int param);

int main(void)
{
    //지역변수 선언
    auto int n = 10;
    printf("%d\n", n);

    //m, sum은 for 문 내부의 블록 지역변수
    for (int m = 0, sum = 0; m < 3; m++)
    {
        sum += m;
        printf("Wt%d %d\n", m, sum);
    }

    printf("%d\n", n); //n 참조 가능
    //printf("%d %d\n", m, sum); //m, sum 참조 불가능

    //함수호출
    sub(20);

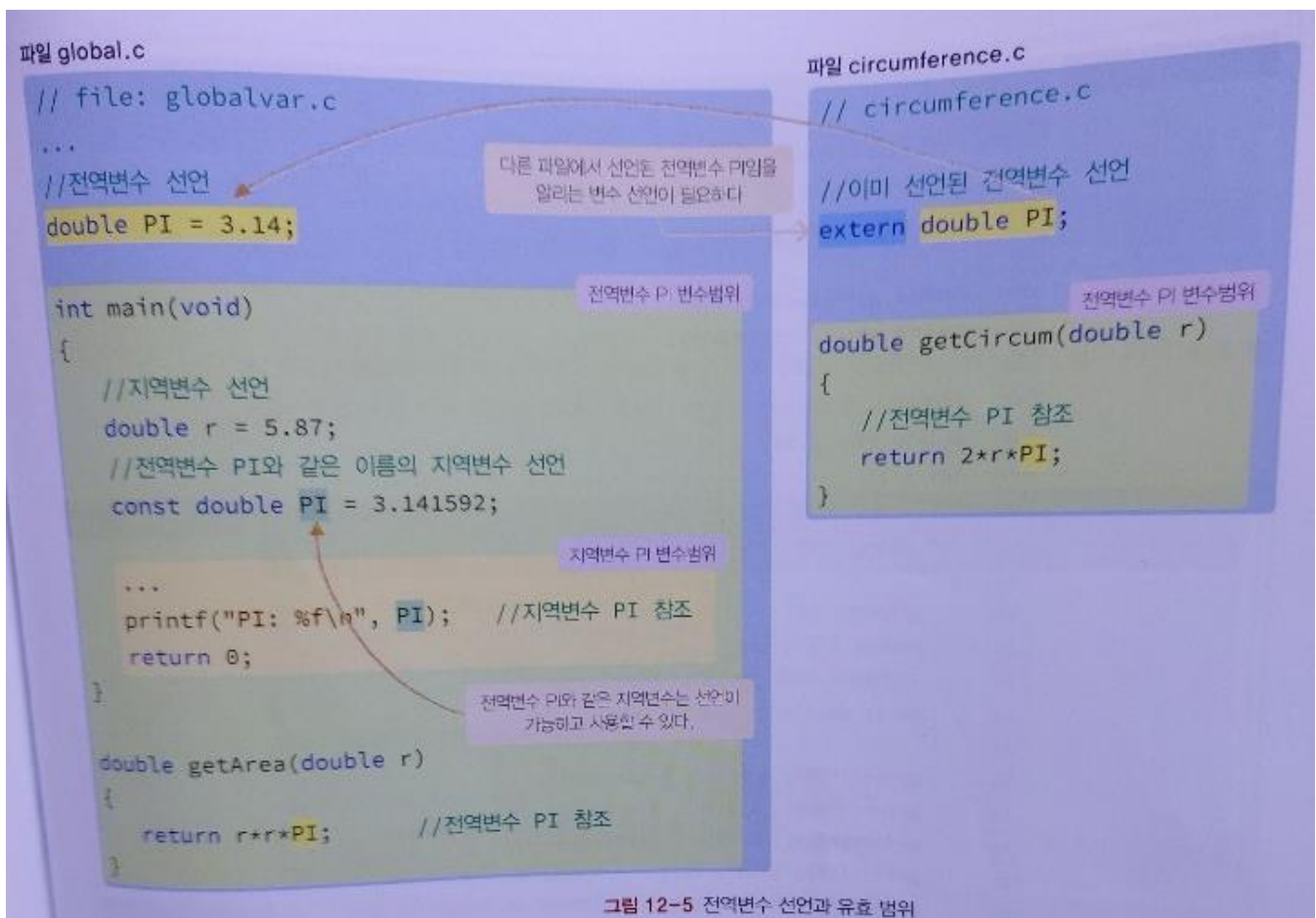
    return 0;
}

//매개변수인 param도 지역 변수와 같이 사용
void sub(int param)
{
    //지역변수 local
    auto int local = 100;
    printf("Wt%d %d\n", param, local); //param과 local 참조 가능
    //printf("%d\n", n); //n 참조 불가능
}
```

## 전역변수와 extern

### 전역변수(global variable)

- 함수 외부에서 선언되는 변수 > 외부변수라고도 부름
- 일반적으로 프로젝트의 모든 함수나 블록에서 참조 가능!
- 선언 시 자동으로 자료형에 맞게 초기값이 지정됨
- 키워드 **extern** 을 사용하여 프로젝트의 다른 파일에서도 참조 가능!
  - \*키워드 **extern** 은 다음 이미지의 **circumference.c** 처럼 “이 변수는 이미 선언된 전역변수이다”라는 것을 알려주는 역할.
- \*\*전역변수는 어디에서든 선언할 수 있지만 절차지향인 C 언어 특성상 선언위치가 참조위치보다 아래라면 extern 을 사용해주어야 한다. 이럴 경우 굉장히 복잡해질 수 있으니 가능한 제한적으로 사용하는 것(파일 상단 배치)이 바람직함!



다음 예제들을 통해 **extern** 의 기능을 확인할 수 있었다.



## 실습예제 12-2

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

double getArea(double);
double getCircum(double);

//전역변수 선언
double PI = 3.14;
int gi;

int main(void)
{
    //지역변수 선언
    double r = 5.87;
    //전역변수 PI와 같은 이름의 지역변수 선언
    const double PI = 3.141592;

    printf("면적: %.2fWn", getArea(r));
    printf("둘레1: %.2fWn", 2 * PI * r);
    printf("둘레2: %.2fWn", getCircum(r));
    printf("PI: %fWn", PI);           //지역변수 PI 참조
    printf("gi: %dWn", gi);          //전역변수 gi 기본 값

    return 0;
}

double getArea(double r)
{
    return r * r * PI;               //전역변수 PI 참조
}
```

## 실습예제 12-3

```
// circumference.c
//이미 외부에서 선언된 전역변수임을 알리는 선언
extern double PI;

double getCircum(double r)
{
    //extern double PI;           //함수 내부에서만 참조 가능
    return 2 * r * PI;           //전역변수 PI 참조
}
```

## 피보나치 수의 출력

### Lab 12-1

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

    printf("1 ");
    fibonacci(prev_number, number);
    printf("\n");

    return 0;
}

void fibonacci(int prev_number, int number)
{
    //정적 지역변수 i
    static int i = 1;

    //전역변수 count와 함수의 정적 지역변수를 비교
    while (i++ < count)
    {
        //지역변수
        int next_num = prev_number + number;
        prev_number = number;
        number = next_num;
        printf("%d ", next_num);
        fibonacci(prev_number, number);
    }
}
```

## 12-2 정적 변수와 레지스터 변수

### 기억부류와 레지스터 변수

#### 기억부류 키워드 : auto, register, static, extern

- 변수의 메모리 영역이 결정되고 메모리 할당과 제거 시기를 결정하는 키워드
- **auto** 는 자료형의 기억부류 기본값으로 생략이 가능함.
- **extern** 은 외부 변수를 불러오는 기능
- **\*\*자료형 앞에 키워드를 붙여주는 방식으로 사용.**
- 각 기억부류의 유효 범위를 정리해보면 다음과 같다.

| 기억부류 종류  | 전역 | 지역 |
|----------|----|----|
| auto     | X  | ○  |
| register | X  | ○  |
| static   | ○  | ○  |
| extern   | ○  | X  |

#### 키워드 register

- 일반적으로 변수는 메모리에 할당되고 연산 시에 CPU 내부 레지스터에 불러들여 연산을 수행한다.
- 레지스터 변수는 **저장공간이 CPU 내부의 레지스터에 할당되는 변수이다.**
- 일반 메모리보다 빠르게 참조 가능하므로 **처리 속도가 빠르다.**
- **\*\*일반 메모리에 할당되는 변수가 아니므로 주소연산자 & 사용 불가능**
- \*레지스터 변수로 선언하더라도 레지스터가 모자라면 일반 지역변수로 할당됨
- 초기값을 지정해주지 않으면 쓰레기 값 저장(= 일반 지역변수)

## 실습예제 12-4

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

int main(void)
{
    //레지스터 지역변수 선언
    register int sum = 0;

    //메모리에 저장되는 일반 지역변수 선언
    int max;
    printf("양의 정수 입력 >> ");
    scanf("%d", &max);

    //레지스터 블록 지역변수 선언
    for (register int count = 1; count <= max; count++)
        sum += count;

    printf("합: %d\n", sum);

    return 0;
}
```

## 정적 변수

### 키워드 static

- 자료형 앞에 키워드 **static** 을 넣어 정적변수(static variable)를 선언할 수 있다.
- 프로그램이 시작되면 메모리에 할당 ~ 종료 시 제거
  - > 전역변수의 특징과 유사
- 초기값 자료형에 맞게 자동 지정 (0 or 'W0')
- \*초기화는 상수로만 가능하고 단 한번만 수행

## 정적 지역변수

- 함수나 블록 내에서 정적으로 선언되는 변수
- \*함수나 블록을 종료해도 메모리에서 제거되지 않음!
- \*단, 범위는 지역변수의 특성을 가짐

다음 실습예제를 보면 이해가 빠름

### 실습예제 12-5

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

void increment(void); //함수원형

int main(void)
{
    //자동 지역변수
    for (int count = 0; count < 3; count++)
        increment();    //3번 함수호출

    return 0;
}

void increment(void)
{
    static int sindex = 1; //정적 지역변수
    auto int aindex = 1;   //자동 지역변수

    printf("정적 지역변수 sindex: %2d,Wt", sindex++);
    printf("자동 지역변수 aindex: %2dWn", aindex++);
}
```

정적 지역변수의 값이 1로 초기화 되지 않고 유지되는 것을 알 수 있다.

|                    |                   |
|--------------------|-------------------|
| 정적 지역변수 sindex: 1, | 자동 지역변수 aindex: 1 |
| 정적 지역변수 sindex: 2, | 자동 지역변수 aindex: 1 |
| 정적 지역변수 sindex: 3, | 자동 지역변수 aindex: 1 |

## 정적 전역변수

- 함수 외부에서 정적으로 선언되는 변수
- \*\*선언된 파일 내부에서만 참조가능
  - > 다른 파일에서 참조 불가(extern 불가능)
- \*프로그램이 크고 복잡하면 전역변수의 사용은 원하지 않은 수정과 같은 부작용의 위험성이 항상 존재하므로 가급적이면 전역변수의 사용은 자제하는 것이 좋음.
  - > 적절한 상황에 사용!!

다음 예제(12-6, 12-7)를 통해서 외부 파일에서 정적 전역변수를 사용하지 못 하는 것을 확인가능

### 실습예제 12-6

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

//정적 전역변수 선언
static int svar;
//전역변수 선언
int gvar;

//함수 원형
void increment();
void testglobal();
//void teststatic();

int main(void)
{
    for (int count = 1; count <= 5; count++)
        increment();
    printf("함수 increment()가 총 %d번 호출되었습니다.\n", svar);

    testglobal();
    printf("전역 변수: %d\n", gvar);
    //teststatic();

    return 0;
}

//함수 구현
void increment()
{
    svar++;
}
```

## 실습예제 12-7

```
// file: gfunc.c

void teststatic()
{
    //정적 전역변수는 선언 및 사용 불가능
    //extern svar;
    //svar = 5;
}

void testglobal()
{
    //전역변수는 선언 및 사용 가능
    extern gvar;
    gvar = 10;
}
```

## LAB 지역변수와 정적변수의 사용

### Lab 12-2

```
//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;
}
```



## 12-3 메모리 영역과 변수 이용

### 메모리 영역

- 메인 메모리의 영역은 프로그램 실행 과정에서 데이터, 스택(stack), 힙(heap) 영역으로 나뉨
- 각 메모리 영역은 변수의 유효범위(scope), 생존기간(life time)에 결정적 역할을 함
- 변수는 기억부류()에 따라 할당되는 메모리 공간이 달라짐

### 데이터 영역

- 전역변수와 정적변수가 할당되는 저장공간
- 메모리 주소가 낮은 값에서 높은 값으로 저장 장소 할당
- 메모리 영역 크기 고정

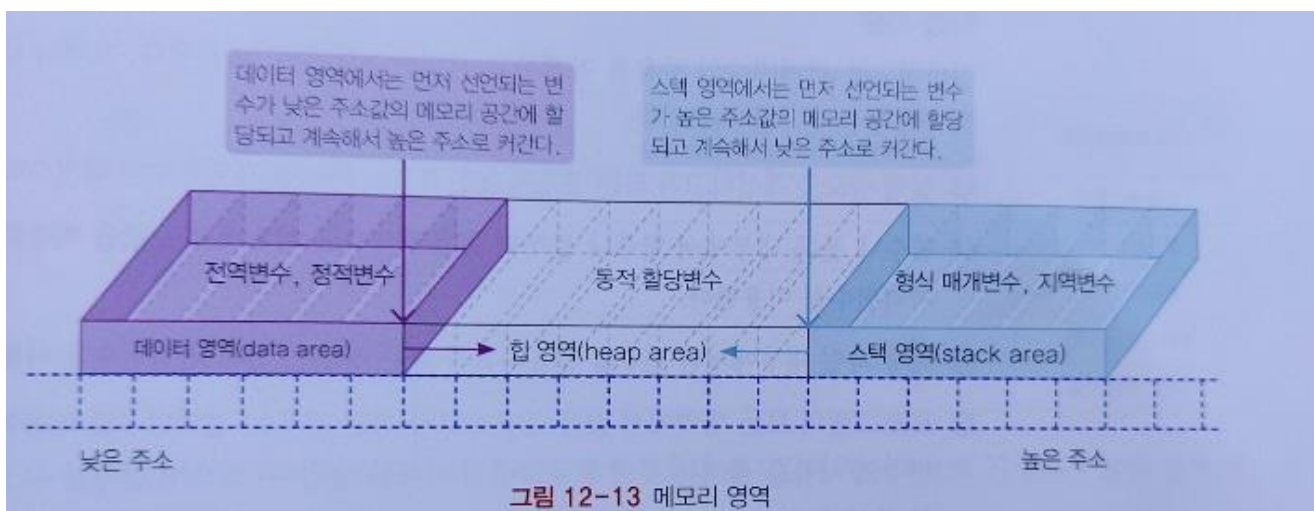
### 힙 영역

- 동적 할당되는 변수가 할당되는 저장공간
- 데이터 영역과 스택 영역 사이에 위치.
- 메모리 주소가 낮은 값에서 높은 값으로 사용하지 않는 공간이 동적으로 할당됨.

### 스택 영역

- 함수 호출에 의한 형식 매개변수, 함수 내부의 지역변수가 할당되는 저장공간.
- \*\*메모리 주소가 높은 값에서 낮은 값으로 저장 장소 할당.
- 함수 호출과 종료에 따라 메모리 할당, 제거의 작업이 반복됨  
-> 크기가 동적으로 변함

이러한 메모리 영역의 특성들을 다음 이미지가 잘 설명해줌



## 변수의 이용

### 이용 기준

- 일반적으로 전역변수 사용을 자제, 지역변수를 주로 이용
- 각 특성에 맞게 **종류, 유효범위, 생존기간**을 고려하여 변수를 사용하는 것이 중요!
- 다음 표에서 **변수의 종류, 유효범위에 따른 생존기간, 초기값**들을 확인가능!

표 12-2 변수의 종류

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

표 12-3 변수의 유효 범위

| 구분 | 종류      | 메모리할당 시기  | 동일 파일 외부 함수에서의 이용 | 다른 파일 외부 함수에서의 이용 | 메모리제거 시기  |
|----|---------|-----------|-------------------|-------------------|-----------|
| 전역 | 전역변수    | 프로그램시작    | ○                 | ○                 | 프로그램종료    |
|    | 정적 전역변수 | 프로그램시작    | ○                 | ×                 | 프로그램종료    |
| 지역 | 정적 지역변수 | 프로그램시작    | ×                 | ×                 | 프로그램종료    |
|    | 레지스터 변수 | 함수(블록) 시작 | ×                 | ×                 | 함수(블록) 종료 |
|    | 자동 지역변수 | 함수(블록) 시작 | ×                 | ×                 | 함수(블록) 종료 |

표 12-4 변수의 초기값

| 지역, 전역 | 종류      | 자동 저장되는 기본 초기값                      | 초기값 저장          |
|--------|---------|-------------------------------------|-----------------|
| 전역     | 전역변수    | 자료형에 따라 0이나 '\0'<br>또는 NULL 값이 저장됨. | 프로그램 시작 시       |
|        | 정적 전역변수 |                                     |                 |
| 지역     | 정적 지역변수 | 쓰레기값이 저장됨.                          | 함수나 블록이 실행될 때마다 |
|        | 레지스터 변수 |                                     |                 |
|        | 자동 지역변수 |                                     |                 |

다음 예제(12-8, 12-9)를 통해 각종 변수의 실행 흐름을 알 수 있었다.

## 실습예제 12-8

```
// file: storageclass.c
#include <stdio.h>
void infunction(void);
void outfunction(void);
/* 전역변수*/
int global = 10;
/* 정적 전역변수*/
static int sglobal = 20;
int main(void)
{
    auto int x = 100; /* main() 함수의 자동 지역변수*/
    printf("%d, %d, %d\n", global, sglobal, x);
    infunction(); outfunction();
    infunction(); outfunction();
    infunction(); outfunction();
    printf("%d, %d, %d\n", global, sglobal, x);

    return 0;
}

void infunction(void)
{
    /* infunction() 함수의 자동 지역변수*/
    auto int fa = 1;
    /* infunction() 함수의 정적 지역변수*/
    static int fs;

    printf("WtWt%d\n", ++global, ++sglobal, fa, ++fs);
}
```

## 실습예제 12-9

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

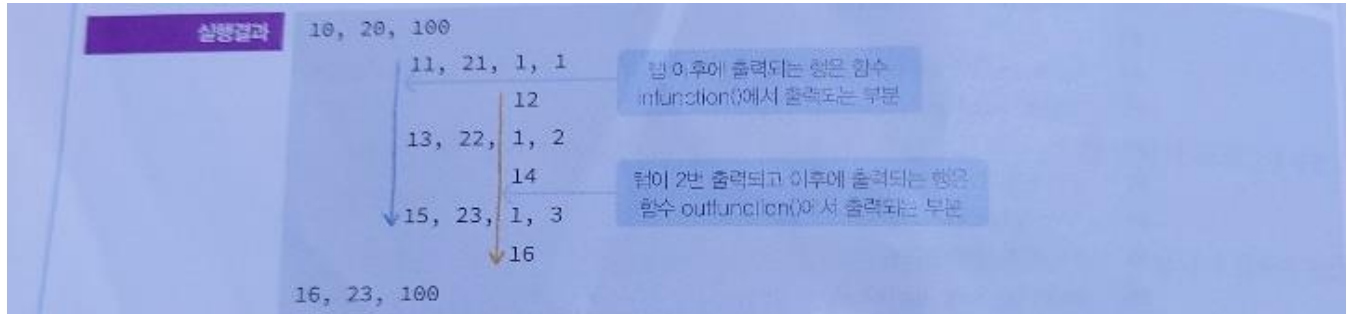
void outfunction()
{
    extern int global, sglobal;

    printf("WtWt%d\n", ++global);

    //외부 파일에 선언된 정적 전역변수이므로 실행 시 오류
    //printf("%d\n", ++sglobal);
}
```

## 실행 결과

- 실행 결과는 다음과 같았고 교재의 그림이 설명이 잘 되어 있어 가져왔다.
- **\*\*전역변수의 사용에서 원하지 않는 과정이 실행되는 것을 주의해야할 것 같다.**



## LAB 은행계좌의 입출금 구현

### Lab 12-3

```
// 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("=====W\n");
    printf("%46d\n", total);
    save(50000);
    withdraw(30000);
    save(60000);
    withdraw(20000);
    printf("=====W\n");

    return 0;
}

//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);
}
```

# CHAPTER 13

## 구조체와 공용체

---

## 13-1 구조체와 공용체

### 구조체 개념과 정의

#### 구조체 개념

- 연관성이 있는 서로 다른 개별적인 자료형의 변수들을 하나의 단위로 묶은 새로운 자료형을 구조체(structure)라 함
- 연관된 멤버로 구성되는 통합 자료형으로 대표적인 유도 자료형
- 기존 자료형으로 새로 만들어진 자료형을 유도 자료형이라 함

#### 구조체 정의

- 구조체를 사용하려면 먼저 구조체를 정의해야 함.
- 앞에 키워드 **struct** 를 붙이고 중괄호로 원하는 멤버(구조체 멤버 or 필드)를 넣으면 됨.
- 구조체 멤버로는 일반 변수, 포인터 변수, 배열, 다른 구조체 변수 및 구조체 포인터 허용

#### 구조체 정의 방법

- 다음과 같이 기술.
- **youraccount** 와 같이 구조체 정의와 변수 선언을 동시에 해줄 수도 있음!

```
struct account
{
    char name[12]; //계좌주 이름
    int actnum;      //계좌번호
    double balance; //잔고
};
```

```
struct account
{
    char name[12]; //계좌주 이름
    int actnum;      //계좌번호
    double balance; //잔고
} youraccount;
```

### 구조체 변수 선언과 초기화

#### 구조체 변수 선언 및 초기화 방법

- 다음과 같이 중괄호 안에 멤버를 쉼표로 구분하여 순서대로 기술하면 됨

```
struct account mine = { "홍길동", 1001, 300000 };
struct account yours;
```

## 구조체의 멤버 접근 연산자 .와 변수 크기

- 선언된 구조체형 변수는 **접근연산자 .**를 사용하여 멤버를 참조  
**Ex)** mine.actnum = 1002;
- 일반적으로 컴파일러는 시스템의 효율성을 위하여 **구조체 크기를 산술적인 구조체의 크기보다 크게 할당할 수 있다.**

시스템은 정보를 4 바이트 or 8 바이트 단위로 전송 처리하므로 이에 맞게 메모리를 할당하다 보면 중간에 사용하지 않는 바이트를 삽입할 수 있음

따라서 실제 구조체의 크기는 멤버의 크기의 합보다 크거나 같게된다.

>> **sizeof() 사용 시 주의!**

## 실습예제 13-1

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

//은행 계좌를 위한 구조체 정의
struct account
{
    char name[12]; //계좌주 이름
    int actnum;      //계좌번호
    double balance; //잔고
};

int main(void)
{
    //구조체 변수 선언 및 초기화
    struct account mine = { "홍길동", 1001, 300000 };
    struct account yours;

    strcpy(yours.name, "이동원");
    //strcpy_s(yours.name, 12, "이동원"); //가능
    //yours.name = "이동원"; //오류
    yours.actnum = 1002;
    yours.balance = 500000;

    printf("구조체 크기: %d\n", sizeof(mine));
    printf("%s %d %.2f\n", mine.name, mine.actnum, mine.balance);
    printf("%s %d %.2f\n", yours.name, yours.actnum, yours.balance);

    return 0;
}
```



## 구조체 활용

### 구조체 멤버로 사용되는 구조체

- 구조체 멤버로 구조체 변수를 넣을 수 있다.

#### Ex) date 구조체를 account 구조체 멤버로 넣은 예시

```
struct date
{
    int year;      //년
    int month;     //월
    int day;       //일
};
struct account
{
    struct date open;    //계좌 개설일자
    char name[12];       //계좌주 이름
    int actnum;          //계좌번호
    double balance;      //잔고
};
```

## 실습예제 13-2

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

//날짜를 위한 구조체
struct date
{
    int year;      //년
    int month;     //월
    int day;       //일
};

//은행계좌를 위한 구조체
struct account
{
    struct date open;    //계좌 개설일자
    char name[12];       //계좌주 이름
    int actnum;          //계좌번호
    double balance;      //잔고
};

int main(void)
{
    struct account me = { { 2018, 3, 9 }, "홍길동", 1001, 300000 };
    printf("구조체 크기: %d\n", sizeof(me));
    printf("[%d, %d, %d]\n", me.open.year, me.open.month, me.open.day);
    printf("%s %d %.2f\n", me.name, me.actnum, me.balance);
    return 0;
}
```

## 구조체 변수의 대입과 동등비교

- 동일한 구조체형의 변수는 대입문이 가능

멤버마다 대입할 필요 없이 변수 대입으로 한번에 모든 멤버 대입 가능

\*그러나 동등 비교(hong == bae)는 불가능.. 비교를 위해서는 멤버 하나 하나를 비교해야함.

### Ex) 동일한 구조체형 변수 대입 예제

```
//학생을 위한 구조체
struct student
{
    int snum;           //학번
    char* dept;         //학과 이름
    char name[12];      //학생 이름
};
struct student hong = { 201800001, "컴퓨터정보공학과", "홍길동" };
struct student bae;

bae = hong;
```

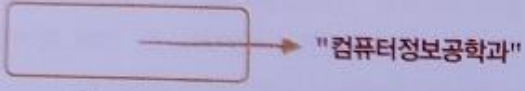
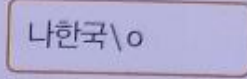
- 예제를 통해서 구조체 대입과 비교의 흐름을 알 수 있었다.
- 그리고 다음 내용들도 확인할 수 있었다.

\*char 포인터에는 주소를 저장 가능하나 scanf()나 memcpy() strcpy()로는 저장이 불가능

\*char 배열에는 문자열 상수를 대입 불가능

\*\*결론적으로 포인터는 상수를 다루는 경우, 배열은 문자열을 저장하고 수정할 경우에 사용!  
교재에 이러한 특징을 잘 비교정리한 표가 있었음.

표 13-1 char 포인터와 char 배열의 비교

| char 포인터  | char 배열  |
|---|--|
| char *dept; //학과 이름   | char name[12]; //학생 이름   |
| char *dept = "컴퓨터정보공학과";  | char name[12] = "나한국";   |
|  |  |
| 변수 dept   | 변수 name[12]  |
| 변수 dept는 포인터로 단순히 문자열 상수를 다루는 경우 효과적  | 변수 name은 배열로 12바이트 공간을 가지며 문자열을 저장하고 수정 등이 필요한 경우 효과적                                |
| dept = "컴퓨터정보공학과";  | name = "나한국"; //오류   |
| 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                      |
| strcpy(dept, "컴퓨터정보공학과"); //오류  | strcpy(name, "배상문");   |
| scanf("%s", dept); //오류   | scanf("%s", name);   |

## 실습예제 13-3

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

int main(void)
{
    //학생을 위한 구조체
    struct student
    {
        int snum;           //학번
        char* dept;         //학과 이름
        char name[12];      //학생 이름
    };
    struct student hong = { 201800001, "컴퓨터정보공학과", "홍길동" };
    struct student na = { 201800002 };
    struct student bae = { 201800003 };

    //학생이름 입력
    scanf("%s", na.name);
    //na.name = "나한국"; //오류
    //scanf("%s", na.dept); //오류

    na.dept = "컴퓨터정보공학과";
    bae.dept = "기계공학과";
    memcpy(bae.name, "배상문", 7);
    strcpy(bae.name, "배상문");
    strcpy_s(bae.name, 7, "배상문");

    printf("[%d, %s, %s]\n", hong.snum, hong.dept, hong.name);
    printf("[%d, %s, %s]\n", na.snum, na.dept, na.name);
    printf("[%d, %s, %s]\n", bae.snum, bae.dept, bae.name);

    struct student one;
    one = bae;
    if (one.snum == bae.snum)
        printf("학번이 %d로 동일합니다.\n", one.snum);
    //if ( one == bae ) //오류
    if (one.snum == bae.snum && !strcmp(one.name, bae.name) && !strcmp(one.dept, bae.dept))
        printf("내용이 같은 구조체입니다.\n");

    return 0;
}
```

## 공용체 활용

### 공용체 개념

- 동일한 저장 장소에 여러 자료형을 저장하는 방법
- 1 인 주차장과 비슷한 개념. 차를 한 대만 주차할 수 있지만 크기에 따라 다양한 종류의 차를 한 공간에 주차할 수 있음

### union 을 사용한 정의 및 변수 선언

- 공용체(union)은 서로 다른 자료형의 값을 동일한 저장공간에 저장하는 자료형.
- 공용체 선언 방법은 구조체 선언과 비슷하며 키워드를 struct 에서 union 으로 바꿔주면 된다.
- \*공용체 변수의 크기는 멤버 중 가장 큰 자료형의 크기로 정해짐
- 마지막에 저장된 단 하나의 멤버 자료값만을 저장
- 구조체와 같이 typedef 로 새로운 자료형으로 정의 가능
- 공용체의 초기화 값은 공용체 멤버의 초기값으로만 저장 가능함

### Ex) 공용체 초기화 예시

```
union data
{
    char ch;        //문자형
    int cnt;         //정수형
    double real;    //실수형
} data1; //data1은 전역변수
union data data2 = { 'A' }; //첫 멤버인 char 형으로만 초기화 가능
```

### 공용체 멤버 접근

- 구조체와 동일하게 접근연산자 .를 사용
- 모든 공용체 멤버의 참조는 가능하나 마지막에 저장한 멤버가 아니면 의미가 없으므로 주의!

다음 예제를 통해 공용체 선언, 접근의 전체적인 흐름을 알 수 있음.

## 실습예제 13-4

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

//유니온 구조체를 정의하면서 변수 data1도 선언한 문장
union data
{
    char ch;          //문자형
    int cnt;          //정수형
    double real;      //실수형
} data1; //data1은 전역변수

int main(void)
{
    union data data2 = { 'A' }; //첫 멤버인 char형으로만 초기화 가능
    //union data data2 = {10.3}; //컴파일 시 경고 발생
    union data data3 = data2;    //다른 변수로 초기화 가능

    printf("%d %d\n", sizeof(union data), sizeof(data3));

    //멤버 ch에 저장
    data1.ch = 'a';
    printf("%c %d %f\n", data1.ch, data1.cnt, data1.real);
    //멤버 cnt에 저장
    data1.cnt = 100;
    printf("%c %d %f\n", data1.ch, data1.cnt, data1.real);
    //멤버 real에 저장
    data1.real = 3.156759;
    printf("%c %d %f\n", data1.ch, data1.cnt, data1.real);

    return 0;
}
```

## LAB 도시의 이름과 위치를 표현하는 구조체

### Lab 13-1

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

//지구 위치 구조체
struct position
{
    double latitude;    //위도
    double longitude;   //경도
};

int main(void)
{
    //도시 정보 구조체
    struct city
    {
        char* name;           //이름
        struct position place; //위치
    };
    struct city seoul, newyork;

    seoul.name = "서울";
    seoul.place.latitude = 37.33;
    seoul.place.longitude = 126.58;

    newyork.name = "뉴욕";
    newyork.place.latitude = 40.8;
    newyork.place.longitude = 73.9;

    printf("[%s] 위도= %.1f 경도= %.1fWn",
           seoul.name, seoul.place.latitude, seoul.place.longitude);
    printf("[%s] 위도= %.1f 경도= %.1fWn",
           newyork.name, newyork.place.latitude, newyork.place.longitude);

    return 0;
}
```

## 13-2 자료형 재정의

### 자료형 재정의 typedef

#### typedef 구문

- **typedef** 는 이미 사용되는 자료 유형을 다른 새로운 자료형 이름으로 재정의할 수 있도록 해주는 키워드
- \*자료형 재정의는 프로그램의 시스템 간 호환성과 편의성을 위해 필요 (프로그램마다 자료형의 크기 등이 다를 수 있음.)
- typedef 도 일반 변수와 같이 사용범위에 제한이 있음 (전역 변수 / 지역 변수)

#### 실습예제 13-5

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

//함수 외부에서 정의된 자료형은 이후 파일에서 사용 가능
typedef unsigned int budget;

int main(void)
{
    //새로운 자료형 budget 사용
    budget year = 24500000;

    //함수 내부에서 정의된 자료형은 이후 함수내부에서만 사용 가능
    typedef int profit;
    //새로운 자료형 profit 사용
    profit month = 4600000;

    printf("올 예산은 %d, 이달의 이익은 %d 입니다.\n", year, month);

    return 0;
}

void test(void)
{
    //새로운 자료형 budget 사용
    budget year = 24500000;

    //profit은 이 함수에서는 사용 불가, 오류 발생
    //profit year;
}
```

## 구조체 자료형 재정의

### struct 를 생략한 새로운 자료형

- 구조체를 미리 정의하고 struct 앞에 typedef 키워드를 사용하여 재정의함으로 편리하게 코딩할 수 있다.

Ex)

```
struct date
{
    int year;      //년
    int month;     //월
    int day;       //일
};
typedef struct date date;
```

- 구조체를 정의함과 동시에 typedef 키워드로 재정의도 가능하다.  
아래 예시에서 date 는 변수명이 아니라 새로운 자료형을 의미함
- \*\*위 코드와 아래 코드는 같은 기능을 함

Ex)

```
typedef struct date
{
    int year;      //년
    int month;     //월
    int day;       //일
} date;
```



## 실습예제 13-6

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

struct date
{
    int year;      //년
    int month;     //월
    int day;       //일
};

//struct date 유형을 간단히 date 형으로 사용하기 위한 구문
typedef struct date date;

int main(void)
{
    //구조체를 정의하면서 바로 자료형 software 로 정의하기 위한 구문
    typedef struct
    {
        char title[30]; //제목
        char company[30]; //제작회사
        char kinds[30]; //종류
        date release;    //출시일
    } software;

    software vs = { "비주얼스튜디오 커뮤니티", "MS", "통합개발환경", { 2018, 8, 29 } };

    printf("제 품 명 : %s\n", vs.title);
    printf("회 사   : %s\n", vs.company);
    printf("종 류   : %s\n", vs.kinds);
    printf("출시일 : %d. %d. %d\n", vs.release.year, vs.release.month, vs.release.day);

    return 0;
}
```

## LAB 영화 정보를 표현하는 구조체

### Lab 13-2

```
// 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;
}
```

## 13-3 구조체와 공용체의 포인터와 배열

### 구조체 포인터

#### 포인터 변수 선언

- 포인터는 각각의 자료형 저장 공간의 주소를 저장하듯이 구조체 포인터는 구조체의 주소값을 저장할 수 있는 변수
- 선언 방법은 이전에 배운 포인터 변수와 같음

#### Ex) 구조체 포인터 선언 예시

```
struct lecture
{
    char name[20]; //강좌명
    int type;      //강좌구분 0: 교양, 1: 일반선택, 2: 전공필수, 3: 전공선택
    int credit;    //학점
    int hours;     //시수
} lecture;

lecture os = { "운영체제", 2, 3, 3 };
lecture* p = &os;
```

#### 포인터 변수의 구조체 멤버 접근 연산자 ->

- 구조체 포인터 멤버 접근연산자 ->는 **p->name** 과 같이 사용
- **(\*p).name** 으로도 사용 가능
- **\*\*간접연산자(\*)보다 접근연산자(.)의 우선순위가 빠르므로 주의!**  
-> **\*p.name** 과 **(\*p).name** 은 다르다는 것을 이해해야함.

#### 실습예제 13-7 에서 전체적인 흐름을 파악할 수 있음

```
printf("%12c %10s %5d %5d\n", *c.name, lectype[c.type], c.credit, c.hours);
```

위 부분에서 \*c.name 은 \*(c.name)을 의미하게 되므로 "C 프로그래밍"에서 첫 글자만 참조하게 되니 주의!

## 실습예제 13-7

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

struct lecture
{
    char name[20]; //강좌명
    int type;      //강좌구분 0: 교양, 1: 일반선택, 2: 전공필수, 3: 전공선택
    int credit;    //학점
    int hours;     //시수
};
typedef struct lecture lecture;

//제목을 위한 문자열
char* head[] = { "강좌명", "강좌구분", "학점", "시수" };
//강좌 종류를 위한 문자열
char* lectype[] = { "교양", "일반선택", "전공필수", "전공선택" };

int main(void)
{
    lecture os = { "운영체제", 2, 3, 3 };
    lecture c = { "C프로그래밍", 3, 3, 4 };
    lecture* p = &os;

    printf("구조체크기: %d, 포인터크기: %d\n\n", sizeof(os), sizeof(p));
    printf("%10s %12s %6s %6s\n", head[0], head[1], head[2], head[3]);
    printf("%12s %10s %5d %5d\n", p->name, lectype[p->type], p->credit, p->hours);

    //포인터 변경
    p = &c;
    printf("%12s %10s %5d %5d\n", (*p).name, lectype[(*p).type], (*p).credit, (*p).hours);
    printf("%12c %10s %5d %5d\n", *c.name, lectype[c.type], c.credit, c.hours);

    return 0;
}
```

## 공용체 포인터

- 공용체 변수도 포인터 변수 사용이 가능
- 구조체 포인터와 동일하게 접근연산자(->)를 사용

### Ex) 공용체 포인터 정의, 선언 및 참조 예시

```
union data
{
    char ch;
    int cnt;
    double real;
}udata value, * p;
p = &value;
p->ch = 'a'; //value.ch = 'a';와 동일한 문장
```

## 실습예제 13-8

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

int main(void)
{
    //유니온 union data 정의
    union data
    {
        char ch;
        int cnt;
        double real;
    };

    //유니온 union data를 다시 자료형 udata로 정의
    typedef union data udata;

    //udata 형으로 value와 포인터 p 선언
    udata value, * p;

    p = &value;
    p->ch = 'a';
    printf("%c %c\n", p->ch, (*p).ch);
    p->cnt = 100;
    printf("%d ", p->cnt);
    p->real = 3.14;
    printf("%.2f\n", p->real);

    return 0;
}
```

## 구조체 배열

### 구조체 배열 변수 선언

- 구조체 변수가 여러 개 필요할 때 구조체 배열을 선언하여 이용할 수 있음

#### Ex) 구조체 변수 배열 선언 및 초기화 예시

```
struct lecture
{
    char name[20]; //강좌명
    int type;      //강좌구분
    int credit;    //학점
    int hours;     //시수
};

lecture course[] = { { "인간과 사회", 0, 2, 2 },
    { "경제학개론", 1, 3, 3 },
    { "자료구조", 2, 3, 3 },
    { "모바일프로그래밍", 2, 3, 4 },
    { "고급 C프로그래밍", 3, 3, 4 } };
```

- 구조체 포인터 변수에 배열이름을 대입 가능
- 다음 예제와 실습예제 13-9 를 통해 전체적인 흐름을 확인할 수 있다.

#### Ex) 구조체 포인터 변수에 배열의 주소를 저장 후 참조 예시

```
struct lecture
{
    char name[20]; //강좌명
    int type;      //강좌구분
    int credit;    //학점
    int hours;     //시수
};

lecture course[] = { { "인간과 사회", 0, 2, 2 },
    { "경제학개론", 1, 3, 3 },
    { "자료구조", 2, 3, 3 },
    { "모바일프로그래밍", 2, 3, 4 },
    { "고급 C프로그래밍", 3, 3, 4 } };

char* lectype[] = { "교양", "일반선택", "전공필수", "전공선택" };
lecture *p = course;

int main(void)
{
    for (int i = 0; i < arysize; i++)
        printf("%16s %10s %5d %5d\n", p[i].name, lectype[p[i].type], p[i].credit, p[i].hours);
    return 0;
}
```

## 실습예제 13-9

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

struct lecture
{
    char name[20]; //강좌명
    int type;      //강좌구분
    int credit;    //학점
    int hours;     //시수
};
typedef struct lecture lecture;

char* lectype[] = { "교양", "일반선택", "전공필수", "전공선택" };
char* head[] = { "강좌명", "강좌구분", "학점", "시수" };

int main(void)
{
    //구조체 lecture의 배열 선언 및 초기화
    lecture course[] = { { "인간과 사회", 0, 2, 2 },
        { "경제학개론", 1, 3, 3 },
        { "자료구조", 2, 3, 3 },
        { "모바일프로그래밍", 2, 3, 4 },
        { "고급 C프로그래밍", 3, 3, 4 } };

    int arysize = sizeof(course) / sizeof(course[0]);

    printf("배열크기: %d\n\n", arysize);
    printf("%12s %12s %6s %6s\n", head[0], head[1], head[2], head[3]);
    printf("=====Wn");

    for (int i = 0; i < arysize; i++)
        printf("%16s %10s %5d %5d\n", course[i].name,
            lectype[course[i].type], course[i].credit, course[i].hours);

    return 0;
}
```

## LAB 영화 정보를 표현하는 구조체의 배열

### Lab 13-3

```
// 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("=====W\n");
    for (int i = 0; i < 3; i++)
        printf("[%8s] %6s %d\n",
            box[i].title, box[i].director, box[i].attendance);

    return 0;
}
```



# CHAPTER 14

## 함수와 포인터 활용

---

## 14-1 함수의 인자전달 방식

### 값에 의한 호출과 참조에 의한 호출

#### 함수에서 값의 전달

- C 언어는 함수의 인자 전달 방식이 기본적으로 **값에 의한 호출(call by value)** 방식
- **값에 의한 호출** 방식은 함수 호출 시 실인자의 값이 형식인자에 복사되어 저장  
-> \*\*값만 복사되므로 외부 변수를 복사할뿐 직접적으로 수정불가(예제 14-1 참고)

#### 실습예제 14-1

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

void increase(int origin, int increment);

int main(void)
{
    int amount = 10;
    //amount가 20 증가하지 않음
    increase(amount, 20);
    printf("%d\n", amount);

    return 0;
}

void increase(int origin, int increment)
{
    origin += increment;
}
```

예제 실행 결과 amount 가 증가하지 않고 10 이 출력됨을 확인할 수 있다.

## 함수에서 주소의 전달

- C 언어에서 **포인터**를 매개변수로 사용하면 함수로 전달된 **실인자의 주소**를 사용하여 그 변수를 참조할 수 있다. => 즉. **참조에 의한 호출(call by reference)**
- \*예제 14-2의 결과를 보면 **amount 값이 증가되어 30으로 출력됨을 알 수 있다.**

## 실습예제 14-2

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

void increase(int *origin, int increment);

int main(void)
{
    int amount = 10;
    //&amount: amount의 주소로 호출
    increase(&amount, 20);
    printf("%d\n", amount);

    return 0;
}

void increase(int *origin, int increment)
{
    // *origin은 origin이 가리키는 변수 자체
    *origin += increment; //그러므로 origin이 가리키는 변수 값이 20 증가
}
```

## 배열의 전달

### 배열이름으로 전달

- \*함수의 매개변수로 배열을 전달하는 것은 배열의 첫 원소를 참조 매개변수로 전달하는 것과 동일.

이러한 특징 때문에 실인자로 전달된 배열의 크기를 알 수 없다.

따라서 함수에 사용 시 배열 크기를 두 번째 인자로 사용하는 것을 추천

> \*매개변수를 배열형태(double ary[])로 기술해도 단순히 포인터 변수(double \*ary)로 인식하기 때문!!

- 함수원형에서 매개변수는 배열 이름을 생략하고 (double [])와 같이 기술 가능  
Ex) double sum(double [], int n);

### 실습예제 14-3

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

#define ARYSIZE 5
double sum(double g[], int n); //배열 원소 값을 모두 더하는 함수원형

int main(void)
{
    //배열 초기화
    double data[] = { 2.3, 3.4, 4.5, 6.7, 9.2 };

    //배열원소 출력
    for (int i = 0; i < ARYSIZE; i++)
        printf("%5.1f", data[i]);
    puts("");

    //배열 원소 값을 모두 더하는 함수호출
    printf("합: %5.1f\n", sum(data, ARYSIZE));

    return 0;
}

//배열 원소 값을 모두 더하는 함수정의
double sum(double ary[], int n)
{
    double total = 0.0;
    for (int i = 0; i < n; i++)
        total += ary[i];

    return total;
}
```

## 다양한 배열원소 참조 방법

- 기본적으로 **간접연산자(\*)**를 사용하는 방식.
- 함수 헤더에 **Int ary[]** 를 **int \* ary** 로 대체 가능
- 각종 **변형된 형태**는 **연산자의 우선순위를** 생각하면 이해할 수 있음.  
후위/전위연산자(++)는 보통 간접연산자(\*) 보다 연산 순위가 우선이므로 주의

## 실습예제 14-4

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

int sumary(int *ary, int SIZE); //int sumary(int ary[], int SIZE)도 가능

int main(void)
{
    int point[] = { 95, 88, 76, 54, 85, 33, 65, 78, 99, 82 };
    //배열크기 구하기
    int aryLength = sizeof(point) / sizeof(int);

    //address는 포인터 변수이며 point는 배열 상수
    int *address = point;
    //메인에서 직접 배열 합 구하기
    int sum = 0;
    for (int i = 0; i < aryLength; i++)
        sum += *(point + i);
        //sum += *(point++); //오류발생
        //sum += *(address++); //가능
    printf("메인에서 구한 합은 %d\n", sum);

    //함수호출하여 합 구하기
    printf("함수sumary() 호출로 구한 합은 %d\n", sumary(point, aryLength));
    printf("함수sumary() 호출로 구한 합은 %d\n", sumary(&point[0], aryLength));
    printf("함수sumary() 호출로 구한 합은 %d\n", sumary(address, aryLength));

    return 0;
}

int sumary(int *ary, int SIZE) //int sumary(int ary[], int SIZE)도 가능
{
    int sum = 0;
    for (int i = 0; i < SIZE; i++)
    {
        //sum += ary[i]; //가능
        //sum += *(ary + i); //가능
        sum += *ary++;
        //sum += *(ary++); //가능
    }

    return sum;
}
```

## 배열크기 계산방법

- 배열 크기 = sizeof(배열이름) / sizeof(배열원소)
- 배열 크기를 이용해서 배열 원소들의 합을 구하는 등의 함수 구현이 가능(예제 14-5)

## 실습예제 14-5

```
// file: arrayfunction
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
void readarray(double[], int); //배열 원소값을 모두 표준입력 받는 함수원형
void printarray(double[], int); //배열 원소값을 모두 출력하는 함수원형
double sum(double[], int); //배열 원소값을 모두 더하는 함수원형
int main(void)
{
    double data[5];
    int arraysize = sizeof(data) / sizeof(data[0]);
    printf("실수 5개의 값을 입력하세요. \n");
    readarray(data, arraysize);
    printf("\n입력한 자료값은 다음과 같습니다.\n");
    printarray(data, arraysize);
    printf("함수에서 구한 합은 %.3f 입니다.\n", sum(data, arraysize));

    return 0;
}
//배열 원소값을 모두 표준입력 받는 함수
void readarray(double data[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("data[%d] = ", i);
        scanf("%lf", &data[i]); //(data + i)로도 가능
    }
    return;
}
//배열 원소값을 모두 출력하는 함수
void printarray(double data[], int n)
{
    for (int i = 0; i < n; i++)
        printf("data[%d] = %.2lf ", i, *(data + i));
    printf("\n");
    return;
}
//배열 원소값을 모두 더하는 함수
double sum(double data[], int n)
{
    double total = 0;
    for (int i = 0; i < n; i++)
        total += data[i]; /*(data + i)
    return total;
}
```

## 다차원 배열 전달

- 다차원 배열을 인자로 이용하는 경우, 함수원형과 함수정의의 헤더에서 첫 번째 대괄호 내부의 크기를 제외한 다른 모든 크기는 반드시 기술하여야함.  
-> 배열의 첫 원소를 참조 매개변수로 전달하기 때문.
- 이차원 배열의 행, 열의 수는 배열 크기를 구하는 방법과 동일  
행 :  $(\text{sizeof}(x)) / \text{sizeof}(x[0])$   
열 :  $(\text{sizeof}(x[0]) / \text{sizeof}(x[0][0]))$

### Ex) 다차원 배열 전달 및 각 행의 수 구하기 예시

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

double x[][3] = { { 1, 2, 3 }, { 7, 8, 9 }, { 4, 5, 6 }, { 10, 11, 12 } };

int main(void)
{
    int rowsize = sizeof(x) / sizeof(x[0]);
    int colsize = sizeof(x[0]) / sizeof(x[0][0]);
    printf( "행 : %d, 열 : %d", rowsize, colsize);

    return 0;
}
```

예제 14-6 에서 배열값을 모두 필요로 하는 예시를 통해 전체적 흐름을 확인함.

## 실습예제 14-6

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

//2차원 배열값을 모두 더하는 함수원형
double sum(double data[][3], int, int);
//2차원 배열값을 모두 출력하는 함수원형
void printarray(double data[][3], int, int);

int main(void)
{
    //4 x 3 행렬
    double x[][3] = { { 1, 2, 3 }, { 7, 8, 9 }, { 4, 5, 6 }, { 10, 11, 12 } };

    int rowsize = sizeof(x) / sizeof(x[0]);
    int colsize = sizeof(x[0]) / sizeof(x[0][0]);
    printf("2차원 배열의 자료값은 다음과 같습니다.\n");
    printarray(x, rowsize, colsize);
    printf("2차원 배열 원소합은 %.3lf 입니다.\n", sum(x, rowsize, colsize));

    return 0;
}

//배열값을 모두 출력하는 함수
void printarray(double data[][3], int rowsize, int colsize)
{
    for (int i = 0; i < rowsize; i++)
    {
        printf("%d행 원소: ", i + 1);
        for (int j = 0; j < colsize; j++)
            printf("x[%d][%d] = %5.2lf ", i, j, data[i][j]);
        printf("\n");
    }
    printf("\n");
}

//배열값을 모두 더하는 함수
double sum(double data[][3], int rowsize, int colsize)
{
    double total = 0;
    for (int i = 0; i < rowsize; i++)
        for (int j = 0; j < colsize; j++)
            total += data[i][j];

    return total;
}
```



## 가변인자

### 가변인자가 있는 함수머리

- 함수에서 인자의 수와 자료형이 결정되지 않은 함수 인자 방식을 가변인자(variable argument)라고 함  
사용방법 (...)으로 기술. 자세한 건 예시 참고 (오류 포함)

### Ex) 가변인자 사용 방법

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

double fun1(int count, ..., int n); // 오류, 마지막이 고정적일 수 없음

double fun2(...); // 오류, 처음부터 가변인자일 수 없음.
```

### 가변인자가 있는 함수 구현

- 가변인자를 구현하려면 4 단계가 필요
  1. 가변인자 선언
  2. 가변인자 처리 시작
  3. 가변인자 얻기
  4. 가변인자 처리 종료
- \* 헤더파일 stdarg.h 삽입 필요!

### Ex) 가변인자가 있는 함수 구현 (흐름에 따른 주석 참고)

```
double avg(int numargs, ...)
{
    //1. 가변인자 변수 선언
    va_list argp;
    //2. numargs 이후의 가변인자 처리 시작
    va_start(argp, numargs);

    double total = 0;
    for (int i = 0; i < numargs; i++)
        //3. 지정하는 double 형으로 가변인자 얻기
        total += va_arg(argp, double);

    //4. 가변인자 처리 종료
    va_end(argp);
    return total / numargs;
}
```

예제 14-7 에서 `stdarg.h` 헤더파일 삽입, 가변인자 처리 흐름을 확인할 수 있다.

## 실습예제 14-7

```
// 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
double avg(int numargs, ...)
{
    //가변인자 변수 선언
    va_list argp;

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

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

    //가변인자 처리 종료
    va_end(argp);

    return total / numargs;
}
```

## LAB 함수에서 배열 활용

동일한 기능을 하는 배열의 변형된 형태를 확인할 수 있음.

### Lab 14-1

```
// 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
}
```

## 14-2 포인터 전달과 반환

### 매개변수와 반환으로 포인터 사용

#### 주소연산자 &

- 함수에서 매개변수를 포인터로 이용하면 참조에 의한 호출(call by reference)이 됨.

예제 14-8 에서 주소연산자(&)를 이용해 주소값을 인자로 호출하는 흐름을 확인

#### 실습예제 14-8

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

void add(int *, int, int);

int main(void)
{
    int m = 0, n = 0, sum = 0;

    printf("두 정수 입력: ");
    scanf("%d %d", &m, &n);
    add(&sum, m, n);
    printf("두 정수 합: %d\n", sum);

    return 0;
}

void add(int *psum, int a, int b)
{
    *psum = a + b;
}
```

## 주소값 반환

- 함수의 결과를 포인터로 반환하는 경우 (예제 14-9)
- \*\*함수 종료 시 메모리에서 제거되는 변수(지역변수)의 주소값 반환은 오류가 생길 수 있다.  
따라서 지역변수의 주소값을 반환하지 않는 것이 바람직

## 실습예제 14-9

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

int * add(int *, int, int);
int * subtract(int *, int, int);
int * multiply(int, int);

int main(void)
{
    int m = 0, n = 0, sum = 0, diff = 0;

    printf("두 정수 입력: ");
    scanf("%d %d", &m, &n);

    printf("두 정수 합: %d\n", *add(&sum, m, n));
    printf("두 정수 차: %d\n", *subtract(&diff, m, n));
    printf("두 정수 곱: %d\n", *multiply(m, n));

    return 0;
}

int * add(int *psum, int a, int b)
{
    *psum = a + b;
    return psum;
}

int * subtract(int *pdiff, int a, int b)
{
    *pdiff = a - b;
    return pdiff;
}

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

## 상수를 위한 const 사용

### 키워드 const

- 수정을 원하지 않는 함수의 인자 앞에 사용하는 키워드
- 원하지 않는 포인터 인자의 잘못된 수정을 예방하는 방법

### Ex) const 인자의 이용에 따른 수정 시도 시 오류 예시

```
void multiply(double *result, const double *a, const double *b)
{
    *result = *a * *b;
    //다음 2 문장 오류발생
    *a = *a + 1;
    *b = *b + 1;
}
```

### 예제 14-10의 키워드 const를 사용한 multiply 함수와

사용하지 않은 devideandincrement 함수의 차이에서 키워드 const의 의미를 확인

### 실습예제 14-10

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

void multiply(double *, const double *, const double *);
void devideandincrement(double *, double *, double *);

int main(void)
{
    double m = 0, n = 0, mult = 0, dev = 0;

    printf("두 실수 입력: ");
    scanf("%lf %lf", &m, &n);
    multiply(&mult, &m, &n);
    devideandincrement(&dev, &m, &n);
    printf("두 실수 곱: %.2f, 나눗: %.2f\n", mult, dev);
    printf("연산 후 두 실수: %.2f, %.2f\n", m, n);

    return 0;
}
```

```
//매개변수 포인터 a, b가 가리키는 변수의 내용을 곱해 result가 가리키는 변수에 저장
//매개변수 포인터 a, b가 가리키는 변수의 내용은 수정하지 못함
void multiply(double *result, const double *a, const double *b)
{
    *result = *a * *b;
    //오류발생
    //*a = *a + 1;
    //*b = *b + 1;
}

//매개변수 포인터 a, b가 가리키는 변수의 내용을 나누어 result가 가리키는 변수에 저장한 후
//a, b가 가리키는 변수의 내용을 모두 1 증가시킴
void devideandincrement(double *result, double *a, double *b)
{
    *result = *a / *b;
    ++*a; //++(*a)이므로 a가 가리키는 변수의 값을 1 증가
    (*b)++; //b가 가리키는 변수의 값을 1 증가, *b++와는 다름
}
```

## 함수의 구조체 전달과 반환

### 복소수를 위한 구조체

- 복소수 연산에 이용되는 함수를 만들기 앞서 사용할 구조체 **complex** 를 다음과 같이 정의 (실수부 **real** 과 허수부 **img** 로 구성)

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

### 인자와 반환형으로 구조체 사용

- 함수 `paircomplex1()`는 인자인 복소수의 켤레 복소수를 구하여 반환하는 함수로 정의
- 구조체는 함수의 인자와 반환값으로 이용 가능!! 다음은 값에 의한 호출(call by value) 방식을 이용한 함수 예시이다.

#### Ex) 구조체를 call by value 방식으로 반환값으로 이용하는 함수 예시

```
complex paircomplex1(complex com)
{
    com.img = -com.img;
    return com;
}
```

- 값에 의한 호출(call by value)방식을 이용하면 값을 복사하는 것이기에 복사할 크기가 크다면 시간이 많이 소요된다. \*따라서 참조에 의한 호출(call by reference)방식이 더 빠르다

#### Ex) 구조체 포인터를 인자로 사용해 call by reference 방식을 이용한 함수

```
void paircomplex2(complex *com)
{
    com->img = -com->img;
}
```

예제 14-11 에서 위 함수들과 구조체를 이용한 흐름을 확인할 수 있다.



## 실습예제 14-11

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

//복소수를 위한 구조체
struct complex
{
    double real; //실수
    double img;  //허수
};
//complex를 자료형으로 정의
typedef struct complex complex;

void printcomplex(complex com);
complex paircomplex1(complex com);
void paircomplex2(complex *com);

int main(void)
{
    complex comp = { 3.4, 4.8 };
    complex pcomp;

    printcomplex(comp);
    pcomp = paircomplex1(comp);
    printcomplex(pcomp);
    paircomplex2(&pcomp);
    printcomplex(pcomp);

    return 0;
}

//구조체 자체를 인자로 사용
void printcomplex(complex com)
{
    printf("복소수(a + bi) = %5.1f + %5.1fi Wn", com.real, com.img);
}

//구조체 자체를 인자로 사용하여 처리된 구조체를 다시 반환
complex paircomplex1(complex com)
{
    com.img = -com.img;
    return com;
}

//구조체 포인터를 인자로 사용
void paircomplex2(complex *com)
{
    com->img = -com->img;
}
```

## LAB 책 정보를 표현하는 구조체 전달

### Lab 14-2

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

typedef struct book
{
    char title[50];
    char author[50];
    int ISBN;
} book;

void print(book *b);

int main()
{
    book python = {"파이썬웹프로그래밍", "김석훈", 2398765};
    book java;
    strcpy(java.title, "절대자바");
    strcpy(java.author, "강환수");
    java.ISBN = 123987;
    print(&java);
    print(&python);

    return 0;
}

void print(book *b)
{
    printf("제목: %s, ", b->title);
    printf("저자: %s, ", b->author);
    printf("ISBN: %d\n", b->ISBN);
}
```

## 14-3 함수 포인터와 void 포인터

### 함수 포인터

#### 함수 주소 저장 변수

- 포인터의 장점은 다른 변수를 참조하여 읽거나 쓰는 것도 가능하다는 것.
- 이를 함수에 이용해서 하나의 함수 이름으로 필요에 따라 여러 함수를 사용하면 편리할 수 있다
- 결론적으로 함수 포인터 변수는 함수의 주소값을 저장하게 된다.  
\*반환형, 인자목록의 수와 각각의 자료형이 일치해야함

#### Ex) 함수 포인터 선언 및 대입 방법 예시

```
void add(double*, double, double);
void subtract(double*, double, double);

int main(void)
{
    void(*pf)(double*, double, double) = NULL; //함수 포인터 pf를 선언
    pf = add; //add() 함수를 함수 포인터 pf에 저장, &add도 가능 pf = add();는 오류 발생
    pf = subtract; //subtract() 함수를 함수 포인터 pf에 저장, &subtract도 가능

    return 0;
}
```

#### 함수 포인터를 이용한 함수 호출

- 일반적으로 함수를 호출하듯이 다음과 같이 함수 포인터를 호출하면 됨.  
-> pf(&result, m, n);

예제 14-12 에서 위 모든 과정의 흐름을 확인할 수 있다.

## 실습예제 14-12

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

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

int main(void)
{
    //함수 포인터 pf를 선언
    void(*pf)(double*, double, double) = NULL;

    double m, n, result = 0;
    printf("+, -를 수행할 실수 2개를 입력하세요. >> ");
    scanf("%lf %lf", &m, &n);

    //사칙연산을 수행
    pf = add; //add() 함수를 함수 포인터 pf에 저장, &add도 가능
    pf(&result, m, n); //add(&result, m, n);
    printf("pf = %p, 함수 add() 주소= %pWn", pf, add);
    printf("더하기 수행: %lf + %lf == %lfWnWn", m, n, result);

    pf = subtract; //subtract() 함수를 함수 포인터 pf에 저장, &subtract도 가능
    pf(&result, m, n); //subtract(&result, m, n);
    printf("pf = %p, 함수 subtract() 주소= %pWn", pf, subtract);
    printf("빼기 수행: %lf - %lf == %lfWnWn", m, n, result);

    return 0;
}

// x + y 연산 결과를 z가 가리키는 변수에 저장하는 함수
void add(double *z, double x, double y)
{
    *z = x + y;
}

// x - y 연산 결과를 z가 가리키는 변수에 저장하는 함수
void subtract(double *z, double x, double y)
{
    *z = x - y;
}
```

## 함수 포인터 배열

### 함수 포인터 배열 개념

- 함수 포인터 배열은 함수 포인터가 원소인 배열.

### 함수 포인터 배열 선언

- 함수 포인터 배열은 다음과 같이 선언하고 초기화함

### Ex) 함수 포인터 배열 선언 및 초기화 예시

```
void add(double*, double, double);
void subtract(double*, double, double);
void multiply(double*, double, double);
void devide(double*, double, double);

void(*fpary[4])(double*, double, double) = { add, subtract, multiply, devide };
```

- 배열의 특성 상 반복문과 함께 사용하는 방식에서 자주 사용하는 것으로 보임

### Ex) 함수 포인터 배열 사용 예시

```
for (int i = 0; i < 4; i++)
{
    fpary[i](&result, m, n);
    printf("%.2lf %c %.2lf == %.2lf\n", m, op[i], n, result);
}
```

예제 14-13 에서 위 내용들의 흐름을 확인할 수 있었다.

## 실습예제 14-13

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

//함수원형
void add(double*, double, double);
void subtract(double*, double, double);
void multiply(double*, double, double);
void devide(double*, double, double);
```

```

int main(void)
{
    char op[4] = { '+', '-', '*', '/' };
    //함수 포인터 선언하면서 초기화 과정
    void(*fpary[4])(double*, double, double) = { add, subtract, multiply, devide };

    double m, n, result;
    printf("사칙연산을 수행할 실수 2개를 입력하세요. >> ");
    scanf("%lf %lf", &m, &n);
    //사칙연산을 배열의 첨자를 이용하여 수행
    for (int i = 0; i < 4; i++)
    {
        fpary[i](&result, m, n);
        printf("%.2lf %c %.2lf == %.2lf\n", m, op[i], n, result);
    }

    return 0;
}

// x + y 연산 결과를 z가 가리키는 변수에 저장하는 함수
void add(double *z, double x, double y)
{
    *z = x + y;
}

// x - y 연산 결과를 z가 가리키는 변수에 저장하는 함수
void subtract(double *z, double x, double y)
{
    *z = x - y;
}

// x * y 연산 결과를 z가 가리키는 변수에 저장하는 함수
void multiply(double *z, double x, double y)
{
    *z = x * y;
}

// x / y 연산 결과를 z가 가리키는 변수에 저장하는 함수
void devide(double *z, double x, double y)
{
    *z = x / y;
}

```

## void 포인터

### void 포인터 개념

- 주소값이란 참조를 시작하는 주소에 불과하며 자료형을 알아야 참조할 범위와 내용을 해석할 방법을 알 수 있는 것이다.  
자료형을 구체적으로 표현하는 것이 일반적이지만, void 를 이용해 모든 자료형을 쓸 수 있도록 할 수 있다.
- 이러한 점을 이용한 것이 void 포인터이다.  
정리하면 void 포인터는 자료형을 무시하고 주소값만을 다룬다. 그리고 void 특성 상 모든 자료형의 주소를 저장할 수 있는 만능 포인터로 사용할 수 있다.
- void 포인터에는 일반 변수 포인터는 물론 배열과 구조체 심지어 함수 주소도 담을 수 있다.  
예제 14-14 에서 void 포인터가 주소를 담는 흐름을 확인할 수 있다.

### 실습예제 14-14

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

void myprint(void)
{
    printf("void 포인터, 신기하네요!\n");
}

int main(void)
{
    int m = 10;
    double d = 3.98;

    void *p = &m; //m의 주소 만을 저장
    printf("주소 %d\n", p); //주소 값 출력
    //printf("%d\n", *p); //오류발생

    p = &d; //d의 주소 만을 저장
    printf("주소 %d\n", p);

    p = myprint; //함수 myprint()의 주소 만을 저장
    printf("주소 %d\n", p);

    return 0;
}
```

## void 포인터 활용

- void 포인터는 모든 주소를 저장할 수 있지만 기리키는 변수를 참조하거나 수정이 불가능  
\*\*변수를 참조하기 위해서 자료형 변환이 필요!!
- \*\*함수 포인터는 일반적으로 해당 포인터가 함수의 포인터라는 것을 숨기기위해 쓴다고 함.  
후에 전문가가 되면 자신의 기술을 지키기 위해 사용하기도 한다 함
- 함수 포인터의 형변환(함수 자료형(\*) (void))를 이해하는 것이 중요!

### Ex) void 포인터 참조 예시

```
void myprint(void)
{
    printf("void 포인터, 신기하네요!\n");
}
int m = 10;
void *p = &m;
printf("p 참조 정수: %d\n", *(int *)p); //int * 로 변환
p = myprint;
printf("p 참조 함수 실행 : ");
((void (*)(void)) p)(); //함수 포인터인 void (*)(void) 로 변환하여 호출 ()
```

## 실습예제 14-15

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

void myprint(void)
{
    printf("void 포인터, 신기하네요!\n");
}

int main(void)
{
    int m = 10;
    double d = 3.98;
    char str[][20] = { { "C 언어," }, { "재미있네요!" } };
    void *p = &m;
    printf("p 참조 정수: %d\n", *(int *)p); //int * 로 변환
    p = &d;
    printf("p 참조 실수: %.2f\n", *(double *)p); //double * 로 변환
    p = myprint;
    printf("p 참조 함수 실행 : ");
    ((void (*)(void)) p)(); //함수 포인터인 void (*)(void) 로 변환하여 호출 ()
    p = str;
    //열이 20인 이차원 배열로 변환하여 1행과 1행의 문자열 출력
    printf("p 참조 2차원 배열: %s %s\n", (char (*)[20])p, (char (*)[20])p + 1);
    printf("str 참조 2차원 배열: %s %s\n", str, str+1);
    return 0;
}
```



## LAB 함수 포인터 배열의 활용

### Lab 14-3

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

int add(int a, int b);
int mult(int a, int b);
int subtr(int a, int b);

int main(void)
{
    int (*pfunary[3])(int, int);
    pfunary[0] = add;
    pfunary[1] = mult;
    pfunary[2] = subtr;

    char *ops = "+-";
    char op;
    while (op = *ops++)
        switch (op) {
            case '+': printf("%c 결과: %d\n", op, pfunary[0](3, 5));
                      break;
            case '-': printf("%c 결과: %d\n", op, pfunary[2](3, 5));
                      break;
            case '*': printf("%c 결과: %d\n", op, pfunary[1](3, 5));
                      break;
        }

    return 0;
}

int add(int a, int b)
{
    return a + b;
}
int mult(int a, int b)
{
    return a * b;
}
int subtr(int a, int b)
{
    return a - b;
}
```

# CHAPTER 15

함수 포인터와

void 포인터

## 15-1 파일 기초

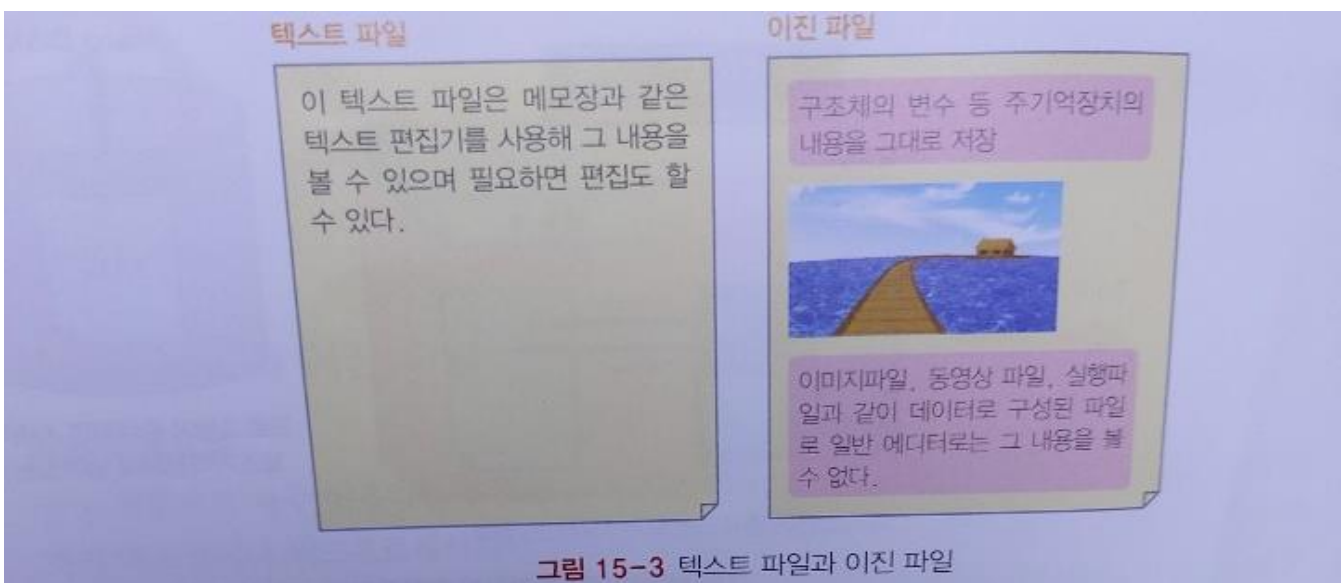
### 텍스트 파일과 이진 파일

#### 파일의 필요성

- 프로그램에서 출력을 파일에 한다면 파일이 생성!
- 입력을 파일에서 입력하면 파일 입력!
- \*\*주 기억장치의 메모리 공간을 사용하는 변수는 프로그램 종료 시 사라진다.  
그러나 보조기억장치인 디스크에 저장되는 파일(file)은 직접 삭제하지 않는 한 프로그램이 종료되더라도 계속 저장 가능.
- \*\*결론적으로 지속적으로 결과를 저장하고 싶다면 파일에 그 내용을 저장해야 한다.

#### 텍스트 파일과 이진파일

- 파일은 텍스트 파일(text file)과 이진파일(binary file) 두 가지 유형으로 나뉜.
- 텍스트 파일 : 문자 기반의 파일로서 내용이 아스키코드(ascii code)와 같은 문자 코드값으로 저장됨. (실수와 정수와 같은 내용도 문자 형식으로 변환되어 저장)
- 이진 파일 : 그림 파일, 동영상 파일, 실행 파일과 같이 각각의 목적에 알맞은 자료가 이진 형태(binary format)로 저장되는 파일
  - + 컴퓨터 내부 형식으로 저장됨, 변환을 거치지 않고 그대로 파일에 기록되므로 입출력 속도가 텍스트 파일에 비해 빠름!
  - \*이진 파일은 텍스트 편집기(메모장)로는 볼 수 없다.
  - \*이진 파일의 자료는 그 내용을 이미 알고 있는 특정한 프로그램에 의해 인지될 때 의미있음!



## 입출력 스트림

- **자료의 입력과 출력은 자료의 이동**이라고 볼 수 있음! (자료가 이동하려면 **이동 경로가 필요**)  
입출력 시 이동 통로가 바로 **입출력 스트림(io stream)**
- **\*\*다른 곳에서 프로그램으로 들어오는 경로가 입력 스트림(input stream).**  
자료의 출발 장소가 **자료 원천부(data source)**로 이 부분이 **키보드이면 표준입력.**  
(\*파일이면 파일로부터 자료를 읽을 수 있고, 터치스크린이면 터치 정보, 네트워크면 네트워크를 통해 자료가 전달되는 것!)  
+)함수 **scanf()**는 **표준 입력 스트림에서 자료를 읽을 수 있는 함수.**
- **\*\*반대로 프로그램에서 다른 곳으로 나가는 경로가 출력 스트림(output stream).**  
자료의 도착 장소가 **자료 목적부(data destination)**로 이 부분이 **콘솔이면 표준출력.**  
(\*파일이면 파일에 원하는 값을 쓸 수 있고, 프린터면 프린터에 출력물이 나오고, 네트워크면 네트워크출력이 되어 다른 곳으로 자료가 이동되는 것!)  
+)함수 **printf()**는 **표준출력 스트림으로 자료를 보낼 수 있는 함수**

## 파일 스트림 이해

- 프로그램에서 보조기억장치에 파일로 정보를 저장하거나 파일에서 정보를 참조하려면 파일(file)에 대한 **파일 스트림(file stream)**을 먼저 연결해야 함.
- **파일 스트림**이란 보조기억장치의 파일과 프로그램을 연결하는 전송경로.
- **파일 입력 스트림(file input stream)**과 **파일 출력 스트림(file output stream)**으로 나뉨

## 파일, 스트림 내용 간단 정리

### \*파일(file)

- 프로그램 종료 후에도 지속적으로 자료를 저장하고 싶다면 파일로 저장해야함.  
파일의 종류는 텍스트 파일 / 이진 파일로 나뉘고 각각 특징을 가짐.

### \*스트림(stream)

- 즉. 파일 이동을 위해서는 이동 경로인 스트림(stream)이 필요
- 각 스트림은 자료에 맞게 기능을 하도록 도와줌

## 파일 스트림 열기

### 함수 `fopen()`으로 파일 스트림 열기

- 프로그램에서 특정한 파일과 파일 스트림을 연결하기 위해서 `fopen()` 또는 `fopen_s()` 이용
- 헤더파일 `stdio.h`에 정의되어 있음.

### `fopen()`과 `fopen_s()` 함수의 원형은 다음과 같음

#### 함수 `fopen()`과 `fopen_s()` 함수원형

```
FILE * fopen(const char * _Filename, const char * _Mode);  
errno_t fopen_s(FILE ** _File, const char * _Filename, const char * _Mode);
```

- 함수 `fopen()`은 파일명 `_Filename`의 파일 스트림을 모드 `_Mode`로 연결하는 함수이며, 스트림 연결에 성공하면 파일 포인터를 반환하며, 실패하면 `NULL`을 반환한다.
- 함수 `fopen_s()`는 스트림 연결에 성공하면 첫 번째 인자인 `_File`에 파일 포인터가 저장되고 정수 0을 반환하며, 실패하면 함수를 반환한다. 현재 Visual C++에서는 함수 `fopen_s()`의 사용을 권장하고 있다.

그림 15-7 함수 `fopen()`과 `fopen_s()` 함수원형

- **FILE**은 `stdio.h`에 정의되어 있는 구조체 유형!
- 구조체 **FILE**은 파일을 표현하는 C 언어의 유도 자료형이며 구성은 다음 이미지와 같다.

```
struct _iobuf {  
    char *_ptr;  
    int _cnt;  
    char *_base;  
    int _flag;  
    int _file;  
    int _charbuf;  
    int _bufsiz;  
    char *_tmpfname;  
};  
typedef struct _iobuf FILE;
```

- 함수 **fopen()**은 인자가 파일이름과 파일열기 모드  
>> 파일 스트림 연결 성공 시 **파일 포인터** 반환, 실패 시 **NULL** 반환
- 함수 **fopen\_s()**는 파일 스트림 연결 성공 시 정수 **0** 반환, 실패 시 **양수** 반환.
- \*\*텍스트 파일 열기 모드 종류  
 "r" : **읽기모드** || 쓰기 불가능  
 "w" : **쓰기모드** || 파일 어디서든 쓰기 가능, 읽기 불가능.  
 "a" : **추가모드** || 파일 마지막에 추가적으로 쓰는 것만 가능, 읽기 불가능

### Ex) fopen(), fopen\_s() 사용 예시 (fopen\_s()는 주석)

```
FILE *f;

if ( (f = fopen(fname, "w")) == NULL )
//if (fopen_s(&f, fname, "w") != 0)
{
    printf("파일이 열리지 않습니다.\n");
    exit(1);
};
```

### 함수 fclose()로 파일 스트림 닫기

- **fopen()**으로 연결한 파일 스트림을 닫는 기능
- 내부적으로 파일 스트림 연결에 할당된 자원을 반납하고, 파일과 메모리 사이에 있던 버퍼의 내용을 모두 지우는 역할
- 성공하면 **0** 실패 시 **EOF** 반환

fclose() 함수 원형과 사용방법은 다음과 같음

```
int fclose(FILE * _File);

fclose(f);
```

## 출력 스트림을 이용한 파일 생성

- 예제 15-1 을 통해서 출력 스트림과 `fprintf()`를 이용해서 “basic.txt”에 출력하는 프로그램 흐름을 알 수 있다.
- `exit()`는 함수를 강제로 종료하는 기능  
정상 종료 시 인자 값 0, 0 이 아닌 값은 정상 종료가 아님을 OS 에 알리는 의미로 사용됨  
`stdlib.h` 헤더파일 삽입 필요.
- 최초 실행 시 프로젝트 폴더에 “basic.txt” 파일이 생성됨을 알 수 있다.
- 마지막에 `fclose(f)` 중요!

## 실습예제 15-1

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

int main()
{
    char *fname = "basic.txt"; //파일이름
    FILE *f; //파일 포인터

    //파일에 쓰려는 자료
    char name[30] = "강미정";
    int point = 99;

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

    //파일 "basic.txt"에 쓰기
    fprintf(f, "이름이 %s인 학생의 성적은 %d 입니다.\n", name, point);
    fclose(f);
    //표준출력 콘솔에 쓰기
    printf("이름이 %s인 학생의 성적은 %d 입니다.\n", name, point);
    puts("프로젝트 폴더에서 파일 basic.txt를 메모장으로 열어 보세요.");

    return 0;
}
```

## LAB 파일 “myinfo.txt”를 열어 간단한 정보를 출력

### Lab 15-1

```
// 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"에 쓰기
    fprintf(f, "전화번호: %s, 주소: %s, 나이: %d\n", tel, add, age);

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

    return 0;
}
```



## 15-2 텍스트 파일 입출력

### 함수 fprintf()와 fscanf()

- 텍스트 파일에 자료를 쓰거나(fprint) 읽기(fsacnf) 위해 쓰는 함수
- `stdio.h` 헤더파일 삽입
- 첫 번째 인자에 `stdin`, `stdout`, `stderr` 를 이용하면 각각 표준입력, 표준 출력, 표준 에러로 이용 가능 (\*예시 15-2 참고)  
(`*stdin`, `stdout`, `stderr` 은 C 언어가 제공하는 표준파일)  
`fprintf(stdout, "제어문자열", ...)` 와 `printf("제어문자열", ...)`은 같은 표준 출력 기능!!
- 각 함수 원형과 인자에 대한 설명은 다음과 같다.

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

`_File` : 서식화된 입출력 스트림의 목적지 파일

`_Format` : 입출력 제어 문자열(형식 지정) ex) `"%d %d"`

`...` : 여러 개의 입출력될 변수 또는 상수 \*예시 15-2 참고

### 표준입력 자료를 파일에 쓰기

- 이번 파트는 예제 15-2 를 보면서 이해하는 것이 빨랐다.
- 먼저 `scanf()`로 표준입력을 받고 이를 변수에 저장한 후에 `fprintf()`로 파일 출력.  
그 후에 파일 입력을 위해 `fscanf()`을 사용 후 `fprintf()`를 표준 출력으로 사용한 것을 확인  
\*\*중요한 부분을 먼저 보면 다음과 같다.

#### 1. `scanf()`표준입력 후 `fprinf()` 파일 출력

```
scanf("%s %d %d", name, &point1, &point2);  
//scanf_s("%s%d%d", name, 30, &point1, &point2);  
fprintf(f, "%d %s %d %d\n", ++cnt, name, point1, point2);
```

## 2. fscanf() 파일 입력 후 fprintf() 표준 출력(stdout)

```
fscanf(f, "%d %s %d %dWn", &cnt, name, &point1, &point2);
//fscanf_s(f, "%d %s %d %dWn", &cnt, name, 30, &point1, &point2); //30은 name의 크기
//표준출력에 쓰기
fprintf(stdout, "Wn%6s%16s%10s%8sWn", "번호", "이름", "중간", "기말");
fprintf(stdout, "%5d%18s%8d%8dWn", cnt, name, point1, point2);
```

## 실습예제 15-2

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

int main()
{
    char fname[] = "grade.txt";
    FILE *f;
    char name[30];
    int point1, point2, cnt = 0;

    if (fopen_s(&f, fname, "w") != 0)
        //if ( (f = fopen(fname, "w")) == NULL )
    {
        printf("파일이 열리지 않습니다.Wn");
        exit(1);
    };
    printf("이름과 성적(중간, 기말)을 입력하세요.Wn");
    scanf("%s %d %d", name, &point1, &point2);
    //scanf_s("%s%d%d", name, 30, &point1, &point2);
    //파일 "grade.txt"에 쓰기
    fprintf(f, "%d %s %d %dWn", ++cnt, name, point1, point2);
    fclose(f);

    if (fopen_s(&f, fname, "r") != 0)
        //if ( (f = fopen(fname, "r")) == NULL )
    {
        printf("파일이 열리지 않습니다.Wn");
        exit(1);
    };
    //파일 "grade.txt"에서 읽기
    fscanf(f, "%d %s %d %dWn", &cnt, name, &point1, &point2);
    //fscanf_s(f, "%d %s %d %dWn", &cnt, name, 30, &point1, &point2);
    //표준출력에 쓰기
    fprintf(stdout, "Wn%6s%16s%10s%8sWn", "번호", "이름", "중간", "기말");
    fprintf(stdout, "%5d%18s%8d%8dWn", cnt, name, point1, point2);
    fclose(f);

    return 0;
}
```

## 파일 문자열 입출력

### 함수 fgets()와 fputs()

- 행 단위 파일 입출력
- 기존 gets()와 puts()를 파일에 사용하는 개념
- \*fgets()는 파일로부터 문자열을 개행문자(wn)까지 읽어 마지막 개행문자를 'w0'문자로 바꾸어 입력 버퍼 문자열에 저장
- 함수 원형은 다음과 같음

```
char * fgets(char * Buf, int _MaxCount, FILE * _File);
```

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

Buf : 문자열이 저장될 문자 포인터

\_MaxCount : 입력할 문자의 최대 수

File : 입력 문자열이 저장될 파일

### 함수 feof()와 ferror()

- feof() : 파일 스트림의 EOF(End Of File) 표시 검사  
읽기 작업이 이전 부분을 읽으면 0 반환  
그렇지 않으면 0 이 아닌 값 반환  
(파일 스트림의 EOF 은 이전 읽기 작업에서 EOF 표시에 도달하면 0 이 아닌 값으로 지정된다. 단순히 파일 지시자가 파일의 끝에 있더라도 feof()의 결과는 0)
- ferror() : 파일 처리에서 오류 발생을 검사  
오류가 발생하지 않으면 0 반환  
오류 발생 시 0 이 아닌 값 반환
- 함수원형은 다음과 같음

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

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

## 실습예제 15-3

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

int main()
{
    char fname[] = "grade.txt";
    FILE *f;
    char names[80];
    int cnt = 0;

    if (fopen_s(&f, fname, "w") != 0)
        //if ( (f = fopen(fname, "w")) == NULL )
    {
        printf("파일이 열리지 않습니다.\n");
        exit(1);
    };
    printf("이름과 성적(중간, 기말)을 입력하세요.\n");
    fgets(names, 80, stdin);

    //콘솔에 이름 중간 기말 입력하고 Enter 키
    //여러 줄에 입력하다가
    //종료하고 싶을 때 새줄 첫 행에서 ctrl + Z 누름
    while ( !feof(stdin) )
    {
        //파일 "grade.txt"에 쓰기
        fprintf(f, "%d ", ++cnt);    //맨 앞에 번호를 삽입
        fputs(names, f);             //이후에 입력 받은 이름과 성적 2개 저장
        fgets(names, 80, stdin);     //다시 표준입력
    }
    fclose(f);

    return 0;
}
```

## 파일 문자 입출력

### 함수 fgetc()과 fputc()

- 문자 하나 단위 입출력
- getchar()와 putchar()를 파일에 사용하는 개념
- 헤더파일 stdio.h
- 함수 원형은 다음과 같음 (fgetc()와 getc()는 같은 기능 || fputc()와 putc()는 같은 기능)

|  |   |
|--|---|
| <code>int fgetc(FILE * _File);</code>          | <code>int getc(FILE * _File);</code>          |
| <code>int fputc(int _Ch, FILE * _File);</code> | <code>int putc(int _Ch, FILE * _File);</code> |

### 예제 15-4

예제 15-4 를 통해서 파일을 쓰기모드로 열어(`fopen()`) 표준입력(`_getche()`)으로 받은 문자를 파일에 출력(`fputc()`)하고 파일을 읽기모드로 열어(`fopen()`) 파일 입력받은(`fgetc()`) 문자를 콘솔에 표준출력(`_putch()`)하는 흐름을 알 수 있음

- `_getche`, `_putch`)는 Visual C++에서 `getch()`와 `putch()`를 대체하는 권장 함수

### 실습예제 15-4

```
// file: fgetc.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int main()
{
    char fname[] = "char.txt"; //입력한 내용이 저장될 파일이름
    FILE *f; //파일 포인터

    //쓰기모드로 파일 열기
    if (fopen_s(&f, fname, "w") != 0)
        //if ( (f = fopen(fname, "w")) == NULL )
    {
        printf("파일이 열리지 않습니다.\n");
        exit(1);
    };
    puts("문자를 입력하다가 종료하려면 x를 입력 >>");

    /*표준입력으로 받은 문자를 파일에 출력하는 부분*/
    int ch; //입력된 문자 저장
    while ((ch = _getche()) != 'x')
        //파일 "char.txt"에 쓰기
        fputc(ch, f); //파일에 문자 출력
```

```

fclose(f); puts("");

//읽기모드로 파일 열기
if (fopen_s(&f, fname, "r") != 0)
{
    printf("파일이 열리지 않습니다.\n");
    exit(1);
};

/*파일에서 다시 문자를 입력받아 콘솔에 표준출력하는 부분*/
while ((ch = fgetc(f)) != EOF)
    //파일 "char.txt"에서 다시문자 읽기
    _putch(ch);    //파일로부터 입력 받은 문자를 표준출력
fclose(f); puts("");

return 0;
}

```

## 파일 내용을 표준출력으로 그대로 출력

- 예제 15-5 는 **도스 명령어 type** 와 같이 내용을 그대로 콘솔에 출력하는 프로그램.  
도스 프롬프트에서 실행하면 다음과 같다.

```

D:\Visual Studio source\anyway\Debug>anyway ../anyway/list.c
1: // file: list.c
2: #include <stdio.h>
3: #include <stdlib.h>
4:
5: int main(int argc, char* argv[])
6: {
7:     FILE* f;
8:
9:     if (argc != 2)
10:    {
11:        printf("사용법: list filename\n");
12:        exit(1);
13:    }
14:
15:    //읽기 모드로 파일 열기
16:    if (fopen_s(&f, argv[1], "r") != 0)
17:        //if ( (f = fopen(argv[1], "r")) == NULL )
18:    {
19:        printf("파일이 열리지 않습니다.\n");
20:        exit(1);
21:    }
22:
23:    //문자를 저장할 ch, 행번호를 저장할 cnt
24:    int ch, cnt = 0;
25:    printf("%4d: ", ++cnt);
26:    while ((ch = fgetc(f)) != EOF)
27:    {
28:        putchar(ch); //putc(ch, stdout);
29:        //행 처음에 행 번호 출력
30:        if (ch == '\n') printf("%4d: ", ++cnt);
31:    }
32:    printf("\n");
33:    fclose(f);
34:
35:    return 0;
36: }

```

## 실습예제 15-5

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

int main(int argc, char *argv[])
{
    FILE *f;

    if (argc != 2)
    {
        printf("사용법: list filename\n");
        exit(1);
    }

    //읽기 모드로 파일 열기
    if (fopen_s(&f, argv[1], "r") != 0)
    //if ( (f = fopen(argv[1], "r")) == NULL )
    {
        printf("파일이 열리지 않습니다.\n");
        exit(1);
    }

    //문자를 저장할 ch, 행번호를 저장할 cnt
    int ch, cnt = 0;
    printf("%4d: ", ++cnt);
    while ((ch = fgetc(f)) != EOF)
    {
        putchar(ch); //putc(ch, stdout);
        //행 처음에 행 번호 출력
        if (ch == '\n') printf("%4d: ", ++cnt);
    }
    printf("\n");
    fclose(f);

    return 0;
}
```

## LAB 파일의 대소문자를 서로 바꿔 다른 파일로 생성

### Lab 15-2

```
// 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;
}
```