

제목 : C언어프로그래밍

REPORT



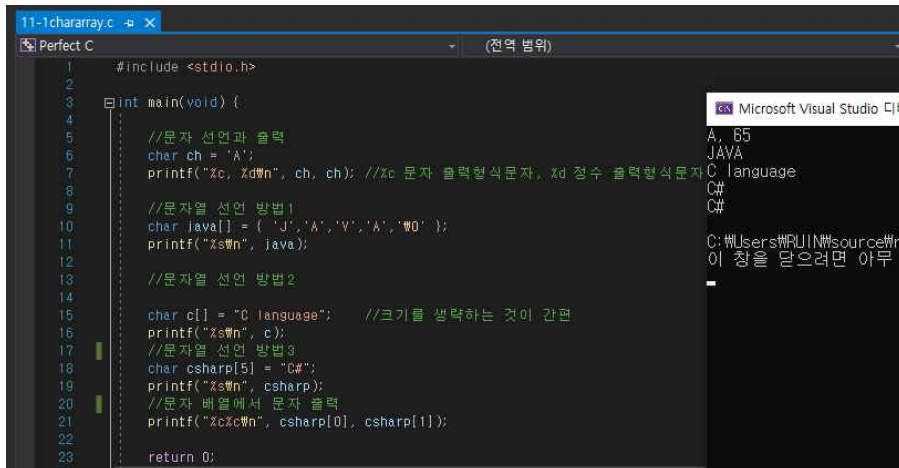
| | |
|------|--------------|
| 과목 | c언어프로그래밍 |
| 담당교수 | 강환수 교수님 |
| 학과 | 컴퓨터정보공학과 |
| 학번 | 20161852 |
| 이름 | 권경환 |
| 제출일 | 2020년 5월 29일 |

【 목 차 】

| | |
|----------------------|----------|
| I. 문자와 문자열 | 1 |
| 1. 문자와 문자열 | |
| 2. 문자열 관련함수 | |
| 3. 여러 문자열 처리 | |
| II. 변수 유효범위 | 2 |
| 1. 전역변수와 지역변수 | |
| 2. 정적변수와 레지스터 변수 | |
| 3. 메모리 영역과 변수 이용 | |
| III. 구조체와 공용체 | 3 |
| 1. 구조체와 공용체 | |
| 2. 자료형 재정의 | |
| 3. 구조체와 공용체의 포인터와 배열 | |

문자와 문자열

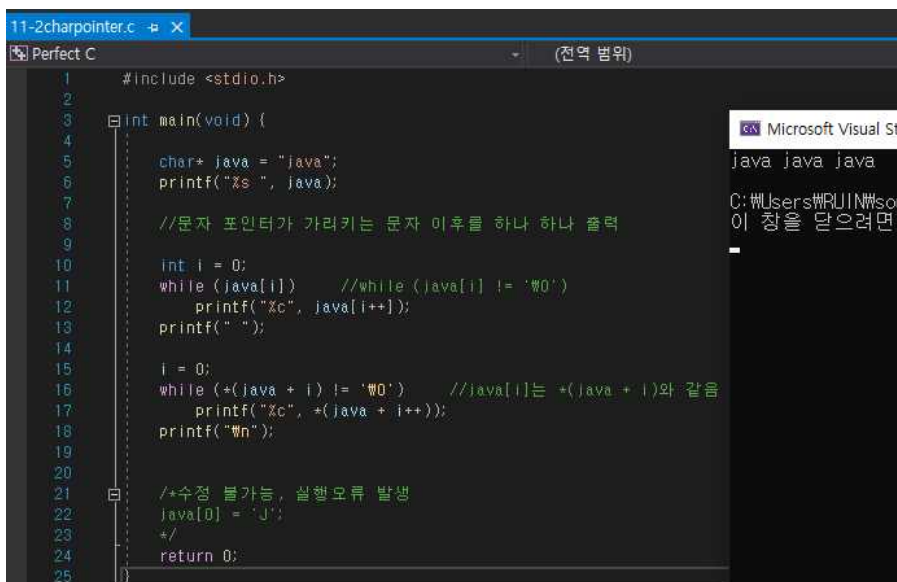
문자열이 저장되는 배열크기는 반드시 저장될 문자수보다 1이 커야한다.
문자열의 마지막을 의미하는 **NULL**문자 '**\0**'가 마지막에 저장되어야함.



```
1 #include <stdio.h>
2
3 int main(void) {
4     //문자 선언과 출력
5     char ch = 'A';
6     printf("%c, %d\n", ch, ch); // %c 문자 출력형식문자, %d 정수 출력형식문자
7
8     //문자열 선언 방법1
9     char java[] = { 'J', 'A', 'V', 'A', '\0' };
10    printf("%s\n", java);
11
12    //문자열 선언 방법2
13
14    char c[] = "C language"; //크기를 생략하는 것이 간편
15    printf("%s\n", c);
16
17    //문자열 선언 방법3
18    char csharp[5] = "C#";
19    printf("%s\n", csharp);
20    //문자 배열에서 문자 출력
21    printf("%c%c\n", csharp[0], csharp[1]);
22
23    return 0;
24 }
```

[위 사진은 문자열 선언방법들을 이용한 실습예제 11-1이다.]

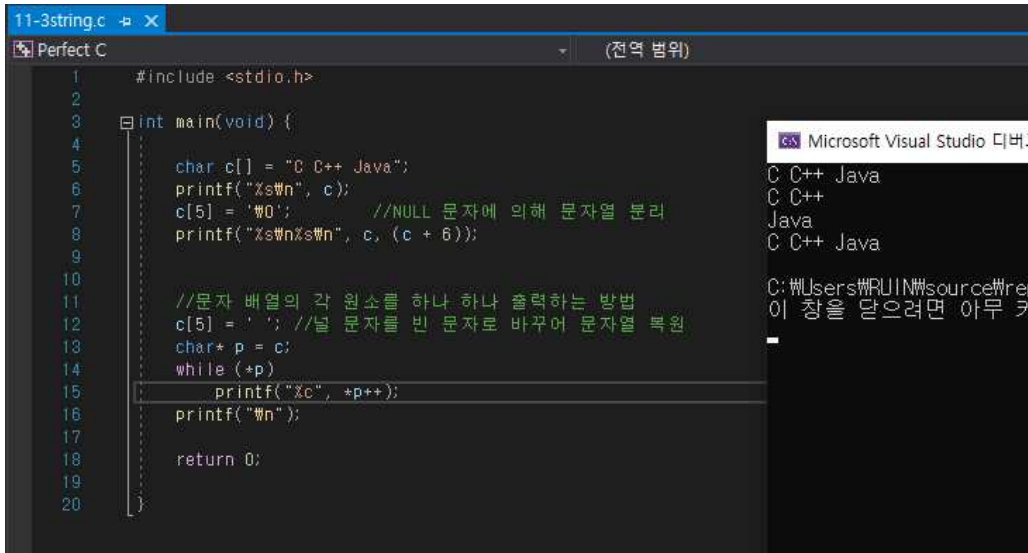
printf()에서 **배열이름** 또는 **문자 포인터**를 사용하여 형식제어문자 **%s**로 문자열을 출력한다.
문자배열을 선언하면서 문자열 **상수**를 **초기화**할 경우, **배열크기**를 생략하면 **간편!**



```
1 #include <stdio.h>
2
3 int main(void) {
4     char* java = "java";
5     printf("%s ", java);
6
7     //문자 포인터가 가리키는 문자 이후를 하나 하나 출력
8
9
10    int i = 0;
11    while (java[i]) //while (java[i] != '\0')
12        printf("%c", java[i++]);
13    printf(" ");
14
15    i = 0;
16    while (*(java + i) != '\0') //java[i]는 *(java + i)와 같은
17        printf("%c", *(java + i++));
18    printf("\n");
19
20    /*수정 불가능, 실행오류 발생
21    java[0] = 'J';
22    */
23    return 0;
24 }
```

[위 사진은 문자포인터 선언을 이용한 실습예제 11-2이다.]

문자포인터에 의한 선언으로는 문자 하나 하나의 **수정**은 **할 수 없다**.
java[i]는 *(java + i)와 동일한 표현방식임.



[위 사진은 문자포인터를 이용한 실습예제 11-3이다.]

printf()에서 **%s**는 문자포인터가 가리키는 위치에서 **NULL**문자까지를 **하나의 문자열로 인식한다.**

```
char c[] = "C C++ Java";
c[5] = '\0';      // c[5]의 NULL문자를 주어서 문자열을 분리함.
printf("%s\n%s\n", c, (c+6));
c[5]= ' ';        // ' '를 다시 저장하면 문자열이 복원됨.
```

코드 13~16행은 **아래 코드와도 같음**

```
① int i = 0;
while (c[i])
    printf("%c", c[i++]);
printf("\n");

② i = 0;
while (*(c+i))
    printf("%c", *(c+ i++));
printf("\n");
```

함수 **getchar()**는 문자의 입력에 사용되며, 함수 **putchar()**는 문자의 출력에 사용된다. **getchar()**는 문자 입력을 위한 함수이며, **라인 버퍼링 방식**을 사용함. 입력한 문자는 임시저장소인 버퍼에 저장되었다가 **enter**키를 만나면 함수는 버퍼에서 문자를 읽기 시작한다.

```

1  #include <stdio.h>
2  #include <conio.h>
3
4  int main(void)
5  {
6      char ch;
7
8      printf("문자를 계속 입력하고 Enter를 누르면 >>\n");
9      while ((ch = getchar()) != 'q')
10         putchar(ch);
11
12     printf("\n문자를 누를 때마다 두 번 출력 >>\n");
13     while ((ch = _getche()) != 'q')
14         putchar(ch);
15
16     printf("\n문자를 누르면 한 번 출력 >>\n");
17     while ((ch = _getch()) != 'q')
18         _putch(ch);
19     printf("\n");
20
21     return 0;
22 }
23

```

Microsoft Visual Studio 디버그 콘솔

```

문자를 계속 입력하고 Enter를 누르면 >>
java
java
python
python
q

문자를 누를 때마다 두 번 출력 >>
jjaavvaad
문자를 누르면 한 번 출력 >>
java
C:\Users\RUIN\source\repos\Perfect_C\Debug
이 창을 닫으려면 아무 키나 누르세요...

```

[위 사진은 **getche**, **getchar**, **getch**를 이용한 실습예제 11-4이다.]

getche()는 **버퍼**를 사용하지 않고 문자를 입력하는 함수이며, 문자 하나를 **바로 바로 입력할 수 있는 함수**이다.

함수를 이용하려면 헤더파일 **conio.h**를 삽입해야함.

getch()는 문자 입력을 위한 함수이며, 입력한 문자가 **화면에 보이지않는 특성**이 있다. 이 함수도 버퍼를 사용하지 않으며, **conio.h** 헤더파일을 삽입해야 사용가능함.

함수 **gets()**는 한 행의 문자열 입력에 유용한 함수이며, 문자열을 출력하는 함수이다. 헤더파일 **stdio.h**를 삽입해야한다.

마지막에 입력된 '**\n**'가 '**\0**'로 교체되어 인자인 배열에 저장된다는 것이다.

puts()도 한 행에 문자열을 출력하는 함수이며, 오류가 발생하면 EOF를 반환함.

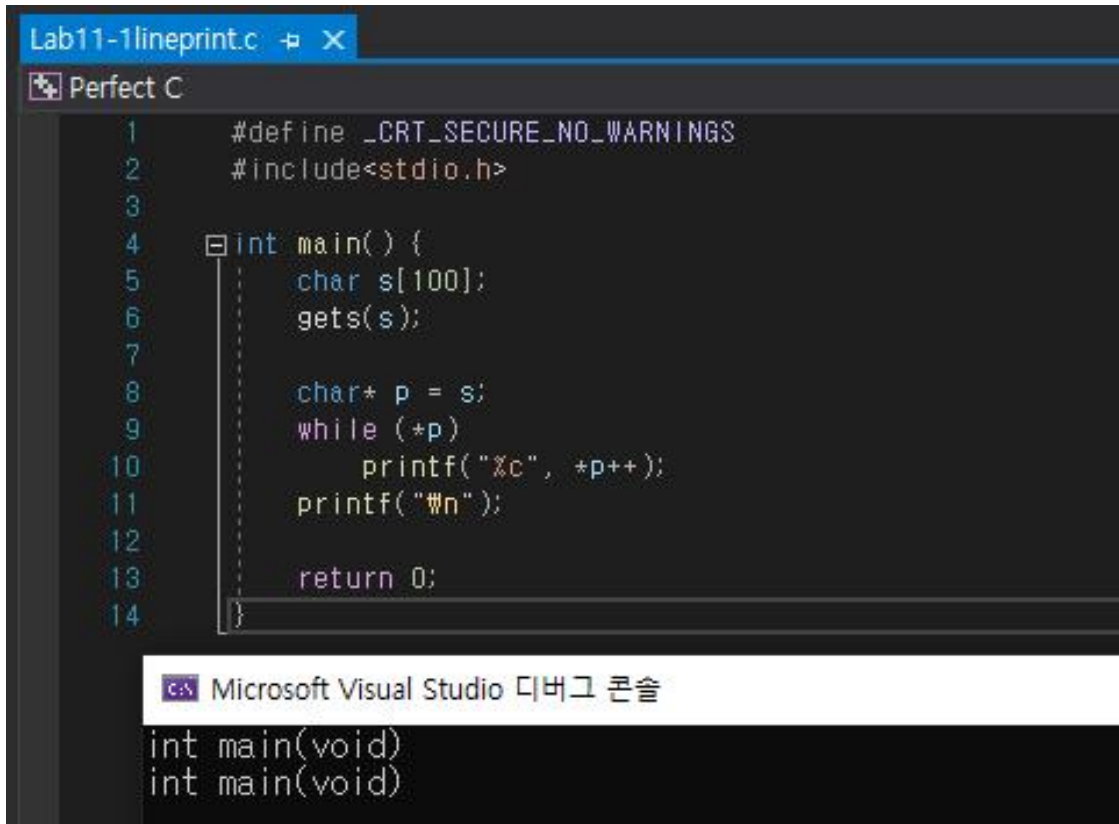
기호상수 **EOF(End of File)**는 파일의 끝이라는 의미이다.

get()와 반대로 문자열의 마지막에 저장된 '**\0**'를 '**\n**'로 교체하여 버퍼에 전송하며, 한 행 출력은 **puts()**함수가 효과적이라고 볼 수 있다.

함수 **printf()**와 **scanf()**는 다양한 입출력에 적합하며, 문자열 입출력 함수 **puts()**와 **gets()**는 처리속도가 빠르다는 장점이 있다.

함수 **gets()**를 사용하여 한 행의 **표준입력을 받아 배열 s에 저장한 후**, 문자포인터를 사용해서 이 배열s에서 **문자 하나 하나를 이동하면서 출력하는 프로그램.**

- ① char 변수 p를 선언하면서 배열 s의 첫 원소를 가리키도록 저장
- ② 포인터 변수 p는 주소값이며 *p는 p가 가리키는 곳의 문자.



```
Lab11-1lineprint.c X
Perfect C
1  #define _CRT_SECURE_NO_WARNINGS
2  #include<stdio.h>
3
4  int main() {
5      char s[100];
6      gets(s);
7
8      char* p = s;
9      while (*p)
10         printf("%c", *p++);
11     printf("\n");
12
13     return 0;
14 }
```

Microsoft Visual Studio 디버그 콘솔

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

[위 사진은 포인터변수와 gets를 이용한 Lab 11-1이다.]

6행에서 gets()를 이용하여 문자배열 s에 표준입력한 행을 저장
8행에서 문자배열에 저장된 한 행을 출력

문자열 관련 함수

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

[문자열 배열에 관한 다양한 함수]

헤더파일 **string.h**를 선언해야 사용할 수 있는 함수들이며, **자료형 size_t**는 비부호정수이며,

void*는 아직
지지않은 다양한
터틀 의미한다.

정해
포인

```

Perfect C (전역 범위)
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(void) {
5
6     char src[50] = "https://www.visualstudio.com";
7     char dst[50];
8
9     printf("문자배열 src= %s\n", src);
10    printf("문자열 크기 strlen(src) = %d\n", strlen(src));
11    memcpy(dst, src, strlen(src) + 1);
12    printf("문자배열 dst = %s\n", dst);
13    memcpy(src, "안녕하세요!", strlen("안녕하세요!") + 1);
14    printf("문자배열 src = %s\n", src);
15
16    char ch = ':';
17    char *ret;
18    ret = memchr(dst, ch, strlen(dst));
19    printf("문자 %c 뒤에는 문자열 %s 이 있다.\n", ch, ret);
20
21    return 0;
22 }

```

선택 Microsoft Visual Studio 디버그 콘솔

```

문자배열 src= https://www.visualstudio.com
문자열 크기 strlen(src) = 28
문자배열 dst = https://www.visualstudio.com
문자배열 src = 안녕하세요!
문자 : 뒤에는 문자열 ://www.visualstudio.com 이 있다.
C:\Users\WRJIN\source\repos\Perfect C\Debug\Perfect C.exe
이 창을 닫으려면 아무 키나 누르세요...

```

[위 사진은 **strlen, memchr, memcpy**를 이용한 실습예제 11-7이다.]

strlen(src)로 문자배열 **src**에 저장된 **문자열 길이를 출력**

포인터위치 **src**에 문자열 상수 “안녕하세요!”를 복사하기 위해서는

(strlen("안녕하세요!)+1) 만큼 복사해야 문자열의 마지막이 **NULL**로 됨.

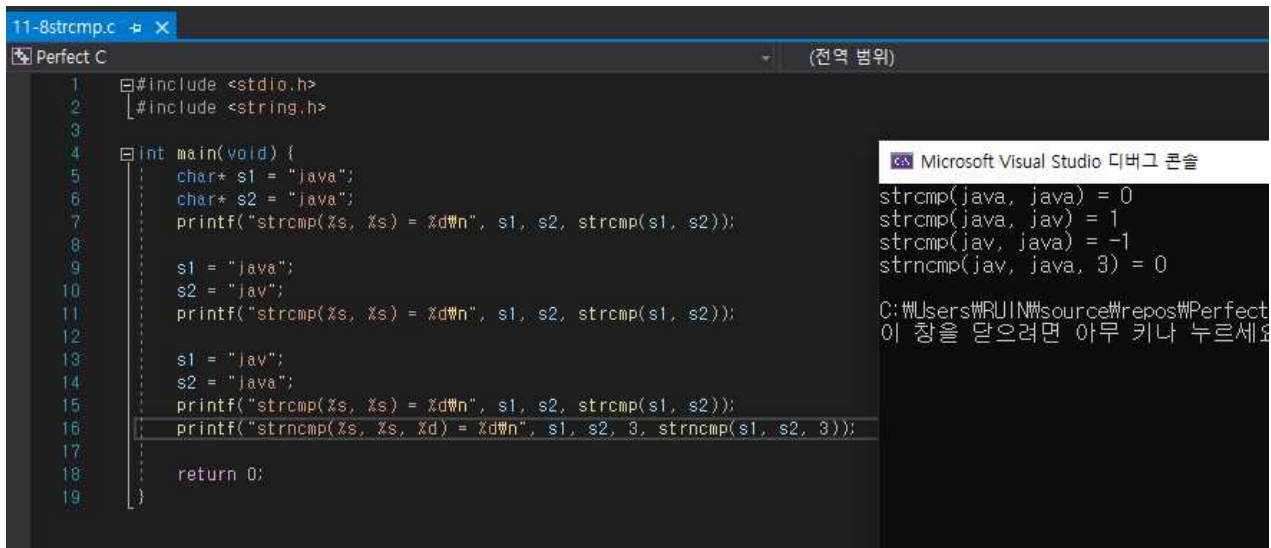
memchr(dst, ch, strlen(dst))호출로 문자열 **dst**에서 문자 **‘:’** 이후의
문자열을 반환하여 변수 **ret**에 저장함.

문자열 비교와 복사, 그리고 문자열 연결 등과 같은 다양한 문자열 처리는 헤더파일 **string.h**에 함수원형으로 선언되 라이브러리 함수로 제공된다.

함수 **strcmp()**는 인자인 두 문자열을 사전상의 순서로 비교하는 함수이며,

함수 **strncmp()**는 두 문자를 비교할 문자의 최대 수를 지정하는 함수이다.

*비교기준은 아스키코드 값이며, 문자가 다른 경우 앞 문자가 작으면 음수, 뒤 문자가 작으면 양수, 같으면 0을 반환함.



The screenshot shows a C program named 11-8strcmp.c in a code editor. The program includes <stdio.h> and <string.h>. The main function declares two character pointers, s1 and s2, and compares them using strcmp and strncmp. The output window shows the results of these comparisons.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(void) {
5     char* s1 = "java";
6     char* s2 = "java";
7     printf("strcmp(%s, %s) = %d\n", s1, s2, strcmp(s1, s2));
8
9     s1 = "java";
10    s2 = "jav";
11    printf("strcmp(%s, %s) = %d\n", s1, s2, strcmp(s1, s2));
12
13    s1 = "jav";
14    s2 = "java";
15    printf("strcmp(%s, %s) = %d\n", s1, s2, strcmp(s1, s2));
16    printf("strncmp(%s, %s, %d) = %d\n", s1, s2, 3, strncmp(s1, s2, 3));
17
18    return 0;
19 }
```

Microsoft Visual Studio 디버그 콘솔

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

C:\Users\RUIN\source\repos\Perfect
이 창을 닫으려면 아무 키나 누르세요

[위 사진은 strcmp를 이용한 실습예제 11-8이다.]

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

함수 **strcpy()**는 앞 인자 문자열 dest에 뒤 인자 문자열 source를 복사한다.



The screenshot shows a C program named 11-9strcpy.c in a code editor. The program includes <stdio.h> and <string.h>. The main function declares two character arrays, dest and source, and copies them using strcpy and strncpy. The output window shows the results of these operations.

```
1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3 #include <string.h>
4
5 int main(void) {
6     char dest[80] = "Java";
7     char source[80] = "C is a language.";
8
9     printf("%s\n", strcpy(dest, source));
10    printf("%s\n", strncpy(dest, "C#", 2));
11
12    printf("%s\n", strncpy(dest, "C#", 3));
13
14    return 0;
15 }
```

Microsoft Visual Studio 디

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

C:\Users\RUIN\source\repos\Perfect
이 창을 닫으려면 아무

[위 사진은 strcpy를 이용한 실습예제 11-9이다.]

함수호출 **strcpy(dest, source)**의 결과는 문자열 source모두를 dest에 복사한 후, 반환값은 문자 포인터 dest임.

함수호출 **strncpy(dest, "C#", 2)**의 결과는 "C#"을 dest에 2바이트만 복사한 후, 반환값은 문자 포인터 dest이므로 C# is a language가 출력됨.

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

이 함수는 앞 인자인 **dest**의 저장공간이 연결된 문자열의 길이를 **모두 저장할 수 있는 공간 보다 부족**하면 문제가 발생한다.

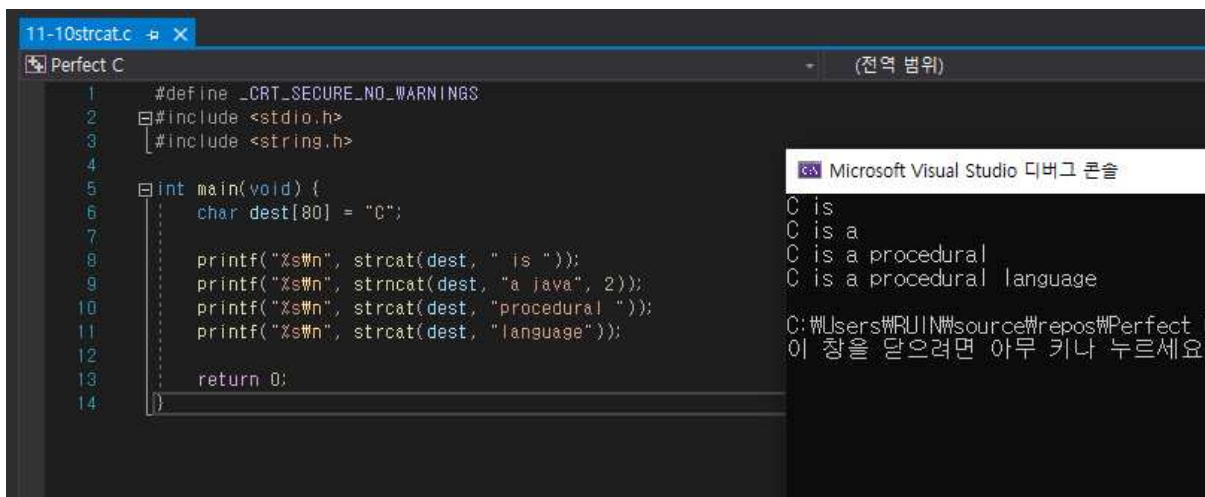
이같은 문제를 예방하기 위한 함수가 **strncat()**함수이다.

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

→ 앞 문자열 dest에 뒤 문자열 source를 연결해 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 연결해 저장함. 지정한 maxn이 문자열 길이보다 크면 null문자까지 연결한다.

The image shows a screenshot of a code editor with a C program. The code defines a character array 'dest' and uses 'strcat' and 'strncat' to append strings. A console window on the right shows the output of the program.

```
11-10strcat.c
Perfect C
1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3 #include <string.h>
4
5 int main(void) {
6     char dest[80] = "C";
7
8     printf("%s\n", strcat(dest, " is "));
9     printf("%s\n", strncat(dest, "a java", 2));
10    printf("%s\n", strcat(dest, "procedural "));
11    printf("%s\n", strcat(dest, "language"));
12
13    return 0;
14 }
```

Microsoft Visual Studio 디버그 콘솔

```
C is
C is a
C is a procedural
C is a procedural language
C:\Users\#RUI\#source\#repos\#Perfect C
이 창을 닫으려면 아무 키나 누르세요.
```

[위 사진은 **strcat**를 이용한 실습예제 11-10이다.]

char dest[5] = "C";

char *destc = "C";

strcpy(dest, "Java language"); //실행시 오류 발생

문자열관련 함수에서 **자료형이(char*)**인 인자에는 문자열 상수를 **사용할 수 없다.**

그래서 **자료형(const char*)**인 인자에는 문자열 상수를 **사용할 수 있다.**

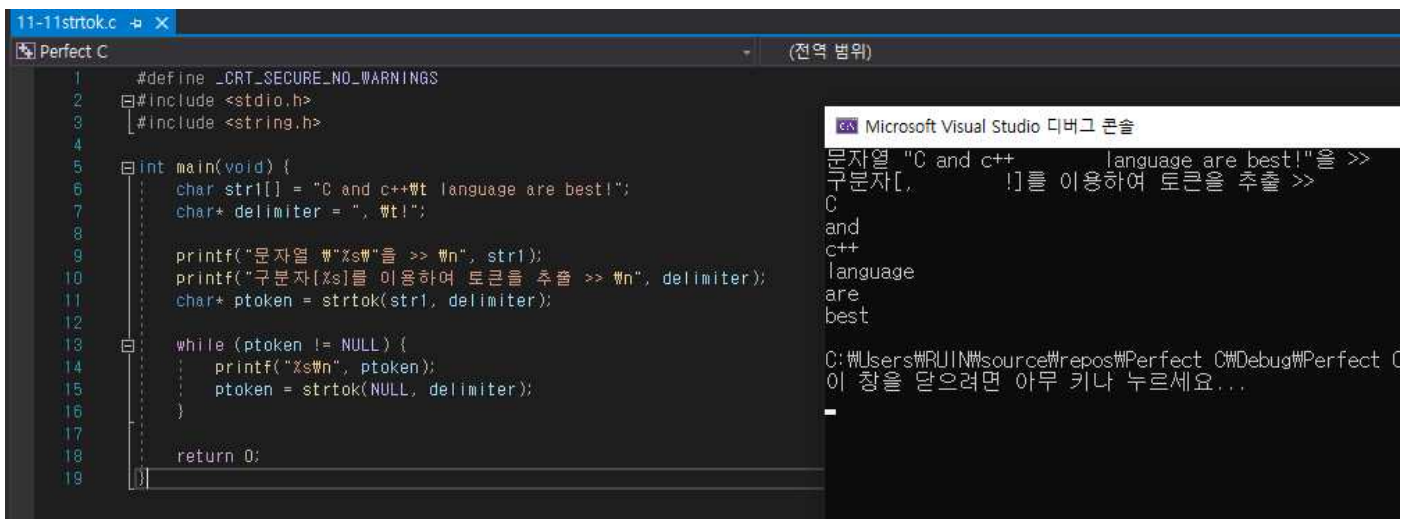
함수 **strtok()**은 문자열에서 구분자인 문자를 여러 개 지정하여 **토큰을 추출하는 함수**이다.

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

-> 앞 문자열 **str**에서 뒤 문자열 **delim**을 구성하는 구분자를 기준으로 순서대로 토큰을 추출하여 **반환하는 함수**이며, 뒤 문자열은 **수정될 수 없다**.

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

-> 마지막 인자인 **context**는 함수 호출에 사용되는 **위치 정보를 위한 인자**이며, Visual C++에서는 앞으로 **함수 strtok_s()의 사용을 권장한다**.



```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <string.h>
4
5  int main(void) {
6      char str1[] = "C and c++ language are best!";
7      char* delimiter = " , !";
8
9      printf("문자열 \"%s\"을 >> \n", str1);
10     printf("구분자[%s]를 이용하여 토큰을 추출 >> \n", delimiter);
11     char* ptoken = strtok(str1, delimiter);
12
13     while (ptoken != NULL) {
14         printf("%s\n", ptoken);
15         ptoken = strtok(NULL, delimiter);
16     }
17
18     return 0;
19 }
```

Microsoft Visual Studio 디버그 콘솔

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

C:\Users\RUIN\source\repos\Perfect C\Debug\Perfect C
이 창을 닫으려면 아무 키나 누르세요...

[위 사진은 **strtok**를 이용한 실습예제 11-11이다.]

strtok(str1, delimiter)호출에 의해 분리되는 문자열 토큰이 저장될 문자 포인터 선언하여 **strtok() 호출값**을 저장.

분리된 토큰일 **NULL**이 아니면 반복문 실행, **분리된 토큰을 한 줄에 출력**

두 번째 토큰 추출에서 **strtok(NULL, delimiter)**으로 호출하여 **반환값을 ptoken에 저장함**.

함수 **strlwr()**은 인자를 모두 소문자로 변환하여 반환한다.

함수 **strupr()**은 인자를 모두 대소문자로 변환하여 반환한다.

여러 문자열 처리

여러개의 문자열을 처리하는 하나의 방법은 **문자 포인터 배열**을 이용하는 방법이다.

ex) `char *ex[] = {"JAVA", "C#", "C++"};`
`printf("%s ", ex[0]); printf("%s ", ex[1]);`

또 다른 방법은 문자의 **이차원 배열**을 이용하는 방법이다.

ex) `char ex[][5] = {"JAVA", "C#", "C++"};`
`printf("%s ", ex[0]); printf("%s ", ex[1]);`

```
#include <stdio.h>

int main(void) {
    char *pa[] = {"JAVA", "C#", "C++"};
    char ca[][5] = { "JAVA", "C#", "C++" };

    printf("%s ", pa[0]); printf("%s ", pa[1]); printf("%s\n", pa[2]);
    printf("%s ", ca[0]); printf("%s ", ca[1]); printf("%s\n", ca[2]);

    printf("%c %c %c\n", pa[0][1], pa[1][1], pa[2][1]);
    printf("%c %c %c\n", ca[0][1], ca[1][1], ca[2][1]);

    return 0;
}
```

Microsoft Visual
JAVA C# C++
JAVA C# C++
A # +
A # +
C:\Users\RUIIN\...
이 창을 닫으려...

[위 사진은 문자포인터 배열을 이용한 실습예제 11-13이다.]

```
11-14commandarg.c - X
Perfect C (전역 범위)

#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;
}
```

Microsoft Visual Studio 디버그 콘솔
실행 명령행 인자(command line arguments) >>
argc = 1
argv[0] = C:\Users\RUIIN\source\repos\Perfect C\Debug\Perfect C.exe
C:\Users\RUIIN\source\repos\Perfect C\Debug\Perfect C.exe(프로세스 24...
이 창을 닫으려면 아무 키나 누르세요...

[위 사진은 명령행인자를 이용한 실습예제 11-13이다.]

명령행에서 입력하는 문자열을 프로그램으로 전달하는 방법이 **명령행인자를 사용하는 방법**
명령행인자를 처리하려면 **main()**의 매개변수를 “**int argc, char *argv[]**”로
기술하는데, **argc**에 **인자의 수**, **argv**에는 인자인 **여러개의 문자열의 포인터가**
저장된 배열이 전달됨.

전역변수와 지역변수

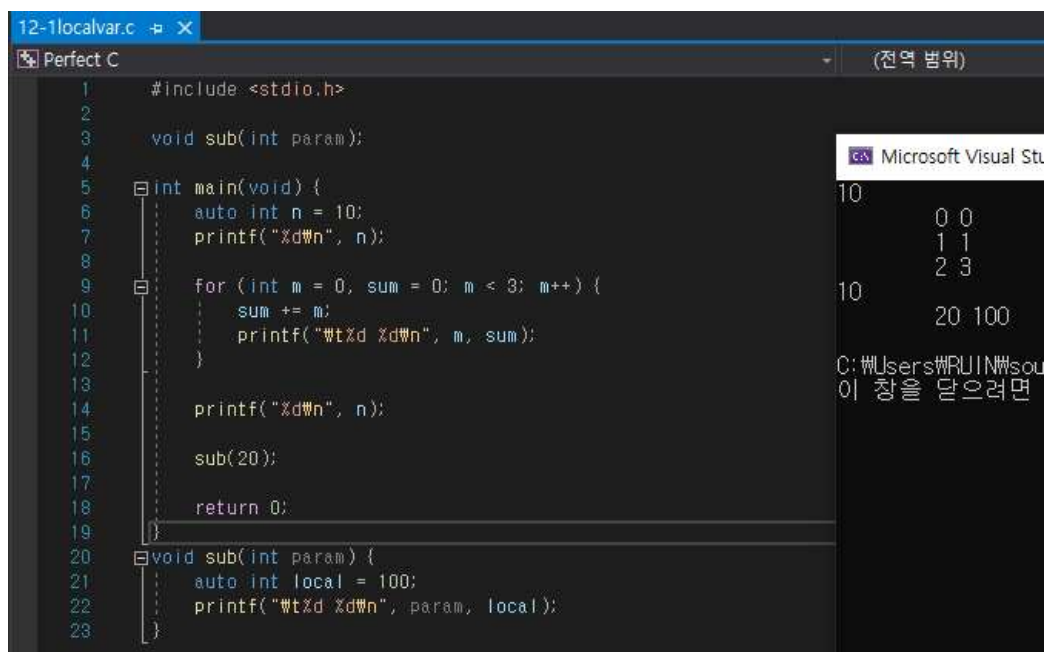
변수의 참조가 유효한 범위를 **변수의 유효범위(scope)**라 한다.

지역 유효범위와 **전역 유효범위**로 나눌 수 있는데, **지역 유효범위**는 **함수** 또는 **블록 내부**에서 선언되어 그 지역에서 변수의 참조가 가능한 범위이다.

①함수의 **매개변수**도 **지역변수**와 같다.

②**지역변수**는 선언후 초기화하지 않으면 쓰레기값이 저장되므로 주의해야한다.

③지역변수는 **자동변수**라고도 불리며, 선언된 부분에서 자동으로 생성되고 함수나 블록이 종료되는 순간메모리에서 **자동으로 제거**된다.



```
12-1localvar.c
Perfect C
(전역 범위)

1  #include <stdio.h>
2
3  void sub(int param);
4
5  int main(void) {
6      auto int n = 10;
7      printf("%d\n", n);
8
9      for (int m = 0, sum = 0; m < 3; m++) {
10         sum += m;
11         printf("%t%d %d\n", m, sum);
12     }
13     printf("%d\n", n);
14
15     sub(20);
16
17     return 0;
18 }
19
20 void sub(int param) {
21     auto int local = 100;
22     printf("%t%d %d\n", param, local);
23 }
```

Microsoft Visual Stu

10

0 0

1 1

2 3

10

20 100

C:\Users\WRUIN\source

이 창을 닫으려면

[위 사진은 지역변수를 사용한 실습예제 12-1이다.]

6행에서는 **지역변수 n**을 선언하여 10을 저장하였으며 ,유효범위는 6행에서 18행까지이다.

20행에서는 **매개변수 param**도 **지역변수**로 사용되는 모습을 볼 수 있음.

유효범위는 20행에서 23행까지이다.

전역변수는 **함수 외부**에서 선언되는 변수이다.

①전역변수는 **외부변수**라고도 불린다.

②전역변수는 일반적으로 **프로젝트의 모든 함수나 블록**에서 참조할 수 있다.

③전역변수는 선언되면 자동으로 초기값이 **자료형에 맞는 0**으로 지정된다.

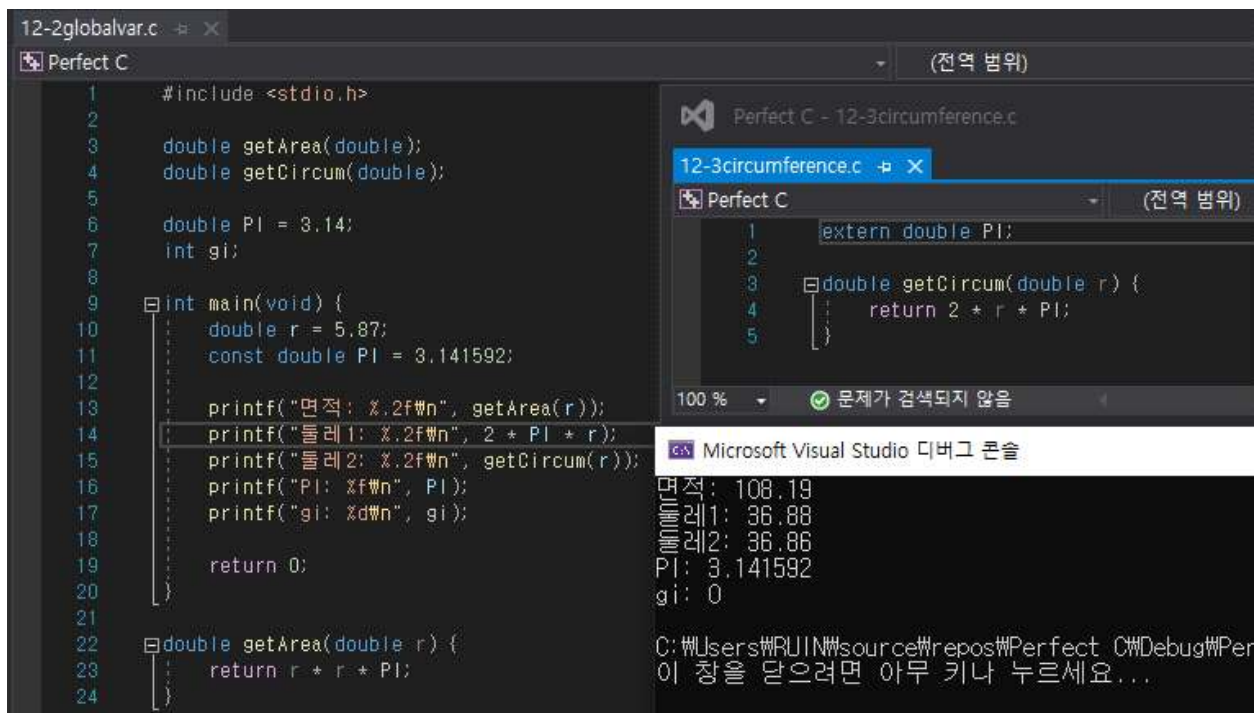
④함수나 블록에서 전역변수와 **같은 이름**으로 지역변수를 선언할 수 있다.

이런 경우, 함수내부나 블록에서 그 이름을 참조하면 지역변수로 인식한다.

그러므로 가능한 같은 이름이 아니게 선언한다.

⑤전역변수는 **프로젝트의 다른 파일**에서도 참조가 가능하다.

⑥전역변수에 예상하지 못한값이 저장된다면 프로그램 어느 부분에서 수정되었는지 알기 어려운 단점이 있다.



```
12-2globalvar.c
Perfect C
1 #include <stdio.h>
2
3 double getArea(double);
4 double getCircum(double);
5
6 double PI = 3.14;
7 int gi;
8
9 int main(void) {
10     double r = 5.87;
11     const double PI = 3.141592;
12
13     printf("면적: %.2f\n", getArea(r));
14     printf("둘레 1: %.2f\n", 2 * PI * r);
15     printf("둘레 2: %.2f\n", getCircum(r));
16     printf("PI: %f\n", PI);
17     printf("gi: %d\n", gi);
18
19     return 0;
20 }
21
22 double getArea(double r) {
23     return r * r * PI;
24 }
```

```
Perfect C - 12-3circumference.c
12-3circumference.c
Perfect C
1 extern double PI;
2
3 double getCircum(double r) {
4     return 2 * r * PI;
5 }
```

100 % 문제 없음

Microsoft Visual Studio 디버그 콘솔

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

C:\Users\WRUIN\source\repos\Perfect C\Debug\Per...
이 창을 닫으려면 아무 키나 누르세요...

[위 사진은 전역변수를 사용한 실습예제 12-2,3이다.]

전역변수 PI를 선언하여 3.14저장 **유효범위는 전체**이다.

전역변수 gi를 선언했으나 초기값을 대입하지않아도 **기본값이 0**이 **저장**이 됨.

12-2예제에 선언된 전역변수 PI를 사용하려면 **extern**을 사용한 참조선언이 필요하므로 **extern double PI** 문장으로 참조선언함.

r은 **매개변수**이며, PI는 12-2의 전역변수이다.

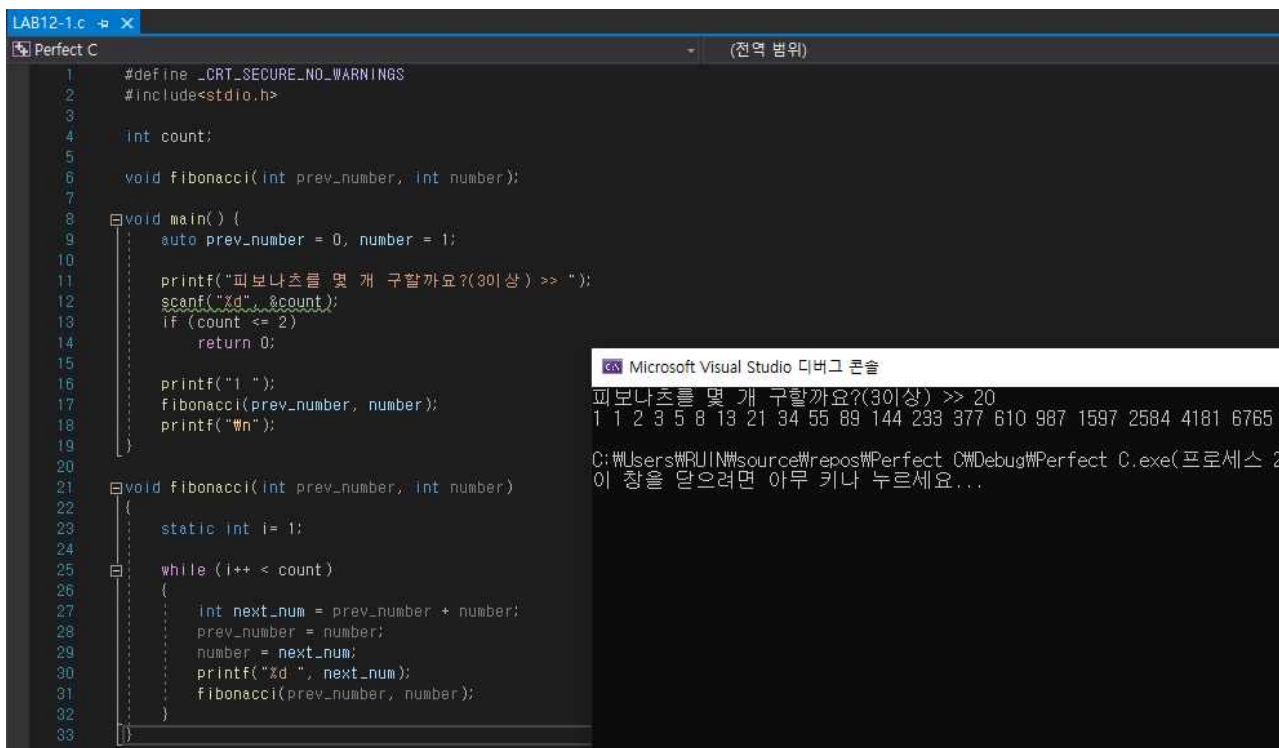
피보나츠의 수는 1,1로 시작하여 이전 두수를 더하는 수이다.

이 프로그램에서 **표준입력**으로 받은 3이상의 정수를 **전역변수 count**에 저장한후, **재귀함수**인 fibonacci()에서 count -1개의 피보나츠의 수를 출력하도록한다.

[조건]

함수 fibonacci()의 **매개변수**는 (**int prev_number, int number**)으로 이전 두 정수가 인자이며, 자기 자신을 호출하는 **재귀함수**

함수 fibonacci()에서 자기 자신이 호출된 수를 저장하는 **정적 지역변수 i**를 사용하며 초기값은 1로 지정.



```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include<stdio.h>
3
4  int count;
5
6  void fibonacci(int prev_number, int number);
7
8  void main() {
9      auto prev_number = 0, number = 1;
10
11      printf("피보나츠를 몇 개 구할까요?(3이상) >> ");
12      scanf("%d", &count);
13      if (count <= 2)
14          return 0;
15
16      printf("1 ");
17      fibonacci(prev_number, number);
18      printf("\n");
19  }
20
21  void fibonacci(int prev_number, int number)
22  {
23      static int i = 1;
24
25      while (i++ < count)
26      {
27          int next_num = prev_number + number;
28          prev_number = number;
29          number = next_num;
30          printf("%d ", next_num);
31          fibonacci(prev_number, number);
32      }
33  }
```

Microsoft Visual Studio 디버그 콘솔

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

C:\Users\WRJIN\source\repos\Perfect C\Debug\Perfect C.exe(프로세스 2
이 창을 닫으려면 아무 키나 누르세요...

[위 사진은 지역변수와 전역변수를 사용한 Lab 12-1이다.]

4행은 전역변수 count를 선언함.

8행은 함수원형 fibonacci을 선언함.

12행은 전역변수를 표준입력으로 저장하는 구문이다.

23행은 정적 지역변수 i이다.

25행에서 전역변수 count와 함수의 정적 지역변수를 비교함.

정적 변수와 레지스터 변수

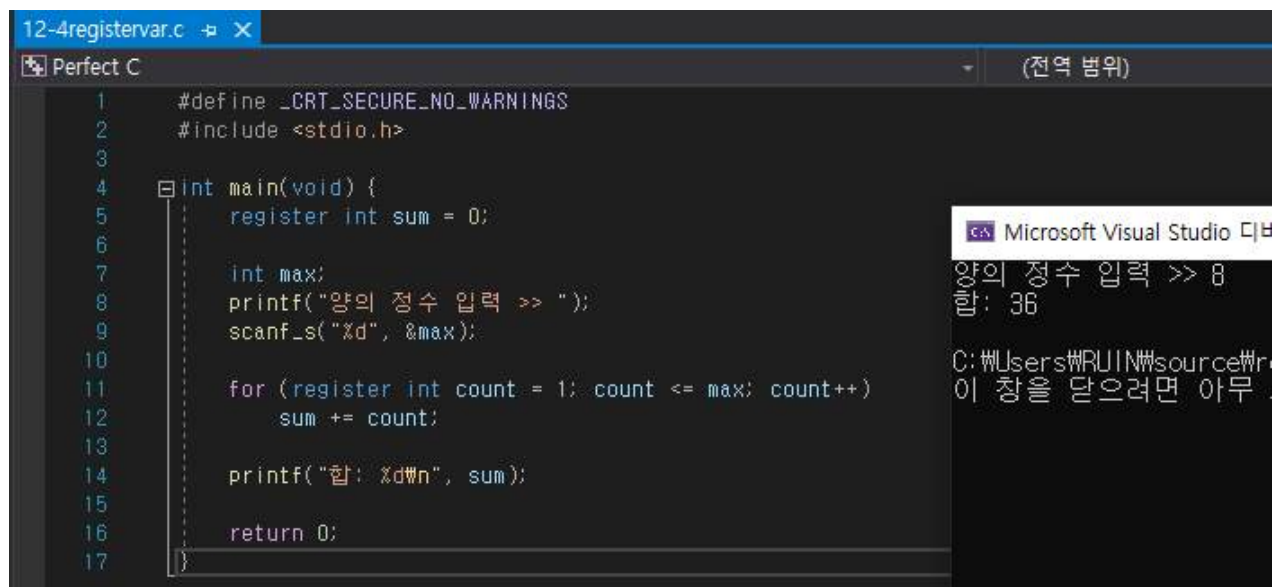
변수는 4가지의 기억부류인 `auto`, `register`, `static`, `extern`에 따라 할당되는 메모리 영역이 결정되고 메모리의 할당과 제거 시기가 결정된다.

| 기억부류 종류 | 전역 | 지역 |
|-----------------------|----|----|
| <code>auto</code> | X | O |
| <code>register</code> | X | O |
| <code>static</code> | O | O |
| <code>extern</code> | O | X |

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

[키워드 `register`]

- ① 레지스터 변수는 **키워드 `register`**를 자료형 앞에 넣어 선언한다.
- ② 레지스터 변수는 **지역변수**에만 이용이 가능하다.
- ③ 레지스터 변수는 일반 메모리에 할당되는 변수가 아니므로 **주소연산자 `&`**를 사용할 수 없다.
- ④ 레지스터 변수는 처리속도를 증가시키려는 변수에 이용한다. 특히 반복문의 횟수를 제어하는 제어변수에 이용하면 효과적이다.



```
12-4registervar.c
Perfect C (전역 범위)
1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3
4 int main(void) {
5     register int sum = 0;
6
7     int max;
8     printf("양의 정수 입력 >> ");
9     scanf_s("%d", &max);
10
11     for (register int count = 1; count <= max; count++)
12         sum += count;
13
14     printf("합: %d\n", sum);
15
16     return 0;
17 }
```

Microsoft Visual Studio 디버깅 콘솔 출력:
양의 정수 입력 >> 8
합: 36
C:\Users\RUIN\source\...
이 창을 닫으려면 아무...

[위 사진은 `register`를 사용한 실습예제 12-2,3이다.]

레지스터 변수 `sum` 선언하면서 초기값 저장.

레지스터 블록 지역 변수 `count` 선언하면서 초기값으로 1저장, `count`는 `max`까지 반복하면서 함수몸체인 `sum += count`를 실행.

[키워드 static]

- ① 변수 선언에서 자료형앞에 키워드 static을 넣어 정적변수를 선언할 수 있다.
- ② 초기 생성된 이후 메모리에서 제거되지않으므로 지속적으로 저장값을 유지하거나 수정할 수 있는 특성이 있다.
- ③ 정적변수는 초기값을 지정하지 않으면 자동으로 자료형에 따라 0이나 '\0'또는 NULL값이 저장된다.

The screenshot shows a C program in a code editor and its debug console output. The code defines a static variable `sindex` and an automatic variable `aindex` in a function `increment`. The console output shows that `sindex` increases from 1 to 3 across three calls, while `aindex` is always 1.

```
1 #include <stdio.h>
2
3 void increment(void);
4
5 int main(void) {
6     for (int count = 0; count < 3; count++)
7         increment();
8 }
9 void increment(void) {
10     static int sindex = 1;
11     auto int aindex = 1;
12
13     printf("정적 지역변수 sindex: %2d,\\n", sindex++);
14     printf("자동 지역변수 aindex: %2d\\n", aindex++);
15 }
```

Microsoft Visual Studio 디버그 콘솔

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

C:\\Users\\WRUIN\\source\\repos\\Perfect C\\Debug\\Perfect C.exe(3)
이 창을 닫으려면 아무 키나 누르세요...

[위 사진은 static을 사용한 실습예제 12-5이다.]

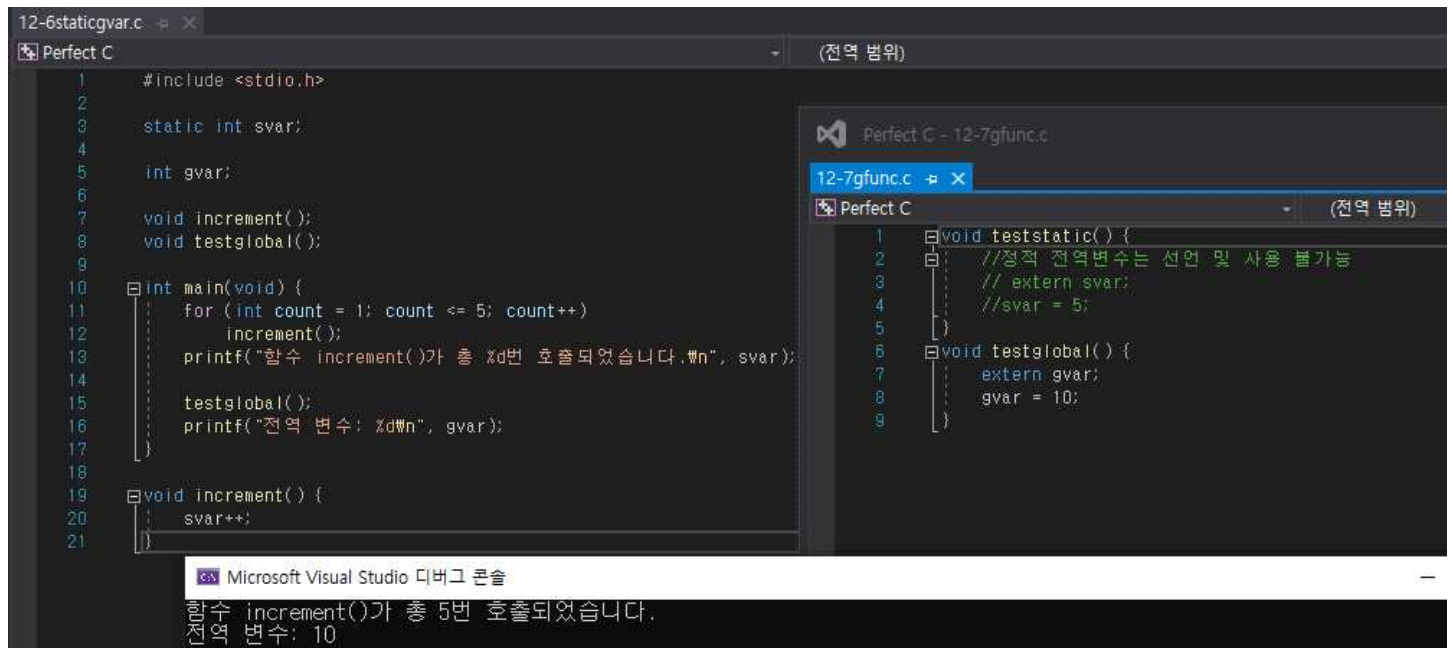
정적지역변수 `sindex` 선언하면서 초기값으로 1저장, 첫 번째 호출되어 실행될 때 초기값으로 `sindex`가 1이 저장되고 **함수가 종료되어도 그 변수는 계속 유지**되는 특성. **자동 지역변수** `aindex` 선언하면서 초기값으로 1저장, 함수가 호출되어 실행될때마다 다시 선언되고 초기값으로 `aindex`가 1이 저장되며, 함수가 종료되면 이 변수는 **메모리에서 사라지므로**, 호출될 때마다 1만 사용됨.

함수나 블록에서 정적으로 선언되는 변수가 **정적 지역변수**이다.

- ① 정적 지역변수는 **함수나 블록을 종료해도 메모리에서 제거되지 않고 계속 메모리에 유지관리되는 특성**이 있다.
- ② 함수에서 이전에 호출되어 **저장된 값을 유지하여 이번 호출에 사용**될 수 있다.

함수 외부에서 정적으로 선언되는 변수가 **정적 전역변수**이다.

- ① 정적 전역변수는 **선언된 파일 내부에서만 참조가 가능한 변수**이다.
- ② 프로그램이 크고 복잡하면 전역변수의 사용은 원하지 않는 전역변수의 수정과 같은 부작용의 위험성이 항상 존재한다.



[위 사진은 정적변수 선언과 사용한 실습예제 12-6,7이다.]

3행에서 키워드 **static**을 사용하여 정적 지역변수 svar선언하므로 이 파일에서만 사용가능하며, 초기값이 없어도 기본값인 0으로 저장.

5행에서 키워드 **static**이 없으므로 전역변수 gvar 선언하므로 이 프로젝트의 모든 파일에서 사용가능하며, 초기값이 없어도 기본값인 0으로 저장.

8행은 함수원형

11행은 변수 count는 자동 지역변수로 for문 내부에서만 사용 가능하며, 1에서 5까지 변하면서 몸체인 함수 increment()를 호출

13행은 정적 전역변수인 svar호출

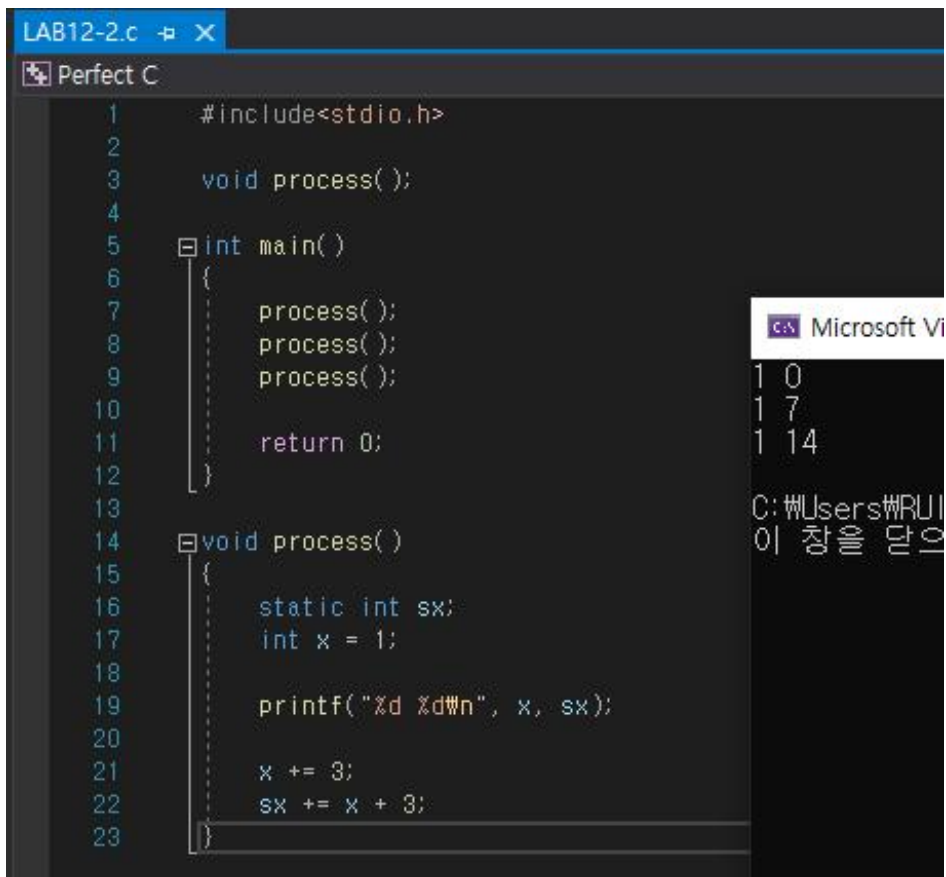
16행은 전역변수인 gvar출력

19행은 함수 increment()함수 헤더

20행은 정적 전역변수인 svar호출

함수 main()에서 함수 **process()**를 세 번 호출한다.

함수 process에는 **지역변수**와 **정적 지역변수**가 선언되고, 간단한 연산과 함께 출력해보도록 한다.



```
LAB12-2.c
Perfect C
1  #include<stdio.h>
2
3  void process();
4
5  int main()
6  {
7      process();
8      process();
9      process();
10
11     return 0;
12 }
13
14 void process()
15 {
16     static int sx;
17     int x = 1;
18
19     printf("%d %d\n", x, sx);
20
21     x += 3;
22     sx += x + 3;
23 }
```

Microsoft Vi
1 0
1 7
1 14
C:\Users\WRII
이 창을 닫으

[위 사진은 정적지역변수와 전역변수를 사용한 실습예제 Lab12-2이다.]

16행에서 정적변수를 선언

17행에서 지역변수를 선언

메모리 영역과 변수 이용

메인메모리의 영역은 프로그램 실행과정에서 데이터 영역, 힙 영역, 스택 영역 세 부분으로 나뉜다.

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

데이터 영역은 전역변수와 정적변수가 할당되는 저장공간이다.

힙 영역은 동적 할당되는 변수가 할당되는 저장공간이다.

스택 영역은 함수 호출에 의한 형식 매개변수 그리고 함수 내부의 지역변수가 할당되는 저장공간이며, 메모리주소가 높은값에서 낮은 값으로 저장 장소가 할당된다.

그러므로 함수 호출과 종료에 따라 높은 주소에서 낮은 주소로 메모리가 할당되었다가 다시 제거되는 작업이 반복된다.

[변수의 종류]

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

전역변수와 정적변수는 모두 생존 기간이 프로그램 시작시에 생성되어 프로그램 종료시에 제거된다.

다만 자동 지역변수와 레지스터 변수는 함수가 시작되는 시점에서 생성되어 함수가 종료되는 시점에서 제거된다.

[변수의 유효 범위]

| 구분 | 종류 | 메모리할당 시기 | 동일 파일 외부 함수에서의 이용 | 다른파일 외부 함수에서의 이용 | 메모리제거 시기 |
|----|---------|----------|----------------------|---------------------|----------|
| 전역 | 전역변수 | 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 |

구조체와 공용체

연관성이 있는 정수나 문자, 실수나 포인터 그리고 이들의 배열등 서로 다른 개별적인 자료형의 변수들을 하나의 단위로 묶은 새로운 자료형을 **구조체**라한다.

[구조체 정의]

구조체를 자료형으로 사용하려면 먼저 구조체를 만들 **구조체 틀을 정의**하여야한다.

구조체를 정의하는 방법은 **키워드 struct** 다음에 **구조체 태그이름을 기술**하고 중괄호를 이용하여 원하는 멤버를 **여러개의 변수로 선언하는 구조**이다.

구조체를 구성하는 하나 하나의 항목을 **구조체 멤버** 또는 **필드**라 한다.

구조체 멤버로는 **일반변수**, **포인터변수**, **배열**, 다른 **구조체 변수** 및 **구조체 포인터**도 허용된다.

[구조체 변수 선언]

struct 구조체태그이름 **변수명**;

struct 구조체태그이름 **변수명1**, **변수명2**, **변수명3**; <- 여러 변수의 선언도 가능하다.

```
ex) struct account{
char name[12];
int actnum;
double balance;
} myaccount; <- 변수 myaccount는 struct account형 변수로 선언된다.
```

```
struct account youraccount; <- 변수 youraccount도 struct
account형 변수로 선언된다.
```

①구조체변수 선언 구문에서 **구조체 태그 이름을 생략**할 수 있다.

그러나 구조체 태그이름이 없는 변수 선언방법은 이 구조체와 동일한 자료형의 변수를 더 이상 선언할 수 없다.

[구조체 변수의 초기화]

struct 구조체태그이름 **변수명** = {초기값1, 초기값2, 초기값3};

[구조체의 멤버 접근연산자. 와 변수크기]

선언된 구조체형 변수는 **접근연산자.**를 사용하여 **멤버를 참조**할 수 있다.

구조체변수이름.멤버

```
mine.actnum=1002; mine.balance=300000;
```

①실제 구조체의 크기는 멤버의 크기의 합보다 크거나 작다.


```

1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <string.h>
4
5  struct account
6  {
7      char name[12];
8      int actnum;
9      double balance;
10 };
11
12 int main(void)
13 {
14     struct account mine = { "홍길동", 1001, 300000 };
15     struct account yours;
16
17     strcpy(yours.name, "이동원");
18     yours.actnum = 1002;
19     yours.balance = 500000;
20
21     printf("구조체 크기: %d\n", sizeof(mine));
22     printf("%s %d %.2f\n", mine.name, mine.actnum, mine.balance);
23     printf("%s %d %.2f\n", yours.name, yours.actnum, yours.balance);
24
25     return 0;
26 }

```

Microsoft Visual Studio 디버그 콘솔

```

구조체 크기: 24
홍길동 1001 300000.00
이동원 1002 500000.00
C:\Users\WRJIN\source\repos\13-1structbasic\13-1structbasic.c
이 창을 닫으려면 아무 키나 누르십시오.

```

[위 사진은 구조체 정의와 변수를 사용한 실습예제 13-1이다.]

7행에서 구조체 **account**를 정의함.

14행에서 **char** 배열인 **name**으로는 바로 문자열 상수 “이동원”으로는 **대입이 불가능함**. 참조연산자,를 사용하여 구조체 변수 **mine,yours**에서 모든 멤버를 참조하여 출력함.

구조체 멤버로 이미 정의된 다른 구조체형 변수와 자기 자신을 포함한 구조체 포인터 변수를 사용할 수 있다.

```

1  #include <stdio.h>
2  #include <string.h>
3
4  struct date
5  {
6      int years;
7      int month;
8      int day;
9  };
10
11 struct account
12 {
13     struct date open;
14     char name[12];
15     int actnum;
16     double balance;
17 };
18
19 int main(void)
20 {
21     struct account me = { { 2018, 3, 9 }, "홍길동", 1001, 300000 };
22
23     printf("구조체 크기: %d\n", sizeof(me));
24     printf("[%d, %d, %d]\n", me.open.years, me.open.month, me.open.day);
25     printf("%s %d %.2f\n", me.name, me.actnum, me.balance);
26 }

```

Microsoft Visual Studio 디버그 콘솔

```

구조체 크기: 40
[2018, 3, 9]
홍길동 1001 300000.00
C:\Users\WRJIN\source\repos\13-2netstedstruct\13-2netstedstruct.c
이 창을 닫으려면 아무 키나 누르십시오.

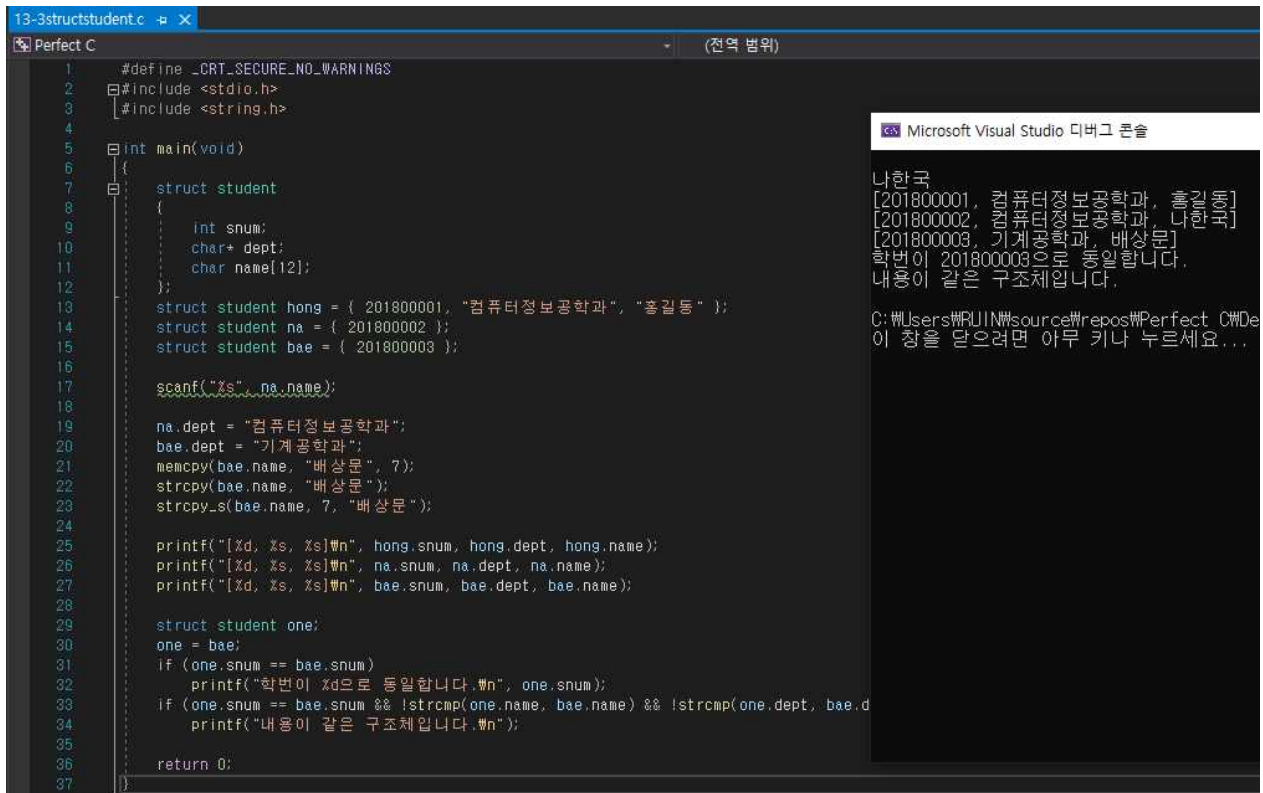
```

[위 사진은 구조체 멤버로 다른 변수를 사용한 실습예제 13-2이다.]

14행 **struct account** 내부 멤버로 구조체 **struct date**형으로 변수 **open**을 선언하며, 이 변수는 계좌 개설 날짜를 의미함.

동일한 구조체형의 변수는 **대입문이 가능하다**.

즉, 구조체 멤버마다 모두 대입할 필요없이 **변수대입으로 한 번에 모든 멤버의 대입이 가능함**.



```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <string.h>
4
5  int main(void)
6  {
7      struct student
8      {
9          int snum;
10         char* dept;
11         char name[12];
12     };
13     struct student hong = { 201800001, "컴퓨터정보공학과", "홍길동" };
14     struct student na = { 201800002 };
15     struct student bae = { 201800003 };
16
17     scanf("%s", na.name);
18
19     na.dept = "컴퓨터정보공학과";
20     bae.dept = "기계공학과";
21     memcpy(bae.name, "배상문", 7);
22     strcpy(bae.name, "배상문");
23     strcpy_s(bae.name, 7, "배상문");
24
25     printf("[%d, %s, %s]\n", hong.snum, hong.dept, hong.name);
26     printf("[%d, %s, %s]\n", na.snum, na.dept, na.name);
27     printf("[%d, %s, %s]\n", bae.snum, bae.dept, bae.name);
28
29     struct student one;
30     one = bae;
31     if (one.snum == bae.snum)
32         printf("학번이 %d으로 동일합니다.\n", one.snum);
33     if (one.snum == bae.snum && !strcmp(one.name, bae.name) && !strcmp(one.dept, bae.dept))
34         printf("내용이 같은 구조체입니다.\n");
35
36     return 0;
37 }
```

Microsoft Visual Studio 디버그 콘솔

나한국
[201800001, 컴퓨터정보공학과, 홍길동]
[201800002, 컴퓨터정보공학과, 나한국]
[201800003, 기계공학과, 배상문]
학번이 201800003으로 동일합니다.
내용이 같은 구조체입니다.

C:\Users\WPUIN\source\repos\Perfect\CWDe
이 창을 닫으려면 아무 키나 누르세요...

[위 사진은 구조체 멤버로 다른 변수를 사용한 실습예제 13-3이다.]

7행에서 구조체 **student**를 정의함.

변수 **name**은 **char**배열로 문자열 자체의 저장이 가능하므로 **scanf()**나 **memcpy()**, **strcpy()**의 사용도 가능

31행에서 **one**의 학번과 **bae**학번을 비교하는 문

구조체 자체로는 비교 연산 **one == bae**는 불가능

33행에서 구조체 변수 **one**에 변수 **bae**의 내용을 모든 비교하려면 각각의 멤버를 모두 비교
내용이 같으면 출력

공용체란? 동일한 저장 장소에 여러 자료형을 저장하는 방법으로, 공용체를 구성하는

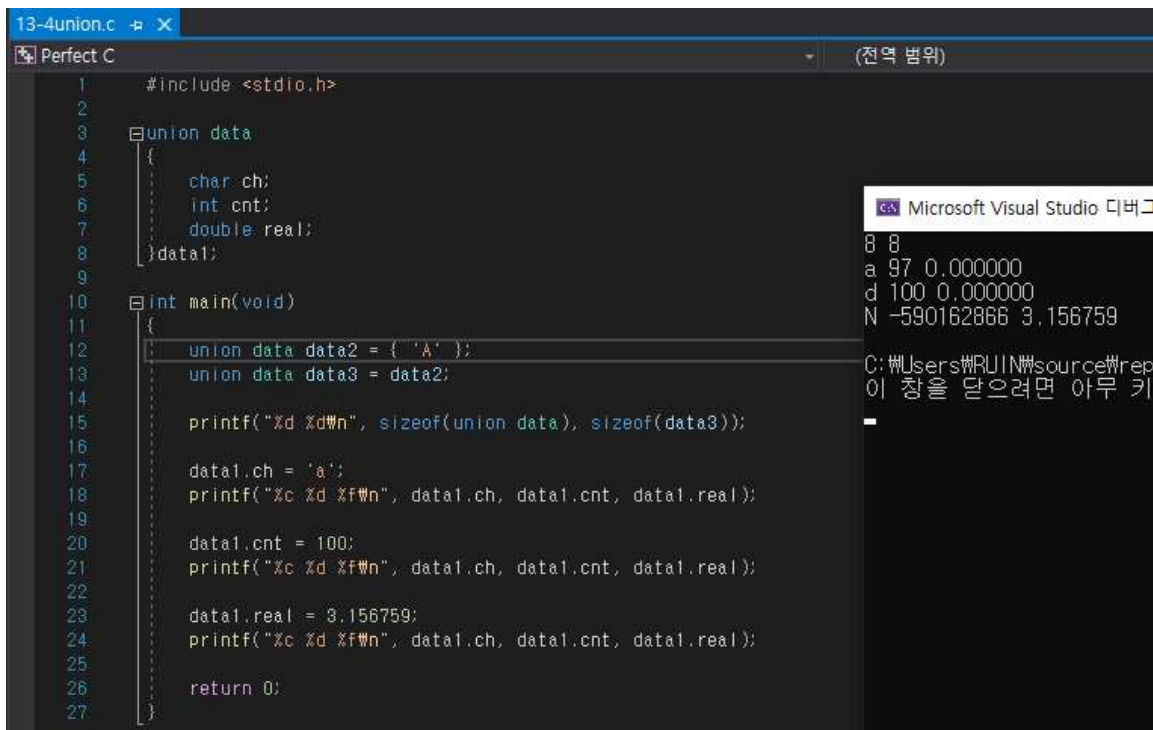
멤버에 한번에 한 종류만 저장하고 참조할 수 있다.

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

선언방법은 **union**을 **struct**로 사용하는 것을 제외하면 구조체 선언 방법과 동일하다.

[공용체의 특징]

- ①공용체 변수의 크기는 **멤버중 가장 큰 자료형의 크기**로 정해진다.
- ②공용체의 멤버는 모든 멤버가 동일한 저장 공간을 사용하므로 동시에 여러 멤버의 값을 동시에 저장하여 이용할 수 없으며, 마지막에 저장된 **단 하나의 멤버 자료값**만은 저장한다.
- ③공용체의 초기화값은 공용체 정의시 **처음 선언한 멤버의 초기값으로만 저장**이 가능하다.
- ④공용체 변수로 멤버를 접근하기 위해서는 구조체와 같이 **접근연산자**.를 사용한다.



```
1  #include <stdio.h>
2
3  union data
4  {
5      char ch;
6      int cnt;
7      double real;
8  }data1;
9
10 int main(void)
11 {
12     union data data2 = { 'A' };
13     union data data3 = data2;
14
15     printf("%d %d\n", sizeof(union data), sizeof(data3));
16
17     data1.ch = 'a';
18     printf("%c %d %f\n", data1.ch, data1.cnt, data1.real);
19
20     data1.cnt = 100;
21     printf("%c %d %f\n", data1.ch, data1.cnt, data1.real);
22
23     data1.real = 3.156759;
24     printf("%c %d %f\n", data1.ch, data1.cnt, data1.real);
25
26     return 0;
27 }
```

Microsoft Visual Studio 디버그 콘솔 출력:

```
8 8
a 97 0.000000
d 100 0.000000
N -590162866 3.156759
```

[위 사진은 공용체를 사용한 실습예제 13-4이다.]

3행에서 char, int, double 자료형 하나를 동시에 저장 할 수 있는 **8바이트 공간의 공용체 data**를 정의하는 문장. 동시에 **변수 data1**을 선언, 이 변수는 **전역변수**로 이 선언이 있는 위치 이후 모든 파일에서 사용가능

13행에서 변수 data3을 선언하면서 **초기값으로 같은 유형의 변수**를 대입.

18행에서 공용체 data1에서 **자료형 char**인 멤버 ch에 문자'a'를 저장, 이 이후로는 data1.ch만 의미가 있음.

모든 멤버를 출력해보나, **data1.ch**만 정확히 출력.

공용체 data1에서 **자료형 int**인 멤버 cnt에 정수 100을 저장, 이 이후로는 data1.cnt만 의미가 있음.

모든 멤버를 출력해보나, **data1.cnt**만 정확히 출력.

자료형 재정의

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

[자료형 재정의 typedef구문]

typedef 기존자료유형이름 **새로운자료형1**, **새로운자료형2**;

일반적으로 자료형을 재정의하는 이유는 프로그램의 시스템 간 **호환성**과 **편의성**을 위해 필요하다.

문장 **typedef**도 일반 변수와 같이 그 **사용범위를 제한한다**.

```
1 #include <stdio.h>
2
3 typedef unsigned int budget;
4
5 int main(void) {
6     budget year = 24500000;
7     typedef int profit;
8     profit month = 4600000;
9
10    printf("올 예산은 %d, 이달의 이익은 %d입니다.\n", year, month);
11
12    return 0;
13 }
14
15 void test(void) {
16     budget year = 24500000;
17
18 }
```

[위 사진은 자료형 재정의의 사용한 실습예제 13-5이다.]

7행에서 **profit**은 **int**와 같은 **자료형**, 이 위치가 함수 내부이므로 자료형 **profit**은 이 위치이후 **함수 main()내부**에서 사용가능.

8행에서 **profit**은 **int**와 같은 자료형으로 변수 **month**를 **profit**으로 선언하면서 초기값을 대입함.

16행에서 자료형 **budget**은 **함수 test()**에서도 사용가능.

구조체 **struct date**가 정의된 상태에서 **typedef**사용하여 구조체 **struct date**를 **date**로 **재정의** 할 수 있다.

typedef를 이용한 다른방법은 구조체 **정의자체**를 **typedef**와 **함께 처리하는 방법**이다.

ex) struct data{

int year;

int month;

int day; };

typedef struct date date; <- 자료유형 **date**는 **struct date**와 함께 **동일한 자료유형으로 이용**이 가능하다.

```

13-6typedefstruct.c
Perfect C (전역 범위)
1 #include <stdio.h>
2
3 struct date {
4     int year;
5     int month;
6     int day;
7 };
8 typedef struct date date;
9
10 int main(void) {
11
12     typedef struct{
13
14         char title[30];
15         char company[30];
16         char kinds[30];
17         date release;
18     } software;
19
20     software vs = { "비주얼 스튜디오 커뮤니티", "MS", "통합개발환경", {2018, 8, 29} };
21
22     printf("제품명: %s\n", vs.title);
23     printf("회사: %s\n", vs.company);
24     printf("종류: %s\n", vs.kinds);
25     printf("출시일: %d, %d, %d\n", vs.release.year, vs.release.month, vs.release.day);
26
27     return 0;
28 }

```

Microsoft Visual Studio 디버그 콘솔

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

C:\Users\RUN\source\repos\Perfect_C\...
이 창을 닫으려면 아무 키나 누르세요..

[위 사진은 구조체 자료형 재정의의 사용한 실습예제 13-6이다.]

3행에서 구조체 struct date 정의

8행에서 struct date형을 간단히 date 형으로 다시정의, 자료형 date는 이 구문 이후 파일 어디에서나 사용이 가능함.

18행에서 software는 변수 이름이 아니라 새로운 자료형

20행에서 software 자료형으로 vs를 선언하면서 초기값 저장.

```

Lab13-2_structcity.c
Perfect C (전역 범위)
1 #include <stdio.h>
2
3 int main(void) {
4     typedef struct movie {
5         char* title;
6         int attendance;
7     } movie;
8     movie assassination;
9
10    assassination.title = "암살";
11    assassination.attendance = 12700000;
12
13    printf("[%s] 관객수: %d\n", assassination.title, assassination.attendance);
14
15    return 0;
16 }

```

Microsoft Visual Studio 디버그 콘솔

[암살] 관객수: 12700000

[위 사진은 구조체 자료형 재정의의 사용한 Lab 13-2이다.]

영화의 제목과 관객수를 표현하는 구조체 struct movie는 멤버로 char *와 int로 구성된다.

구조체 struct movie의 멤버인 title은 영화제목을 attendance는 관객수를 저장함. 이 프로그램에서 구조체 struct movie의 자료형을 다시 movie로 정의하고, 변수 assassination에 하나의 영화정보를 저장한 후 다시 출력하는 문이다.

구조체와 공용체의 포인터와 배열

포인터는 각가의 자료형 저장 공간의 주소를 저장하듯이 **구조체 포인터**는 구조체의 **주소값을 저장**할 수 있는 변수이다.

구조체 포인터 변수의 선언은 **일반 포인터 변수선언과 동일**하다.

```
ex) struct lecture{
    char name[20];
    int type;
    int credit;
    int hours; };
typedef struct lecture lecture;
lecture *p;
lecture os = {"운영체제, 2, 3,3};
lecture *p = &os;          <- lecture 포인터 변수p에 &os를 저장한다.
```

구조체 포인터 멤버 접근연산자 ->는 p->name과 같이 사용한다.

연산식 p-> name은 포인터 p가 가리키는 구조체 변수의 멤버 name을 **접근하는 연산식**이다.

연산식 ***p.name**은 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 os멤버 강좌명의 첫 문자임, 다한 한글인 경우에는 실행오류 |
| *p->name | *(p->name)을 의미하며, 포인터 p이 가리키는 구조체의 멤버 name이 가리키는 변수로 이 경우는 구조체 포인터 p이 가리키는 구조체의 멤버 강좌명의 첫 문자임, 마찬가지로 한글인 경우에는 실행오류 |

연산자 **->**와 **.**은 **우선순위 1**이고 결합성은 **좌에서 우**이며, 연산자 *****은 **우선순위2**이고 결합성은 **우에서 좌**이다.


```

13-7strcutpointer.c
Perfect C (전역 범위)
1 #include <stdio.h>
2
3 struct lecture {
4     char name[20];
5     int type;
6     int credit;
7     int hours;
8 };
9 typedef struct lecture lecture;
10
11 char* head[] = { "강좌명", "강좌구분", "학점", "사수" };
12 char* lectype[] = { "교양", "일반선택", "전공필수", "전공선택" };
13
14 int main(void) {
15     lecture os = { "운영체제", 2, 3, 3 };
16     lecture c = { "C프로그래밍", 3, 3, 4 };
17     lecture* p = &os;
18
19     printf("구조체 크기: %d, 포인터 크기: %d\n\n", sizeof(os), sizeof(p));
20     printf("%10s %12s %6s %6s\n", head[0], head[1], head[2], head[3]);
21     printf("%12s %10s %5d %5d\n", p->name, lectype[p->type], p->credit, p->hours);
22
23     p = &c;
24     printf("%12s %10s %5d %5d\n", (*p).name, lectype[(*p).type], (*p).credit, (*p).hours);
25     printf("%12c %10s %5d %5d\n", *c.name, lectype[c.type], c.credit, c.hours);
26
27     return 0;
28 }

```

Microsoft Visual Studio 디버그 콘솔

```

구조체 크기: 32, 포인터 크기: 4
강좌명      강좌구분  학점   사수
운영체제    전공필수   3      3
C프로그래밍 전공선택   3      4

```

[위 사진은 구조체 포인터의 선언과 사용한 실습예제 13-7이다.]

3행에서 구조체 **struct lecture** 정의

9행에서 **struct lecture**를 자료형 **lecture**로 정의

15행에서 구조체 자료형 **lecture** 변수인 **os**를 선언하면서 초기화, 두 번째 멤버 초기값 2에서 2는 강좌구분 “**전공필수**”를 의미하고, 12행 배열 **lectype**의 **첨자인 2**

16행에서 구조체 자료형 **lecture** 변수인 **c**를 선언하면서 초기화, 두 번째 멤버 초기값 3에서 3은 강좌구분 “**전공선택**”을 의미하고, 12행 배열 **lectype**의 **첨자인 3**

17행에서 구조체 자료형 **lecture** 포인터 **p**를 선언하면서 **os**의 주소를 저장

19행에서 **os**와 **p**의 저장공간 크기를 출력

21행에서 구조체 포인터 **p**와 참조연산자 **->**를 이용하여 멤버를 참조하여 출력

23행에서 구조체 자료형 **lecture**포인터 **p**에 **c**의 주소로 변경, 이제 **p**는 **c**를 가리킴

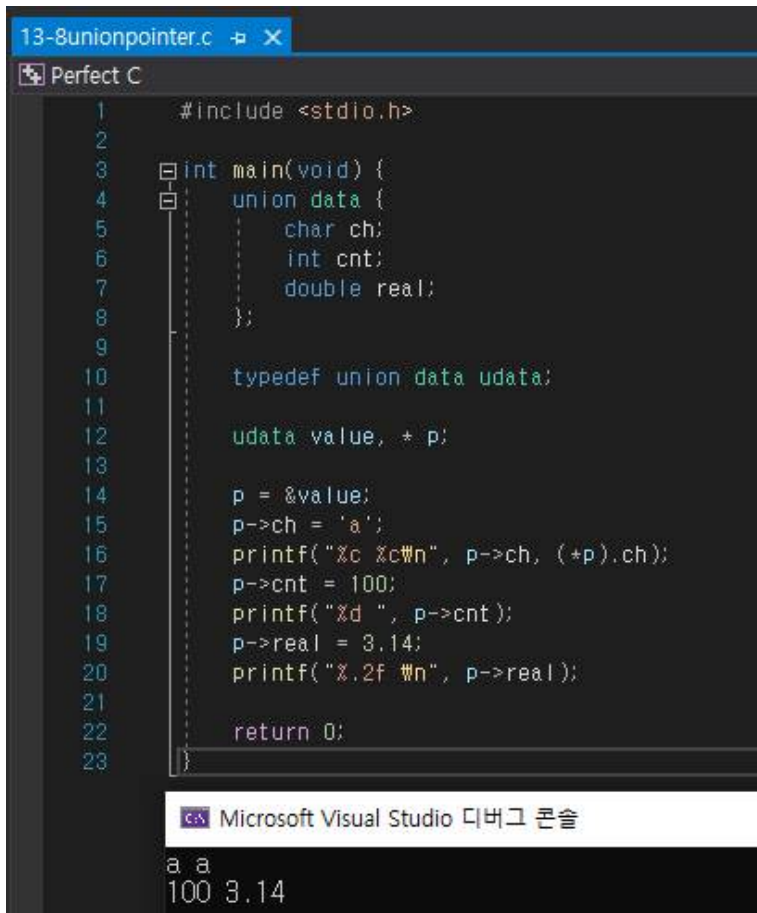
24행에서 구조체 포인터 **p**와 간접연산자 **(*p)**와 참조연산자 **.**를 이용하여 멤버를 참조하여 출력

25행구조체자체인 **c**와 참조연산자**.**를 이용하여 멤버를 참조하여 출력, 첫 번째 출력인 ***c.name**은 ***(c.name)**을 의미하므로 첫 글자인 **C**가 출력

[공용체 포인터]

공용체 변수도 포인터 변수 사용이 가능하며, **공용체 포인터** 변수로 멤버를 접근하려면 **접근연산자 ->**를 이용한다.

ex) union data{
 char ch;
 int cnt;
 double real;} value, *p; <- p는 union data 포인터 형으로 선언
p = &value; <- 포인터 p에 value의 주소값을 저장
p->ch = 'a'; <- value.ch = 'a';와 동일한 문장



```
1  #include <stdio.h>
2
3  int main(void) {
4      union data {
5          char ch;
6          int cnt;
7          double real;
8      };
9
10     typedef union data udata;
11
12     udata value, *p;
13
14     p = &value;
15     p->ch = 'a';
16     printf("%c %c\n", p->ch, (*p).ch);
17     p->cnt = 100;
18     printf("%d ", p->cnt);
19     p->real = 3.14;
20     printf("%.2f\n", p->real);
21
22     return 0;
23 }
```

Microsoft Visual Studio 디버그 콘솔

```
a a
100 3.14
```

[위 사진은 공용체 정의와 변수선언 및 사용한 실습예제 13-8이다.]

4행에서 char, int ,double자료형 하나를 동시에 저장할 수 있는 **공용체를 정의함**.

10행에서 union data를 다시 **자료형 udata로 정의**

14행에서 포인터 변수 p에 공용체 변수 **value의 주소**를 저장

16행에서 p가 가리키는 공용체 변수 value의 멤버 **ch**를 **p->ch**와 **(*p).ch**로 참조하여 출력

18행에서 p가 가리키는 공용체 변수 value의 멤버 **cnt**를 **p->cnt**로 참조하여 출력,

(*p).cnt도 가능

[구조체 배열 변수 선언]

다른 배열과 같이 동일한 구조체 변수가 여러개 필요하다면 **구조체 배열**을 선언하여 이용할 수 있다.

구조체 배열의 초기값 지정 구문에서는 **중괄호가 중첩**되게 나타난다.

외부 중괄호는 **배열초기화의 중괄호**이며, **내부 중괄호**는 배열원소인 **구조체 초기화**를 위한 **중괄호**이다.

```
ex) lecture c[] = { {"인간과사회", 0, 2, 2},  
                  {"경제학개론", 1, 3, 3},  
                  {"자료구조", 2, 3, 3} };
```

구조체 배열이름은 **구조체 포인터 변수에 대입이 가능**하며, 구조체 포인터 변수를 이용한 **배열원소 접근이 가능**함.

```
1 #include <stdio.h>
2
3 struct lecture {
4     char name[20];
5     int type;
6     int credit;
7     int hours;
8 };
9 typedef struct lecture lecture;
10
11 char* lectype[] = { "교양", "일반선택", "전공필수", "전공선택" };
12 char* head[] = { "강좌명", "강좌구분", "학점", "시수" };
13
14 int main(void) {
15     lecture course[] = { {"인간과 사회", 0, 2, 2},
16                         {"경제학개론", 1, 3, 3},
17                         {"자료구조", 2, 3, 3},
18                         {"모바일 프로그래밍", 2, 3, 4},
19                         {"고급 C프로그래밍", 3, 3, 4} };
20
21     int arysize = sizeof(course) / sizeof(course[0]);
22
23     printf("배열 크기: %d\n", arysize);
24     printf("%12s %12s %6s %6s\n", head[0], head[1], head[2], head[3]);
25     printf("-----\n");
26     for (int i = 0; i < arysize; i++)
27         printf("%16s %10s %5d %5d\n", course[i].name, lectype[course[i].type], course[i].credit, course[i].hours);
28
29     return 0;
30 }
```

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 |

C:\Users\WPUIN\source\repos\Perfect C\Debug\Perfect C.exe(프
이 창을 닫으려면 아무 키나 누르세요...

[위 사진은 구조체 배열을 선언한 후 출력처리한 실습예제 13-9이다.]

15행에서 구조체 struct의 배열을 선언하면서 바로 **초기값**을 대입, 배열크기는 지정하지않고, 초기값을 **지정한 원소 수가 5**

21행에서 배열크기를 계산하여 변수 **aryszie**에 저장

24행에서 배열크기와 제목 출력

26행에서 첨자 **i**는 **0**에서 **arysize-1**까지 실행

27행에서 구조체 struct의 모든 배열 원소를 각각 출력

```

1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <string.h>
4
5  int main(void) {
6      typedef struct movie {
7          char* title;
8          int attendance;
9          char director[20];
10     }movie;
11
12     movie box[] = {
13         { "명량", 1761300, "김한민" },
14         { "국제시장", 14257000, "윤제균" },
15         { "베테랑", 13383000 };
16
17     strcpy(box[2].director, "류승완");
18
19     printf("   제목   감독   관객수\n");
20     printf("=====");
21     for (int i = 0; i < 3; i++)
22         printf("[%8s] %6s %d\n",
23             box[i].title, box[i].director, box[i].attendance);
24
25     return 0;
26 }

```

Microsoft Visual Studio 디버그 콘솔

```

   제목   감독   관객수
=====
[   명량]   김한민 1761300
[국제시장]   윤제균 14257000
[   베테랑]   류승완 13383000

```

C:\Users\WUIN\source\repos\Perfect C
이 창을 닫으려면 아무 키나 누르세요.

[위 사진은 구조체 배열을 선언한 후 출력처리한 Lab 13-3이다.]

영화의 제목과 감독, 관객수를 표현하는 **구조체** struct movie는 멤버로 char *title, int attendance, char director[20]로 구성된다.

이 프로그램에서 **구조체 moive**의 **배열을 선언**하고 **초기화**로 영화 세 편의 정보를 저장한다.

세 번째 영화의 감독을 “류승완”을 저장하고 모든 영화의 정보를 다시 출력 해보도록 하는 프로그램이다.

구조체는 아래 구조와 같아야함.

```

구조체 struct moive{
char *title;
int attendance;
char director[20];
};

```

- 15행에서 box의 배열을 선언하면서 바로 초기값을 대입
- 17행에서 류승완을 strcpy함수로 복사함.
- 23행에서 구조체 struct의 모든 배열 원소를 각각 출력함.

감사합니다.