

# PORTFOLIO

20161865 박종근

## CONTENTS



### 011 문자와 문자열

- 문자와 문자열
- 문자열 관련 함수
- 여러 문자열 처리



### 012 변수 유효범위

- 전역변수와 지역변수
- 정적 변수와 레지스터 변수
- 메모리 영역과 변수 이용



### 013 구조체와 공용체

- 구조체와 공용체
- 자료형 재정의
- 구조체와 공용체의 포인터와 배열



# Part 11.

---

문자와 문자열



## 11.1 문자와 문자열

### 문자와 문자열의 개념

#### 문자

- 영어의 알파벳이나 한글의 한 글자를 작은 따옴표로 둘러싸서 표현 ( ‘ A ‘ )
- C언어에서 저장공간 크기 1byte인 자료형 char로 지원
- 작은 따옴표에 의해 표기된 문자를 문자 상수라고 칭함.

#### 문자열

- 문자의 모임인 일련의 문자
- 일련의 문자 앞 뒤로 큰 따옴표로 둘러싸서 표현 ( “ java” )
- 문자 하나를 큰 따옴표로 둘러싸도 문자열 상수지만, 문자의 나열을 작은 따옴표로 둘러싸면 문자열이 아님
- EX ( “A” -> O , ‘ABC’ -> X )

## 11.1 문자와 문자열

### 문자와 문자열의 선언

문자열을 저장하려면 문자의 모임인 ‘문자 배열’을 사용한다.

문자열의 마지막은 의미하는 NULL 문자 ‘\0’가 마지막에 저장되어야 한다.

그러므로 문자열이 저장되는 배열크기는 반드시 저장될 문자 수보다 1이 커야 한다.

배열 선언 시 초기화 방법을 사용하면 문자열을 쉽게 저장 할 수 있다.

배열 초기화 배열크기는 지정하지 않는게 좋고, 지정하면 실제 문자 수 +1로 배열크기를 지정한다.

IF 지정한 배열크기가 문자 수+1보다 크면 나머지 부분은 모두 ‘\0’으로 채워진다.

배열크기가 작으면 문자열 출력 등에서 문제가 발생한다.

```
char c[] = "C language";    //크기를 생략하는 것이 간편하다.
```

```
char c[11] = "C language"   //크기 지정 시 (문자수+1)을 잊지 말자.
```

```
char go[5] = "go";          //크기가 (문자+1)보다 크면 나머지는 모두 '\0'으로 채워진다.
```

## 11.1 문자와 문자열

### 문자와 문자열 출력

함수 printf()에서 형식제어문자 %c로 문자를 출력한다.

배열이름 또는 문자 포인터를 사용하여 형식제어문자 %s로 문자열을 출력한다.

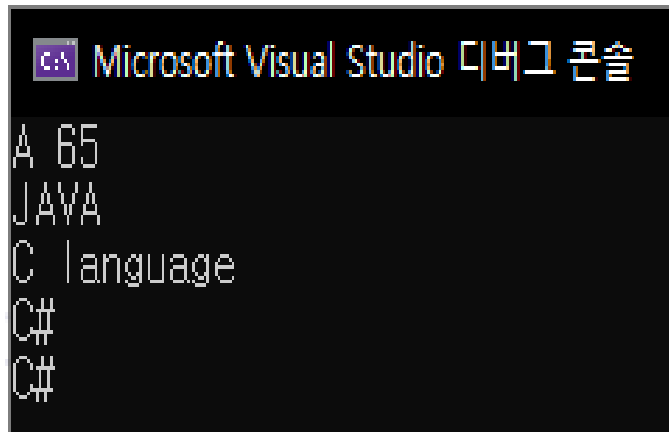
```
// file: chararray.c
#include <stdio.h>

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;
}
```



Microsoft Visual Studio 디버그 콘솔

```
A 65
JAVA
C language
C#
C#
```

## 11.1 문자와 문자열

### 문자열을 구성하는 문자 참조

문자열 상수를 문자 포인터에 저장하는 방식

문자 포인터 변수에 문자열 상수를 저장할 수 있다.

문자열 출력도 함수 printf()에서 포인터 변수와 형식제어문자 %s로 간단히 처리할 수 있다.

문자 포인터에 의한 선언으로는 문자 하나 하나의 수정은 불가능하다.

```
int i = 0;  
char* java = "java";  
printf("%s ", java);
```

```
while (java[i] != '\0')  
    printf("%c", java[i++]);  
printf("\n");
```

문자열을 구성하는 하나 하나의 문자를 배열형식으로 직접 참조하여 출력하는 방식도 사용 가능하다.

하지만 변수를 사용하여 문자를 수정하거나 수정될 수 있는 함수의 인자로 사용하면 실행 오류가 발생한다.

## 11.1 문자와 문자열

### '\0' 문자에 의한 문자열 분리

함수 printf()에서 %s는 문자 포인터가 가리키는 위치에서 NULL 문자 까지를 하나의 문자열로 인식한다.

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

```
int main(void)
{
    char c[] = "C C++ Java";
    printf("%s\n", c);
    c[5] = '\0'; // NULL 문자에 의해 문자열 분리
    printf("%s\n%s\n", 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 != '\0')도 가능
        printf("%c", *p++);
    printf("\n");

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
C C++ Java
C C++
Java
C C++ Java
```



## 11.1 문자와 문자열

### 다양한 문자 입출력

버퍼처리 함수 `getchar()` & `putchar()`

`getchar()`는 문자의 입력에 사용, `putchar()`는 문자의 출력에 사용

`getchar()`는 라인 버퍼링 방식을 사용해서 엔터키를 눌러야 이전에 입력한 문자의 입력이 실행된다.

입력한 문자는 임시 저장소인 버퍼에 저장되었다가 엔터키를 만나면 함수는 버퍼에서 문자를 읽기 시작한다.

함수 `getche()`

버퍼를 사용하지 않고 문자를 입력하는 함수

함수를 이용하려면 헤더파일 `conio.h`를 삽입해야 한다.

함수 `getch()`

문자 입력을 위한 함수 `getch()`는 입력한 문자가 화면에 보이지 않는 특성이 있다.

버퍼를 사용하지 않고 문자를 입력하는 함수

함수를 이용하려면 헤더파일 `conio.h`를 삽입해야 한다.

두가지 함수는 여러 컴파일러에서 이름이 수정되어 서비스 되고 있다.

`getch()` -> `_getch()`

`getche()` -> `_getche()`

## 11.1 문자와 문자열

### 문자열 입력

`scanf()`

공백으로 구분되는 하나의 문자열을 입력 받을 수 있다.

문자열 입력은 충분한 공간의 문자배열이 있어야 가능하다.

`gets()`와 `puts()`

함수 `gets()`는 한 행의 문자열 입력에 유용한 함수이고, `puts()`는 한 행에 문자열을 출력하는 함수이다.

`char * gets (char * buffer);`

문자열을 입력 받아 `buffer`에 저장하고 입력 받은 첫 문자의 주소 값을 반환한다.

표준입력으로 엔터키를 누를 때까지 공백을 포함한 한 행의 모든 문자열을 입력 받는다.

입력된 문자열에서 마지막 엔터키를 `'\0'` 문자로 대체하여 저장한다.

`char * gets_s (char * buffer, size_t sizebuffer);`

두번째 인자인 `sizebuffer`는 정수형으로 `buffer`의 크기를 입력한다.

`gets()`보단 `gets_s()`의 사용을 권장한다.

`int puts(const char * str);`

인자인 문자열 `str`에서 마지막 `'\0'` 문자를 개행 문자인 `'\n'`로 대체하여 출력한다.

함수 `puts()`는 일반적으로 정수 값 0을 반환하는데, 오류가 발생하면 EOF를 반환한다.

EOF : 파일의 끝이라는 의미로, `stdio.h` 헤더파일에 정수 -1로 정의 되어 있다.

## 11.2 문자열 관련 함수

### 문자열 입력

문자열 배열에 관한 다양한 함수가 있다.

`void *memchr(const void *str, int c, size_t n)`

메모리 `str`에서 `n` 바이트까지 문자 `c`를 찾아 그 위치를 반환한다

`int memcmp(const void *str1, const void *str2, size_t n)`

메모리 `st1`과 `str2`를 첫 `n` 바이트를 비교 검색하여 같으면 0, 다르면 음수 또는 양수를 반환한다.

`void *memcpy(void *dest, const void *src, size_t n)`

포인터 `src` 위치에서 `dest`에 `n` 바이트를 복사한 후 `dest` 위치를 반환한다

`void *memmove(void *dest, const void *src, size_t n)`

포인터 `src` 위치에서 `dest`에 `n` 바이트를 복사한 후 `dest` 위치를 반환한다.

`void *memset(void *str, int c, size_t n)`

포인터 `str` 위치에서부터 `n` 바이트까지 문자 `c`를 지정한 후 `str` 위치를 반환한다.

`size_t strlen(const char *str)`

포인터 `str` 위치에서부터 널 문자를 제외한 문자열의 길이를 반환한다.

## 11.2 문자열 관련 함수

### 함수 strcmp()

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

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

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

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

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

```
// 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;
}
```

#### C:\ Microsoft Visual Studio 디버그 콘솔

```
strcmp(java, java) = 0
strcmp(java, jav) = 1
strcmp(jav, java) = -1
strncmp(jav, java, 3) = 0
```

## 11.2 문자열 관련 함수

### 함수 strcpy()

함수 strcpy()와 strncpy()는 문자열을 복사하는 함수이다.

`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 길면 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);`

두번째 인자인 sizedest는 정수형으로 dest의 크기를 입력한다.

반환형 errno\_t는 정수형이며 반환 값은 오류번호로 성공하면 0을 반환한다.

```
// 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", strcpy_s(dest, 80, "C#", 3));
    //printf("%s\n", dest);
    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
C is a language.
C# is a language.
C#
```

## 11.2 문자열 관련 함수

### 함수 strcat()

하나의 문자열 뒤에 다른 하나의 문자열을 연이어 추가해 연결하려면 함수 `strcat()`를 사용한다.

함수 `strcat()`는 앞 문자열에 뒤 문자열의 null 문자까지 연결하여, 앞의 문자열 주소를 반환하는 함수이다.

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

앞 문자열 `dest`에 뒤 문자열 `source`를 연결해 저장하며, 이 연결된 문자열을 반환하고 뒤 문자열은 수정될 수 없다.

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

앞 문자열 `dest`에 뒤 문자열 `source` 중에서 `n`개의 크기만큼을 연결해 저장하며, 이 연결된 문자열을 반환하고 뒤 문자열은 수정될 수 없다.

지정한 `maxn`이 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);
```

두번째 인자인 `sizedest`는 정수형으로 `dest`의 크기를 입력한다.

반환형 `errno_t`는 정수형이며 반환값은 오류번호로 성공하면 0을 반환한다.

```
// 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", strcat(dest, " a java", 2));
    //printf("%d\n", strcat_s(dest, 80, " a proce", 2));
    //printf("%s\n", dest);
    printf("%s\n", strcat(dest, "procedural"));
    printf("%s\n", strcat(dest, "language."));
    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
C is
C is a java
C is a javaprocedural
C is a javaprocedural language.
```

## 11.2 문자열 관련 함수

### 함수 strtok()

함수 strtok()은 문자열에서 구분자인 문자를 여러 개 지정하여 토큰을 추출하는 함수이다.

함수 strtok()에서 첫번째 인자는 토큰을 추출할 대상인 문자열, 두번째 인자는 구분자로 문자의 모임인 문자열이다.

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

앞 문자열 str에서 뒤 문자열 delim을 구성하는 구분자를 기준으로 순서대로 토큰을 추출하여 반환하는 함수이며, 뒤 문자열 delim은 수정될 수 없다.

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

마지막 인자인 contex는 함수 호출에 사용되는 위치 정보를 위한 인자이다.

문장 ptoken = strtok(str, delimiter);으로 첫 토큰 추출

결과를 저장한 ptoken이 NULL이면 더 이상

분리할 토큰이 없는 경우이다.

```
// 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 = ",!";
    // char *next_token;

    printf("문자열 \"%s\"을 >>\n", str1);
    printf("구분자[%s]를 이용하여 토큰을 추출 >>\n", delimiter);
    char* ptoken = strtok(str1, delimiter);
    //char *ptoken = strtok_s(str, delimiter, &next_token);
    while (ptoken != NULL)
    {
        printf("%s\n", ptoken);
        ptoken = strtok(NULL, delimiter); //다음 토큰을 반환
        //ptoken = strtok_s(NULL, delimiter, &next_token); //다음 토큰을 반환
    }
    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

문자열 "C and C++ language are best!"을 >>  
 구분자[ ,!]를 이용하여 토큰을 추출 >>  
 C  
 and  
 C++  
 language  
 are  
 best

## 11.2 문자열 관련 함수

### 문자열의 길이와 위치 검색

함수 `strlen()`은 NULL 문자를 제외한 문자열 길이를 반환하는 함수이다.

함수 `strlwr()`는 인자를 모두 소문자로 변환하여 반환한다.

함수 `strupr()`는 인자를 모두 대소문자로 변환하여 반환한다.

```
char * strlwr(char * str);
```

```
errno_t _strlwr_s(char * str, size_t strsize);
```

문자열 `str`을 모두 소문자로 변환하고 변환한 문자열을 반환한다.

`str`은 상수이면 오류가 발생하고 `errno_t`는 정수형의 오류 번호이며, `size_t`도 정수형으로 `strsize`는 `str`의 길이이다.

```
char * strupr(char * str);
```

```
errno_t _strupr_s(char * str, size_t strsize);
```

문자열 `str`을 모두 소문자로 변환하고 변환한 문자열을 반환한다.

```
char * strpbrk(const char * str, const char * charset);
```

앞의 문자열 `str`에서 뒤 문자열 `charset`에 포함된 문자가 나타나는 처음 위치를 찾아 그 주소 값을 반환, 못 찾으면 NULL포인터를 반환한다.

```
char * strstr(const char * str, const char * strsearch);
```

앞의 문자열 `str`에서 뒤 문자열 `strsearch`이 나타나는 처음 위치를 찾아 그 주소 값을 반환하며, 만일 찾지 못하면 NULL 포인터를 반환한다

```
char * strchr(const char * str, char ch);
```

앞의 문자열 `str`에서 뒤 문자 `ch`가 나타나는 처음 위치를 찾아 그 주소 값을 반환하며, 만일 찾지 못하면 NULL 포인터를 반환한다.



## 11.3 여러 문자열 처리

### 문자 포인터 배열

여러 개의 문자열을 처리하는 하나의 방법은 문자 포인터 배열을 이용하는 방법이다.

하나의 문자 포인터가 하나의 문자열을 참조 가능 -> 여러 개의 문자열을 참조할 수 있음.

이 방법은 각각의 문자열 저장을 위한 최적의 공간을 사용하는 장점을 갖는다.

그러나 이러한 문자 포인터를 사용해서는 문자열 상수의 수정은 불가능하다.

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

### 이차원 문자 배열

여러 개의 문자열을 처리하는 다른 방법은 문자의 이차원 배열을 이용하는 방법이다.

**Char ca[행][열]**

첫번째 행 크기는 문자열 개수를 지정하거나 빈 공백으로 지정하고

두번째 열 크기는 문자열 중에서 가장 긴 문자열의 길이보다 1크게 지정한다.

그러므로 가장 긴 문자열"JAVA"보다 1이 큰 5를 2차원 배열의 열 크기로 지정한다.

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

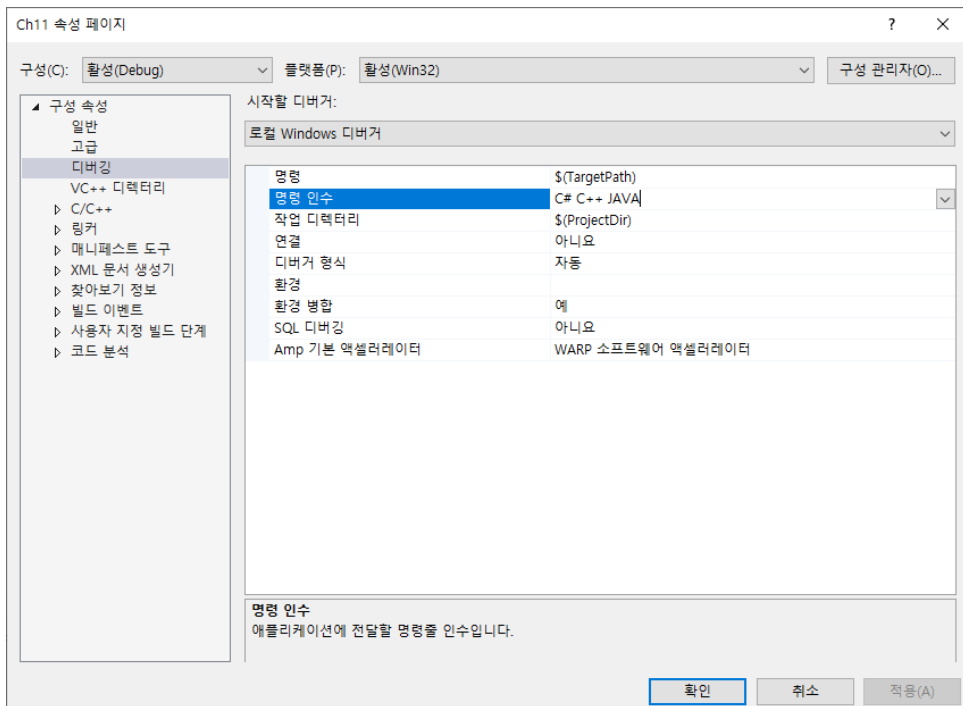
## 11.3 여러 문자열 처리

### 명령행 인자

Main(int argc, char \* argv[])

명령행에서 입력하는 문자열을 프로그램으로 전달하는 방법이 명령해 인자를 사용하는 방법이다.

실행 프로그램 이름도 하나의 명령행 인자에 포함된다.



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

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

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
실행 명령행 인자(command line argument) >>
argc = 4
argv[0] = C:\Users\82105\source\repos\Ch11\Debug\Ch11.exe
argv[1] = C#
argv[2] = C++
argv[3] = JAVA
```

# 11.1 ~ 11.3 프로그래밍 연습

## 문제 1

한 행을 표준입력으로 입력 받은 문자열의 길이를 구하는 함수 `mystrlen()`을 구현하여 라이브러리 `strlen()`과 결과를 비교하는 프로그램을 작성 하시오.

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

int main() {
    int mystrlen(const char* p);
    char word[30];
    gets_s(word, sizeof(word));

    printf("입력된 값의 문자열 길이를 mystrlen으로 찾습니다. : %d\n", mystrlen(word));
    printf("입력된 값의 문자열 길이를 strlen으로 찾습니다. : %d\n", strlen(word));
}

int mystrlen(const char* p) {
    int count = 0;
    while (p[count] != NULL)
        count++;

    return count;
}
```

Microsoft Visual Studio 디버그 콘솔

```
20161865 박종근
입력된 값의 문자열 길이를 mystrlen으로 찾습니다. : 15
입력된 값의 문자열 길이를 strlen으로 찾습니다. : 15
```

# 11.1 ~ 11.3 프로그래밍 연습

## 문제 9

여러 줄의 문자열을 표준입력으로 입력 받아 구두점의 수를 구하여 출력하는 프로그램을 작성 하시오.

- 구두점인지는 함수 `ispunct()`를 사용하여 판단

```
#include <stdio.h>
#include <string.h>
#define LINENUM 81

int main(void)
{
    char line[10][LINENUM];
    int count = 0;
    int A = 0, B = 0, maxline = 0;
    printf("문장 여러줄을 점 포함해서 입력\n");
    printf("새로운 줄 처음에 ctrl+Z, 그리고 Enter를 입력하세요\n");
    while (gets(line[A]))
    {
        line[A++];
        maxline++;
    }
    printf("\n");
    for (A = 0; A < maxline; A++) {
        printf("%d줄에 입력한 문자열에서 구두점 출력\n", A + 1);
        B = 0;
        while (line[A][B]) {
            if (ispunct(line[A][B])) {
                printf("구두점 %d : %c\n", ++count, line[A][B]);
            }
            B++;
        }
    }
    printf("\n구두점의 수는 모드 %d개입니다.\n", count);
}
```

### Microsoft Visual Studio 디버그 콘솔

문장 여러줄을 점 포함해서 입력  
새로운 줄 처음에 ctrl+Z, 그리고 Enter를 입력하세요  
HI .

....  
BYE.  
^Z

1줄에 입력한 문자열에서 구두점 출력

구두점 1 : .

2줄에 입력한 문자열에서 구두점 출력

구두점 2 : .

구두점 3 : .

구두점 4 : .

구두점 5 : .

구두점 6 : .

3줄에 입력한 문자열에서 구두점 출력

구두점 7 : .

구두점의 수는 모드 7개입니다.

# Part 12.

---

변수 유효범위



## 12.1 전역변수와 지역변수

### 변수 범위와 지역변수

변수의 참조가 유효한 범위를 변수의 유효 범위라 한다.

변수의 유효 범위는 지역 유효 범위와 전역 유효 범위로 나뉜다.

지역 유효 범위는 함수 또는 블록 내부에서 선언되어 그 지역에서 변수의 참조가 가능한 범위이다.

전역 유효 범위는 2가지로 나뉜다

- 하나의 파일에서만 변수의 참조가 가능한 범위
- 프로젝트를 구성하는 모든 파일에서 변수의 참조가 가능한 범위

## 12.1 전역변수와 지역변수

### 지역변수

지역변수는 함수 또는 블록에서 선언된 변수이다.

함수나 블록에서 지역변수는 선언 문장 이후에 함수나 블록의 내부에서만 사용이 가능하다.

함수의 매개변수도 함수 전체에서 사용 가능한 지역변수와 같다.

지역변수는 선언 후 초기화하지 않으면 쓰레기 값이 저장되므로 주의해야 한다.

지역변수가 할당되는 메모리 영역을 스택(stack)이라 한다.

지역변수는 선언된 부분에서 자동으로 생성되고 함수나 블록이 종료되는 순간 메모리에서 자동으로 제거된다.

그러한 이유로 지역변수는 자동변수라 한다.

```
// 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("%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("%d %d\n", param, local); //param과 local 참조 가능
    //printf("%d\n",n); //n 참조 불가능
}
```

Microsoft Visual Studio 디버그 콘솔

```
10
    0 0
    1 1
    2 3
10
    20 100
```

## 12.1 전역변수와 지역변수

### 전역변수

전역변수는 함수 외부에서 선언되는 변수이다.

전역변수는 외부변수라고도 부르며, 일반적으로 프로젝트의 모든 함수나 블록에서 참조할 수 있다.

전역변수는 선언되면 자동으로 초기값이 자료형에 맞는 0으로 지정된다.

함수나 블록에서 전역변수와 같은 이름으로 지역변수를 선언할 수 있다.

전역변수는 프로젝트의 다른 파일에서도 참조가 가능하다.

Extern을 사용한 참조선언 구문을 변수선언 문장 맨 앞에 extern을 넣는 구조이다.

전역변수는 어디에서든지 수정이 가능하다.

그러나 전역변수에 예상하지 못한 값이 저장되면 프로그램 어느 부분에서 수정되었는지 알기 어려운 단점이 있다.

```
// 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
}
```

```
// 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: %.2f\n", getArea(r));
    printf("둘레1: %.2f\n", 2* PI*r);
    printf("둘레2: %.2f\n", getCircum(r));
    printf("PI: %f\n", PI); //지역변수 PI 참조
    printf("gi: %d\n", gi); //전역변수 gi 기본값
    return 0;
}

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



## 12.1 전역변수와 지역변수

### 전역변수

```
// 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: %.2f\n", getArea(r));
    printf("둘레1: %.2f\n", 2* PI*r);
    printf("둘레2: %.2f\n", getCircum(r));
    printf("PI: %f\n", PI); //지역변수 PI 참조
    printf("gi: %d\n", gi); //전역변수 gi 기본값
    return 0;
}
double getArea(double r)
{
    return r*r * PI; //전역변수 PI 참조
}
```

```
// circumference.c
//이미 외부에서 선언된 전역변수임을 알리는 선언
extern double PI;
double getCircum(double r)
{
    //return double PI; //함수 내부에서만 참조 가능
    return 2 * r * PI; //전역변수 PI 참조
}
```

C:\ Microsoft Visual Studio 디버그 콘솔

```
면적: 108.19
둘레1: 36.88
둘레2: 36.86
PI: 3.141592
gi: 0
```

## 12.2 정적변수와 레지스터 변수

### 기억부류와 레지스터 변수

변수 선언의 위치에 따라 변수는 전역과 지역으로 나뉜다.

변수는 4가지의 기억부류인 `auto`, `register`, `static`, `extern`에 따라 할당되는 메모리 영역이 결정되고 메모리의 할당과 제거 시기가 결정된다.

모든 기억 부류는 전역 변수 또는 지역 변수이다.

기억부류 `Auto`와 `register`는 지역변수에만 이용이 가능, `static`은 지역과 전역 모든 변수에 이용이 가능하고 `extern`은 전역변수에만 사용이 가능하다.

키워드 `extern`을 제외하고 나머지 3개의 기억부류의 변수선언에서 초기값을 저장할 수 있다.

키워드 `auto`는 지역변수 선언에 사용되며 생략이 가능하다.

## 12.2 정적변수와 레지스터 변수

### 키워드 register

레지스터 변수는 변수의 저장공간이 일반 메모리가 아니라 CPU 내부의 레지스터에 할당되는 변수이다.

레지스터 변수는 키워드 register를 자료형 앞에 넣어 선언한다.

레지스터 변수는 지역변수에만 이용이 가능하다.

레지스터는 CPU내부에 있는 기억장소이므로 일반 메모리보다 빠르게 참조될 수 있다.

일반 메모리에 할당되는 변수가 아니므로 주소연산자 &를 사용할 수 없다.

주로 처리 속도를 증가시키려는 변수에 이용한다.

특히 반복문의 횟수를 제어하는 제어변수에 이용하면 효과적이다.

```
// 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: %.2f\n", getArea(r));
    printf("둘레1: %.2f\n", 2* PI*r);
    printf("둘레2: %.2f\n", getCircum(r));
    printf("PI: %f\n", PI); //지역변수 PI 참조
    printf("gi: %d\n", gi); //전역변수 gi 기본값
    return 0;
}
double getArea(double r)
{
    return r*r * PI; //전역변수 PI 참조
}
```

Microsoft Visual Studio 디버그 콘솔

양의 정수 입력 >>9  
합: 45

## 12.2 정적변수와 레지스터 변수

### 정적 변수

변수 선언에서 자료형 앞에 키워드 `static`을 넣어 정적변수를 선언할 수 있다.

정적변수는 초기값을 지정하지 않으면 자동으로 자료형에 따라 0이나 '\0' 또는 NULL 값이 저장된다.

정적변수의 초기화는 단 한번만 수행되며, 상수로만 가능하다

### 정적 지역변수

함수나 블록에서 정적으로 선언되는 변수가 정적 지역변수이다.

정적 지역변수는 함수나 블록을 종료해도 메모리에서 제거되지 않고 계속 메모리에 유지 관리되는 특성이 있다.

```
// 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
}

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

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

Microsoft Visual Studio 디버그 콘솔

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

## 12.2 정적변수와 레지스터 변수

### 정적 변수

#### 정적 전역변수

함수 외부에서 정적으로 선언되는 변수가 정적 전역변수이다.

일반 전역변수는 파일 소스가 다르더라도 `extern`을 사용하여 참조가 가능하나, 정적 전역변수는 선언된 파일 내부에서만 참조가 가능하다.

모든 함수에서 공유할 수 있는 저장공간을 이용할 수 있는 장점이 있다.

하지만 어느 한 함수에서 잘못 다루면 모든 함수에 영향을 미칠 수도 있다.

프로그램이 크고 복잡하면 전역변수의 사용은 원하지 않는 전역변수의 수정과 같은 부작용의 위험성이 존재한다.

```
// 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();
}

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

```
//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;
}
```

Microsoft Visual Studio 디버그 콘솔

함수 increment()가 총 5번 호출되었습니다,  
전역 변수: 10

## 12.3 메모리 영역과 변수 이용

### 메모리 영역

메인 메모리의 영역은 프로그램 실행 과정에서 데이터, 힙, 스택 영역 세 부분으로 나뉜다.

메모리 영역은 변수의 유효범위와 생존기간에 결정적 역할을 하며, 변수는 기억부류에 따라 할당되는 메모리 공간이 달라진다.

### 데이터 영역

데이터 영역은 전역변수와 정적변수가 할당되는 저장공간이다.

데이터 영역은 메모리 주소가 낮은 값에서 높은 값으로 저장 장소가 할당된다.

프로그램이 시작되는 시점에 정해진 크기대로 고정된 메모리 영역이 확보된다.

### 힙 영역

힙 영역은 동적 할당되는 변수가 할당되는 저장공간이다.

데이터 영역과 스택 영역 사이에 위치한다.

프로그램이 실행하면서 영역 크기가 계속적으로 변한다.

### 스택 영역

스택 영역은 함수 호출에 의한 형식 매개변수 그리고 함수 내부의 지역변수가 할당되는 저장공간이다.

메모리 주소가 높은 값에서 낮은 값으로 저장 장소가 할당된다.

프로그램이 실행하면서 영역 크기가 계속적으로 변한다.

# 12.3 메모리 영역과 변수 이용

## 변수의 이용

변수의 종류

선언위치	상세 종류	키워드		유효범위	기억장소	생존기간
전역	전역 변수	참조 선언	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)				

변수의 유효 범위

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

# 12.3 메모리 영역과 변수 이용

## 변수의 이용

### 변수의 초기값

지역, 전역	종류	자동 저장되는 기본 초기값	초기값 저장
전역	전역변수	자료형에 따라 0이나 '0' 또는 NULL 값이 저장됨.	프로그램 시작 시
	정적 전역변수		
지역	정적 지역변수	쓰레기 값이 저장됨.	함수나 블록이 실행될 때 마다
	레지스터 변수		
	자동 지역변수		



## 12.1 ~ 12.3 프로그래밍 연습

## 문제6 ~ 문제7

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int number;
int user;
int trycount;
static int min = 1;
static int max = 100;
void setNumber(void);
void printHead(void);
void printHigher(void);
void printLower(void);
void printAnswer(void);
int main() {
    setNumber();
    printHead();
    while (1) {
        trycount++;
        if (user > number) {
            printHigher();
        }
        else if (user < number) {
            printLower();
        }
        else {
            printAnswer();
            break;
        }
        if (trycount == 4) {
            printf("5번의 기회를 모두 사용했습니다. 난수 값 = %d 입니다.\n", number);
            break;
        }
    }
}
```

```
void printAnswer() {
    printf("축하합니다! 정답은 %d 입니다.\n", user);
}

void printHigher() {
    max = user - 1;
    printf("%d. 맞추어야 할 정수가 입력한 정수 %d 보다 작습니다.\n", trycount, user);
    printf("%d 에서 %d 사이의 정수를 다시 입력하세요. > ", min, max);
    scanf_s("%d", &user, sizeof(int));
    puts("");
}

void printLower(void)
{
}

void printLower() {
    min = user + 1;
    printf("%d. 맞추어야 할 정수가 입력한 정수 %d 보다 큼니다.\n", trycount, user);
    printf("%d 에서 %d 사이의 정수를 다시 입력하세요. > ", min, max);
    scanf_s("%d", &user, sizeof(int));
    puts("");
}

void setNumber() {
    long seconds = (long)time(NULL);
    srand(seconds);
    number = rand() % 100 + 1;
}

void printHead(void)
{
}

void printHead() {
    printf("%d 에서 %d 까지의 하나의 정수가 결정되었습니다.\n이 정수를 맞추어 보세요. >", min, max);
    scanf_s("%d", &user, sizeof(int));
    puts("");
}
```

Microsoft Visual Studio 디버그 콘솔

1. 에서 100 까지의 하나의 정수가 결정되었습니다.  
이 정수를 맞추어 보세요. >50

1. 맞추어야 할 정수가 입력한 정수 50 보다 큼니다.  
51 에서 100 사이의 정수를 다시 입력하세요. > 75

2. 맞추어야 할 정수가 입력한 정수 75 보다 작습니다.  
51 에서 74 사이의 정수를 다시 입력하세요. > 60

3. 맞추어야 할 정수가 입력한 정수 60 보다 큼니다.  
61 에서 74 사이의 정수를 다시 입력하세요. > 65

4. 맞추어야 할 정수가 입력한 정수 65 보다 큼니다.  
66 에서 74 사이의 정수를 다시 입력하세요. > 68

5번의 기회를 모두 사용했습니다. 난수 값 = 66 입니다.

# Part 13.

---

## 구조체와 공용체



## 13.1 구조체와 공용체

### 구조체 개념과 정의

#### 구조체 개념

구조체란 정수나 문자, 실수나 포인터 그리고 이들의 배열 등을 묶어 하나의 자료형으로 이용하는 것이 구조체이다.

연관성이 있는 서로 다른 개별적인 자료형의 변수들을 하나의 단위로 묶은 새로운 자료형을 구조체라 한다.

구조체는 연관된 멤버로 구성되는 통합 자료형으로 대표적인 유도 자료형이다.

기존 자료형으로 새로이 만들어진 자료형을 유도 자료형이라 한다.

#### 구조체 정의

구조체를 자료형으로 사용하려면 먼저 구조체를 정의해야 한다.

구조체를 사용하려면 먼저 구조체를 만들 구조체 틀을 정의해야 한다.

구조체를 정의하는 방법은 키워드 `struct` 다음에 구조체 태그이름을 기술하고 중괄호를 사용하여 원하는 멤버를 여러 개의 변수로 선언하면 된다.

구조체를 구성하는 하나 하나의 항목을 구조체 멤버 또는 필드라 한다.

구조체 정의는 변수의 선언과는 다르며, 변수선언에서 이용될 새로운 구조체 자료형을 정의하는 구문이다.

구조체 멤버로는 다른 구조체 변수 및 구조체 포인터도 허용된다.

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

struct lecture datastructure;
```

## 13.1 구조체와 공용체

### 구조체 변수 선언과 초기화

구조체가 정의되었다면 이제 구조체형 변수 선언이 가능하다.

새로운 자료형 struct account 형 변수 mine을 선언하려면 struct account mine;으로 선언한다.

```
// struct 구조체태그이름 변수명;  
struct account mine;
```

구조체 변수를 선언하는 다른 방법은 다음과 같이 구조체 정의와 변수 선언을 함께하는 방법이다.

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

## 13.1 구조체와 공용체

### 구조체 변수의 초기화

초기화 값은 다음과 같이 중괄호 내부에서 구조체의 각 멤버 정의 순서대로 초기값을 쉼표로 구분하여 기술한다.

배열과 같이 초기값에 기술되지 않은 멤버 값은 자료형에 따라 기본값인 0, 0.0, '\0' 등으로 저장된다.

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

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

### 구조체의 멤버 접근 연산자 .와 변수 크기

선언된 구조체형 변수는 접근연산자 .를 사용하여 멤버를 참조할 수 있다.

```
//구조체변수이름.멤버
mine.actnum = 1002;
```

실제 구조체의 크기는 멤버의 크기의 합보다 크거나 같다.

```
// 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, 30000 };
    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;
    }
}
```

Microsoft Visual Studio 디버그 콘솔

```
구조체 크기: 24
홍길동 1001 30000.00
이동원 1002 500000.00
```

## 13.1 구조체와 공용체

### 구조체 활용

구조체 멤버로 이미 정의된 다른 구조체 형 변수와 자기 자신을 포함한 구조체 포인터 변수를 사용 할 수 있다.


```
// 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 생성일자
    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);
}
```



C:\ Microsoft Visual Studio 디버그 콘솔

구조체 크기: 40  
[2018, 3, 9]  
홍길동 1001 300000.00

# 13.1 구조체와 공용체

## 구조체 변수의 대입과 동등비교

동일한 구조체형의 변수는 대입문이 가능하다.

즉 구조체 멤버마다 모두 대입할 필요 없이 변수 대입으로 한번에 모든 멤버의 대입이 가능하다.

```
// 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 = { 20180001, "컴퓨터정보공학과", "홍길동" };
    struct student na = { 20180002 };
    struct student bae = { 20180003 };

    //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("[Xd, Xs, Xs]\n", hong.snum, hong.dept, hong.name);
    printf("[Xd, Xs, Xs]\n", na.snum, na.dept, na.name);
    printf("[Xd, Xs, Xs]\n", bae.snum, bae.dept, bae.name);

    struct student one;
    one = bae;
    if (one.snum == bae.snum)
        printf("학번이 Xd으로 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;
}
```

Microsoft Visual Studio 디버그 콘솔

```
나한국
[20180001, 컴퓨터정보공학과, 홍길동]
[20180002, 컴퓨터정보공학과, 나한국]
[20180003, 기계공학과, 배상문]
학번이 20180003으로 동일합니다.
내용이 같은 구조체입니다.
```

## 13.1 구조체와 공용체

### 공용체 활용

공용체(union)는 서로 다른 자료형의 값을 동일한 저장공간에 저장하는 자료형이다.

공용체 변수의 크기는 멤버 중 가장 큰 자료형의 크기로 정해진다.

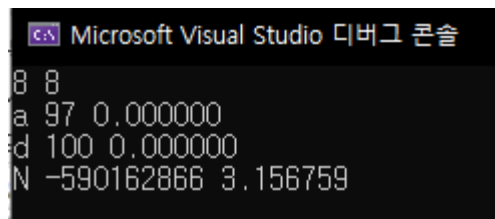
공용체의 멤버는 모든 멤버가 동일한 저장 공간을 사용하므로 동시에 여러 멤버의 값을 저장하여 이용 할 수 없다.

마지막에 저장된 단 하나의 멤버 자료값 만을 저장한다.

공용체의 초기화 값은 공용체 정의 시 처음 선언한 멤버의 초기값으로만 저장이 가능하다.

공용체 변수로 멤버를 접근하기 위해서는 구조체와 같이 접근연산자 .를 사용한다.

```
// 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("Xd Xd\n", sizeof(union data), sizeof(data3));
    //멤버 ch에 저장
    data1.ch = 'a';
    printf("Xc Xd Xf\n", data1.ch, data1.cnt, data1.real);
    //멤버 cnt에 저장
    data1.cnt = 100;
    printf("Xc Xd Xf\n", data1.ch, data1.cnt, data1.real);
    //멤버 real에 저장
    data1.real = 3.156759;
    printf("Xc Xd Xf\n", data1.ch, data1.cnt, data1.real);
    return 0;
}
```



```
Microsoft Visual Studio 디버그 콘솔
8 8
a 97 0.000000
d 100 0.000000
N -590162866 3.156759
```



## 13.2 자료형 재정의

### 자료형 재정의 typedef

Typedef는 이미 사용되는 자료 유형을 새로운 자료형 이름으로 재정의할 수 있도록 하는 키워드이다.

일반적으로 자료형을 재정의하는 이유는 프로그램의 시스템 간 호환성과 편의성을 위해 필요하다.

문장 typedef도 일반 변수와 같이 그 사용 범위를 제한한다.

그러므로 함수 내부에서 재정의되면 선언된 이후의 그 함수에서만 이용이 가능하다.

```
// 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;
}
```

Microsoft Visual Studio 디버그 콘솔

올 예산은 24500000, 이달의 이익은 4600000 입니다.

## 13.2 자료형 재정의

### 구조체 자료형 재정의

Struct를 생략한 새로운 자료형

구조체 struct date가 정의된 상태에서 typedef를 사용하여 구조체 struct date를 date로 재정의가 가능하다.

```
// 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 = { "비주얼 스튜디오 커뮤니티", "MS", "통합개발환경", {2018, 8, 29} };

    printf("제품명 : %s\n", vs.title);
    printf("회사 : %s\n", vs.company);
    printf("종류 : %s\n", vs.kinds);
    printf("출시일 : %d. %d. %d\n", vs.release.year, vs.release.month, vs.release.day);

    return 0;
}
```

C:\ Microsoft Visual Studio 디버그 콘솔

```
제품명 : 비주얼 스튜디오 커뮤니티
회사 : MS
종류 : 통합개발환경
출시일 : 2018. 8. 29
```

# 13.3 구조체와 공용체의 포인터와 배열

## 구조체 포인터

포인터는 각각 자료형 저장 공간의 주소를 저장하듯이 구조체 포인터는 구조체의 주소 값을 저장 할 수 있는 변수이다. 구조체 포인터 멤버 접근연산자 ->는 p->name과 같이 사용한다. 연산식 p->name은 포인터 p가 가리키는 구조체 변수의 멤버 name을 접근하는 연산식이다. 연산식 \*p.name은 접근연산자(.)가 간접연산자(\*)보다 우선순위가 빠르므로 \*(p.name)과 같은 연산식이다.

구조체 변수와 구조체 포인터 변수를 이용한 멤버의 참조

접근 연산식	구조체 변수 os와 구조체 포인터변수 p인 경우의 의미
p->name	포인터 p가 가리키는 구조체의 멤버 name
(*p).name	포인터 p가 가리키는 구조체의 멤버 name
*p.name	*(p.name)이고 p가 포인터이므로 p.name은 문법오류가 발생
*os.name	*(os.name)를 의미하며, 구조체 변수 os의 멤버 포인터 name이 가리키는 변수로, 이 경우는 구조체 변수 os 멤버 강조명의 첫 문자임, 다만 한글인 경우에는 실행 오류
*p->name	*(p->name)을 의미하며, 포인터 p이 가리키는 구조체의 멤버 name이 가리키는 변수로 이 경우는 구조체 포인터 p이 가리키는 구조체의 멤버 강조명의 첫 문자임, 마찬가지로 한글인 경우에는 실행오류

## 13.3 구조체와 공용체의 포인터와 배열

### 구조체 포인터

```
// 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("구조체 크기: %d, 포인터 크기: %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;
}
```

Microsoft Visual Studio 디버그 콘솔

구조체 크기: 32, 포인터 크기: 4

강좌명	강좌구분	학점	시수
운영체제	전공필수	3	3
C프로그래밍	전공선택	3	4
C	전공선택	3	4

## 13.3 구조체와 공용체의 포인터와 배열

### 공용체 포인터

공용체 변수도 포인터 변수 사용이 가능하며, 공용체 포인터 변수로 멤버를 접근하려면 접근연산자 ->를 이용한다. 다만 공용체 변수 value를 가리키는 포인터 p를 선언하여 p가 가리키는 공용체 멤버 ch에 'a'를 저장하는 소스이다.

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

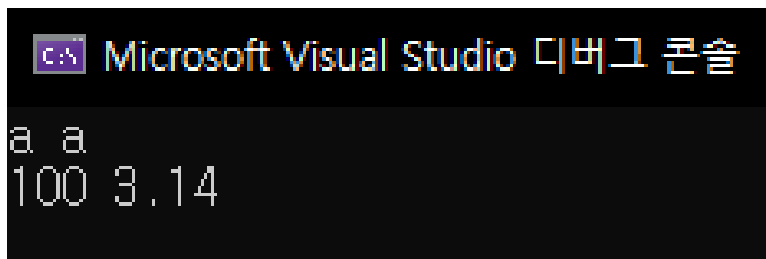
int main(void)
{
    union data
    {
        char ch;
        int cnt;
        double real;
    };

    // 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;
}
```



Microsoft Visual Studio 디버그 콘솔

```
a a
100 3.14
```

## 13.3 구조체와 공용체의 포인터와 배열

### 구조체 배열

다른 배열과 같이 동일한 구조체 변수가 여러 개 필요하면 구조체 배열을 선언하여 이용 할 수 있다.

```
// 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("배열 크기: %d\n\n", arysize);

    printf("%12s    %12s %6s %6s\n", head[0], head[1], head[2], head[3]);

    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;
}
```

Microsoft Visual Studio 디버그 콘솔

배열 크기: 5

강좌명	강좌구분	학점	시수
인간과 사회	교양	2	2
경제학개론	일반선택	3	3
자료구조	전공필수	3	3
모바일 프로그래밍	전공필수	3	4
고급 C프로그래밍	전공선택	3	4

# 13.1 ~ 13.3 프로그래밍 연습

## 문제 1 ~ 문제 2

```
#include <stdio.h>

typedef struct {
    int numerator;
    int denominator;
} fraction;

int main(void)
{
    fraction a;
    fraction b;

    printf("첫번째 분자와 분모를 입력하세요 : ");
    scanf_s("%d %d", &a.numerator, &a.denominator, sizeof(int));

    printf("두번째 분자와 분모를 입력하세요 : ");
    scanf_s("%d %d", &b.numerator, &b.denominator, sizeof(int));

    printf("a = %d/%d, b = %d/%d\n", a.numerator, a.denominator, b.numerator, b.denominator);

    printf("%d/%d * %d/%d의 결과는 %d/%d\n", a.numerator, a.denominator, b.numerator, b.denominator,
           a.numerator * b.numerator, a.denominator * b.denominator);
}
```

### C\N Microsoft Visual Studio 디버그 콘솔

```
첫번째 분자와 분모를 입력하세요 : 4 3
두번째 분자와 분모를 입력하세요 : 5 8
a = 4/3, b = 5/8
4/3 * 5/8의 결과는 20/24
```

감사합니다

20161865 박종근