

# C 포트폴리오 과제

20164113 황상건

**저는 이 포트폴리오를 세가지 방식으로  
구성해 보았습니다.**

첫 번째는 직접 자필로 작성한 필기들을  
사진으로 찍은 것이며,

두 번째는 비주얼 스튜디오를 활용하여  
집에서 작성한 소스이고,

세 번째는 발표 준비를 하면서 복습한 자료 입니다.

책에 직접 필기한 내용들 입니다.

### TIP 메모리 주소가 왜 필요하지요?

이 메모리 주소값을 이용하면 보다 편리하고 융통성 있는 프로그램이 가능하다. 그러나 메모리 주소를 잘못 다루면 시스템에 심각한 문제를 일으킬 수 있다. 그리고 메모리 주소를 처음 학습하는 초보자에게 좀 어렵다는 단점이 있다. 그러나 걱정하지 말자. 이 단원을 차근차근 알아가면 메모리 주소도 어렵지 않다는 것을 이해할 것이다.

- 주소 정보를 이용하여 주소가 가리키는 변수의 값을 참조할 수 있다.
- 주소 정보의 이전 또는 이후의 이웃한 저장 공간의 값도 쉽게 참조할 수 있다.

### 주소연산자 &

주소는 변수이름과 같이 저장장소를 참조하는 하나의 방법이다. 지금까지 함수 scanf()를 사용하면서 입력 자료의 값을 저장하기 위해 인자를 '&변수이름'으로 사용하였다. 바로 &(ampersand)가 피연산자인 변수의 메모리 주소를 반환하는 주소연산자이다.

✓ 즉 함수 scanf()에서 입력값을 저장하는 변수의 주소값이 인자의 자료형이다. — ?

- 그러므로 함수 scanf()에서 일반 변수 앞에는 주소연산자 &를 사용해야 한다.

```
int input;  
scanf("%d", &input);
```

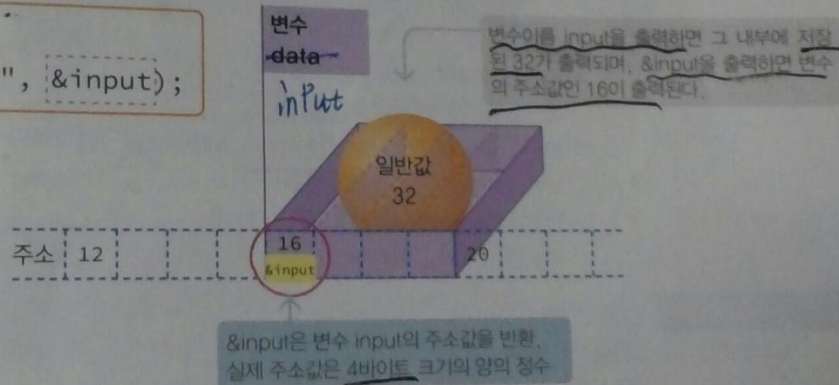


그림 8-2 주소연산자 & 이해

변수의 주소값은 형식제어문자 %u 또는 %d로 직접 출력할 수 있다. 그러나 최근 비주얼 스튜디오에서는 경고가 발생하니 주소값을 int 또는 unsigned로 변환하여 출력한다. 만일 16진수로 출력하려면 형식제어문자 %p를 사용한다. 주소 연산자 &는 다음 사용에 주의가 필요하다.

- & 연산자는 '&변수'와 같이 피연산자 앞에 위치하는 전위연산자로 변수에만 사용할 수 있다.
- '&32'와 '&(3+4)'와 같이 상수나 표현식에는 사용할 수 없다.

## 포인터 변수 개념과 선언

### 메모리 주소 저장 변수인 포인터 변수

우리가 친구의 주소를 스마트폰의 주소록에 저장하듯이 어느 변수의 주소도 포인터 변수에 저장할 수 있다. 어느 변수의 주소도 일반 정수값이라고 볼 수 있으나 일반 변수에 저장하면 주소값이라는 의미가 없어지므로 변수의 주소값은 반드시 포인터 변수에 저장해야 한다. 포인터 변수는 주소 연산자를 이용한 연산식 &data의 결과값은 바로 포인터 변수(pointer variable)에 저장하여 사용할 수 있다.



그림 8-3 메모리 주소를 저장하는 포인터 변수

### 포인터 변수 선언

포인터 변수는 일반 변수와 선언 방법이 다르다. 그렇다고 변수 선언이 복잡한 것은 아니고 포인터 변수 선언에서 자료형과 포인터 변수 이름 사이에 연산자 \*(asterisk)를 삽입한다. 즉 다음 변수 선언에서 `ptrint`, `ptrshort`, `ptrchar`, `ptrdouble`은 모두 포인터 변수이며 간단히 포인터라고도 부른다. 예로 `'int *ptrint'` 선언은 'int 포인터 ptrint'라고 읽는다.

변수 { 일반 변수  
 포인터 변수



## 실습예제 8-3

nullpointer.c

```

01 // file: nullpointer.c
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int *ptr1, *ptr2, data = 10;
07     ptr1 = NULL;
08     printf("%p\n", ptr1);
09     //printf("%p\n", ptr2);
10     printf("%d\n", data);
11     return 0;
12 }

```

data = 10;

초기값  $\times \rightarrow$  시작시간

```

06 int 형 포인터 변수 ptr1과 ptr2를 선언하면서 int 형 변수 data 선언하여 10 저장
07 int 형 포인터 변수 ptr1에 주소값 0인 NULL을 저장
09 int 형 포인터 변수 ptr1에 저장된 주소값인 0을 출력
10 int 형 포인터 변수 ptr2에 저장된 주소값을 출력하려 하나 저장된 것이 없어 컴파일오류 발생
11 int 형 변수 data에 저장된 값인 10을 출력

```

△**신용평가**  
신용평가

간접연산자 \*

간접연산자  
포인터를 사용하는 이유를 알아보면, 포인터는 변수를 참조할 수 있는 또 다른 방법을 제공한다. 즉 포인터 변수가 갖는 주소로 그 주소의 원래 변수를 참조할 수 있다. (포인터 변수가 리키고 있는 변수를 참조하려면 간접연산자 (indirection operator) \*를 사용한다.)

```
int data = 40;
```

```
int *p = &data;
```

\*  $p = 50\%$  - 한 번이 절반

double d = 4.3;

double \*p\_d = &d;

$$* p d = 5.7;$$

```
int data1 = 100, data2;
```

```
int *p;
```

```
printf("간접참조 출력: %d \n", *pprint);
```

```
*ptring = 200;
```

rint);  
rintel 2020  
Rings  
Circuit  
+ 1000  
1000

간접연산자를 이용한 \*printf는 포인터 printf가 가리키고 있는 변수 자체를 의미한다. 즉 포인터 printf가 가리키는 변수가 data라면 \*printf는 변수 data를 의미한다. 그러므로 이제 \*printf를 사용하면 data 변수 저장값을 참조할 수 있다. 이제 변수 data로 가능한 작업은 \*printf로도 가능하다. 문장 \*printf = 200;으로 변수 data의 저장값을 200으로 수정할 수 있다.

(변수 data 자체를 사용해 자신을 참조하는 방식을 직접참조(direct access)라 한다면, \*printf를 이용해서 변수 data를 참조하는 방식을 간접참조(indirect access)라 한다.

```
int data = 100;
int *printf = &data;
char *ptrchar = &ch;
printf("간접참조 출력: %d %c\n", *printf, *ptrchar);
*printf = 200;
printf("직접참조 출력: %d %c\n", data, ch);
```

*16*  
포인터 printf는 데이터의 주소값으로 참조  
포인터 ptrchar는 char의 주소값으로 참조  
포인터 printf는 200으로 데이터의 주소값을 변경  
ptrch = &ch;

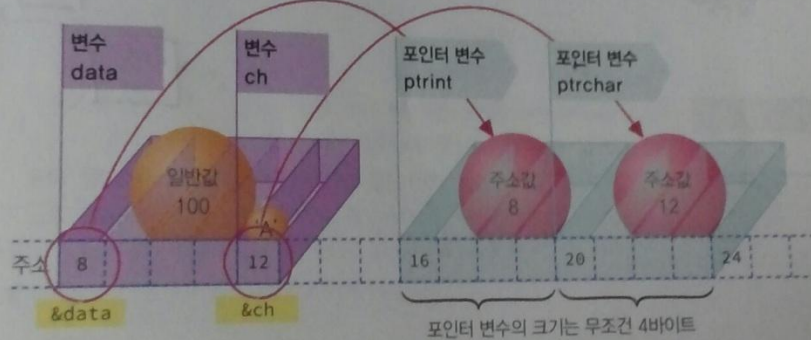


그림 8-9 포인터 변수와 주소값이 저장된 변수 사이의 관계

포인터 ≠ 간접연산자  
기호만 같음

TIP 주소 연산자 &와 간접 연산자 \*

주소 연산자 &와 간접 연산자 \*, 모두 전위 연산자로 서로 반대 역할을 한다. 즉 주소 연산 \*변수는 변수의 주소값이 결과값이며, 간접 연산 \*포인터변수는 포인터 변수가 가리키는 변수 자체가 결과값이다.

```
int n = 100;
int *p = &n;
n = *p + 1;
*p = *p + 1;
&n = 3;
```

*간접 연산*  
// 이제 \*p와 n은 같은 변수  
// n = n + 1;과 같음  
// \*p는 어느 위치에도 사용 가능  
// &n은 l-value로는 사용 불가하므로 오류 발생

\*포인터변수는 l-value와 r-value로 모두 사용이 가능하나, 주소값인 &변수는 r-value로만 사용이 가능하다.

\*포인터변수와 같이 간접연산자는 포인터 변수에만 사용이 가능하나, 주소 연산자는 &변수와 같이 모든 변수에 사용이 가능하다.



03

다음 프로그램의 출력을 기술하시오.

```
double x = 5.29, *y;
double *p = &x;
y = p;
*y = 3.89;
printf("%f\n", x);
```

원래 쓰려고 했

5.29

3.89

x  
5.29

\*p = 주소

### 포인터 변수의 연산

\*y = 3.89 주소 연산

간접 연산자

포인터 변수는 간단한 더하기와 뺄셈 연산으로 이웃한 변수의 주소 연산을 수행할 수 있다. 포인터의 연산은 절대적인 주소의 계산이 아니며, 포인터가 가리키는 변수 크기에 비례한 연산이다. 즉 포인터에 저장된 주소값의 연산으로 이웃한 이전 또는 이후의 다른 변수를 참조할 수 있다.

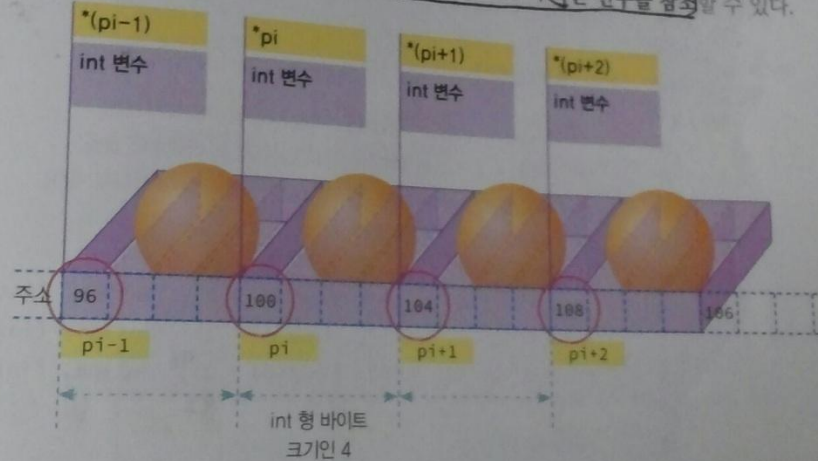


그림 8-10 주소의 연산을 이용한 이웃한 변수의 참조

int형권값이 4바이트단위만큼

위 그림과 같이 int형 포인터 pi에 저장된 주소값이 100이라고 가정하자. 그렇다면 (pi+1)은 101이 아니라 주소값 104이다. 즉 (pi+1)은 pi가 가리키는 다음 int형의 주소를 의미한다. 그러므로 (pi+1)은 int형의 바이트 크기인 4만큼 증가한 주소값 104가 되는 것이다. 또한 (pi-1)은 4만큼 감소한 96이 된다. 마찬가지로 double형 포인터 pd의 값이 100이라면, (pd+1)은 108이며, (pd-1)은 92가 된다. 더하기와 빼기 연산에는 포인터 변수가 피연산자로 참여할 수 있다. 그러나 곱하기와 나누기에는 포인터 변수가 피연산자로 참여할 수 없다. 즉 포인터 변수의 나누기와 곱하기 연산은 문법오류를 발생시킨다.

크기 4바이트  
주소값은 4바이트

double형 권값이 8바이트 단위만큼

```
*pi = i + 2; // i = i + 2;
**dpi = *pi + 2; // i = i + 2;
```

그림 8-16 다중포인트의 이용

## 실습예제 8-8

## multipointer.c

실습예제 8

- 줄거리 -

$i = \text{변수 } i \text{ 의}$   
저장값

$\&i = \text{변수 } i \text{ 의}$   
주소를

$p_i = \text{변수 } i \text{ 의}$   
주소값

$*p_i = \text{변수 } i \text{ 의}$   
저장값

$\&p_i = \text{변수 } p_i \text{ 의}$   
주소값

$d p_i = \text{변수 } p_i \text{ 의}$   
주소값

$*d p_i = \text{변수 } p_i \text{ 의}$   
저장값  
(즉, 변수  $i$  의  
주소)

$**d p_i = \text{변수 } i \text{ 의}$   
( $= *p_i$ ) 저장값

$dP_i = \frac{d}{dt} P_i$

$P_i$ 의 변화율

$\rightarrow dP_i$ 를 구할 때  $P_i$ 의 변화율을 구한다.

\*  $K = dP_i/dt$  즉  $P_i$ 의 변화율

|다시 생각하기

```

01 // file: multipointer.c
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int i = 20;
07     int *pi = &i; //포인터 선언
08     int **dpi = &pi; //이중 포인터 선언
09
10     printf("%p %p %p\n", &i, pi, *dpi);
11     (z:)
12     *pi = i + 2; // i = i + 2;
13     printf("%d %d %d\n", i, *pi, **dpi);
14     (i) (z:)
15     **dpi = *pi + 2; // i = i + 2;
16     printf("%d %d %d\n", i, *pi, **dpi);
17
18     return 0;
19 }

```

설명  
 $Qdp_i = \frac{1}{2} Qdp_i$   
 전압

### 실행결과

$$\begin{aligned} x \oplus p_i &= 0_i \\ (x \oplus p_i) &= 0_i \\ x \oplus p_i &= p_i = 0_i \end{aligned}$$



## 간접 연산자와 증감 연산자 활용

간접 연산자 \*는 증감 연산자 ++, --와 함께 사용하는 경우가 흔하다. 간접 연산자 \*는 전위 연산자로 연산자 우선순위가 2위이며, 증감 연산자 ++, --는 전위이면 2위이고, 후위이면 1위이다. 다음은 단항 연산자 \*, &, ++, --의 우선순위를 정리한 표이다.

표 8-3 연산자 우선순위

우선순위	단항 연산자	설명	결합성(계산방향)
1	a++ a--	후위 증가, 후위 감소	-> (좌에서 우로)
2	++a --a & *	전위 증가, 전위 감소 주소 간접 또는 역참조	<- (우에서 좌로)

포인터 변수 p에서 다음과 같이 정리할 수 있으며, 그 의미를 정리하면 다음 표와 같다.

\*p++는 \*(p++)으로 (\*p)++와 다르다.

- ++\*p와 ++(\*p)는 같다.
- \*++p는 (\*(++p))는 같다.

$$*(p++) \neq (*p)++$$

$$*(p++) = *(p)$$

\*p++ = \*(p++)  
\*p++의 결과  
30  
\*(p+1) = 50;  
int data = 30;  
int \*p = &data

표 8-4 증가연산자 ++와 간접연산자 \*의 사용 사례

연산식	결과값	연산 후 *p의 값	연산 후 p 증가
*p++	*(p++)	*p: p의 간접참조 값	변동 없음 p+1: p 다음 주소
+++p	*(++p)	*(p+1): p 다음 주소 (p+1) 간접참조 값	변동 없음 p+1: p 다음 주소
(*p)++		*p: p의 간접참조 값	*p가 1 증가 p: 없음
++*p	++(*p)	*p + 1: p의 간접참조 값에 1 증가	*p가 1 증가 p: 없음

주위는 ++이 다음에 적음  
전위는 ++이 바로 적용

## variousop.c

```

01 // file: variousop.c
02 #include <stdio.h>
03
04 int main(void)
05 {
06     int i;
07     int *pi = &i; //포인터 선언
08     int **dpi = &pi; //이중포인터 선언
09     i = 5;
10     *pi = 5; // *pi = *pi + 1과 같음
11     *pi += 1;
12     printf("%d\n", i);

```

16 \*p0는 변수 d와 같으므로 4.4 출력

실행결과 20  
4.400000

간접변수

ex) `const int *p1 = &i;`  
`*p1 = 20` → 예외

`const int *p1 = &i;`  
`int const *p1 = &i;` ] \*p1의 값(i)을 변경할 수 없음 (자료값 변경 불가)

`int *const p1 = &i;` - p1에 저장된 주소값을 (주소값 변경 불가) 변경할 수 없음

ex) `int *const p1 = &i;`  
`→ p1 = &j` → 예외  
 포인터변수

PTER 08 포인터 기초

출간일

01 포인터 실수의 종류 두 가지를 설명하시오.

02 다음 소스의 오류 원인을 찾아보고 바르게 수정하시오.

`const int`  
`int *const`  
`char c = 'A';`  
`char *const p;` p에 저장된 주소값 변경 불가 → 변수를 저장할 때 초기값 지정했으므로 &c로 변경하면 됨  
`p = &c;`  
`*p = 'B'` 등으로 수정

03 다음 소스의 오류 원인을 찾아보고 바르게 수정하시오.

`float f = 3.14f;`  
`float const * p;` (\*p)에 저장된 값 수정 불가 / 초기값 없음  
`p = &f;` → 포인터변수 p가 주소값을 참조 - f의 주소  
`*p = 5.62;` → 예외  
`p = &e` 등으로 수정

포인터는 주소값만 저장

비주얼 스튜디오 프로그램을 활용해  
복습한 자료들입니다.

## 2장-프로그래밍 기초

### C프로그래밍 2주차

크기 순서

솔루션(위치) > 프로젝트 > 소스

(작업공간)

코딩 시작 전

1.비주얼 스튜디오 켜기

2.파일-새로 만들기-프로젝트 선택

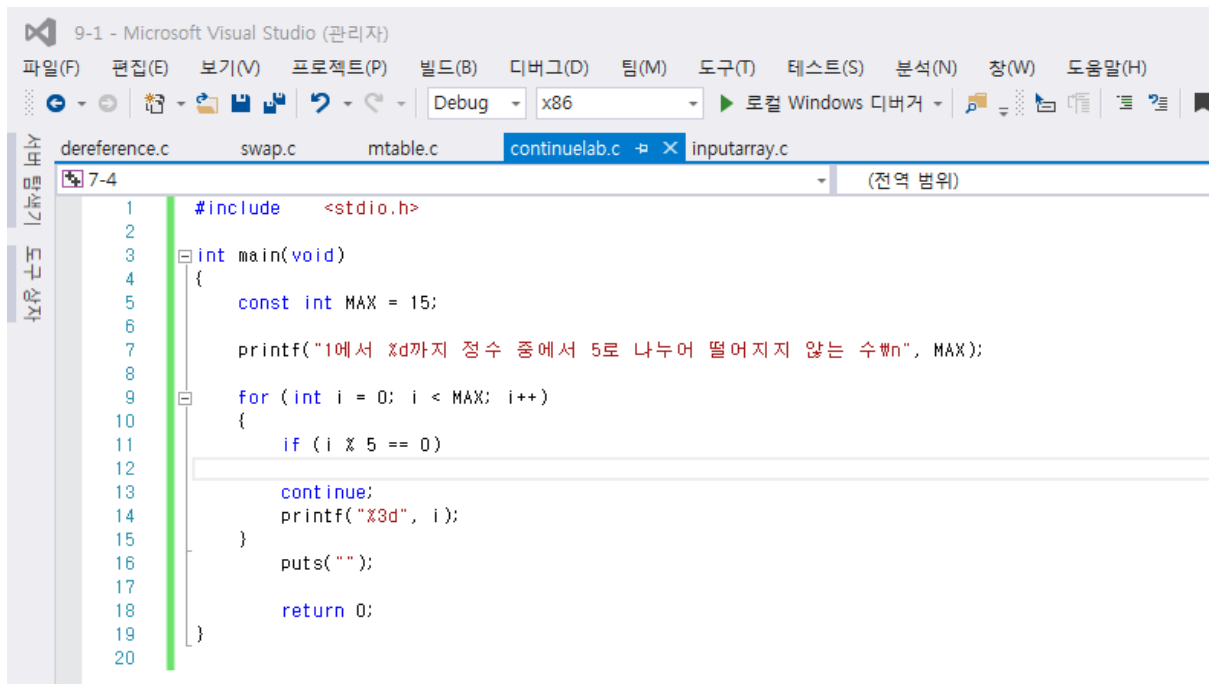
3.visual c++ 선택

4.win 32 콘솔 응용 프로그램 선택

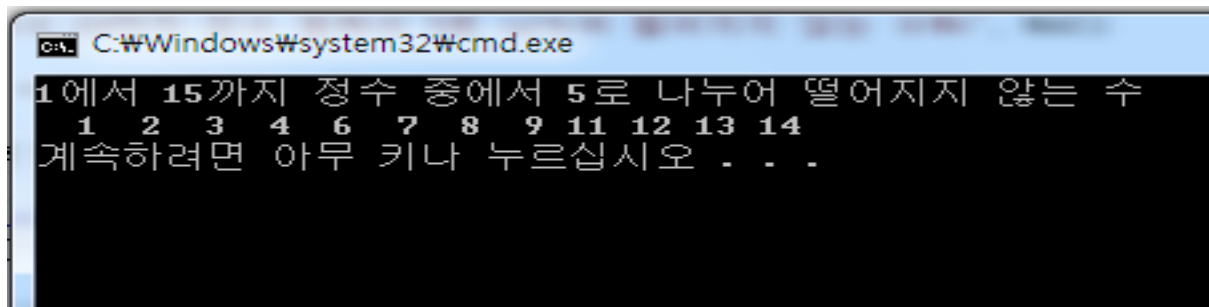


## 7장-반복 복습

7-1.반복문 for과 연산자를 활용해 5로 나누어 떨어지지 않는 수를 출력하기

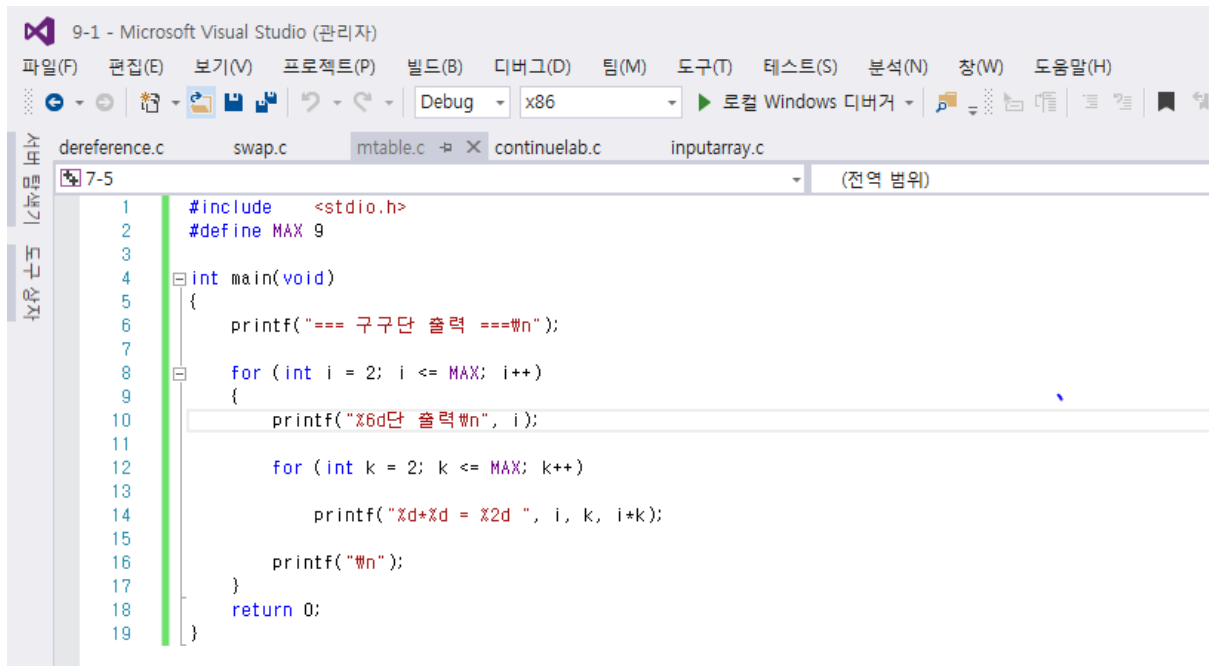


```
9-1 - Microsoft Visual Studio (관리자)
파일(F)  편집(E)  보기(V)  프로젝트(P)  빌드(B)  디버그(D)  팀(M)  도구(T)  테스트(S)  분석(N)  창(W)  도움말(H)
Debug x86 로컬 Windows 디버거
dereference.c swap.c mtable.c continuelab.c inputarray.c
7-4 (전역 범위)
1 #include <stdio.h>
2
3 int main(void)
4 {
5     const int MAX = 15;
6
7     printf("1에서 %d까지 정수 중에서 5로 나누어 떨어지지 않는 수\n", MAX);
8
9     for (int i = 0; i < MAX; i++)
10    {
11        if (i % 5 == 0)
12
13            continue;
14        printf("%3d", i);
15    }
16    puts("");
17
18    return 0;
19
20
```



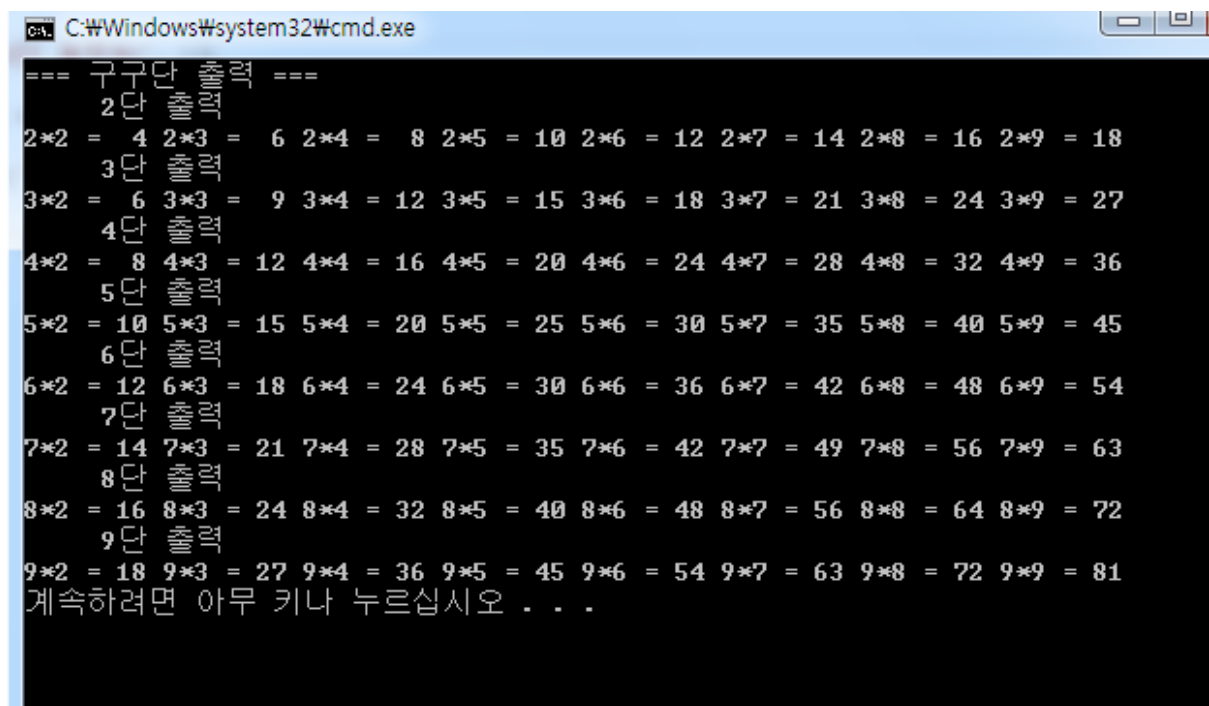
```
C:\Windows\system32\cmd.exe
1에서 15까지 정수 중에서 5로 나누어 떨어지지 않는 수
 1  2  3  4  6  7  8  9 11 12 13 14
계속하려면 아무 키나 누르십시오 . . .
```

## 7-2. 중첩된 for문을 사용해 구구단을 출력하기



The screenshot shows the Microsoft Visual Studio IDE with a C program open in the editor. The program is named '7-5' and is located in the file 'dereference.c'. The code defines a constant 'MAX' as 9 and implements a 'main' function that prints multiplication tables from 2 to 9. It uses two nested 'for' loops: the outer loop iterates over 'i' (2 to 9) and the inner loop iterates over 'k' (2 to 9). The output format is '%d단 출력\n' for the outer loop and '%d\*%d = %2d ' for the inner loop. The program ends with 'return 0;'.

```
1  #include <stdio.h>
2  #define MAX 9
3
4  int main(void)
5  {
6      printf("=== 구구단 출력 ===\n");
7
8      for (int i = 2; i <= MAX; i++)
9      {
10         printf("%d단 출력\n", i);
11
12         for (int k = 2; k <= MAX; k++)
13         {
14             printf("%d*%d = %2d ", i, k, i*k);
15
16             printf("\n");
17         }
18     }
19     return 0;
20 }
```

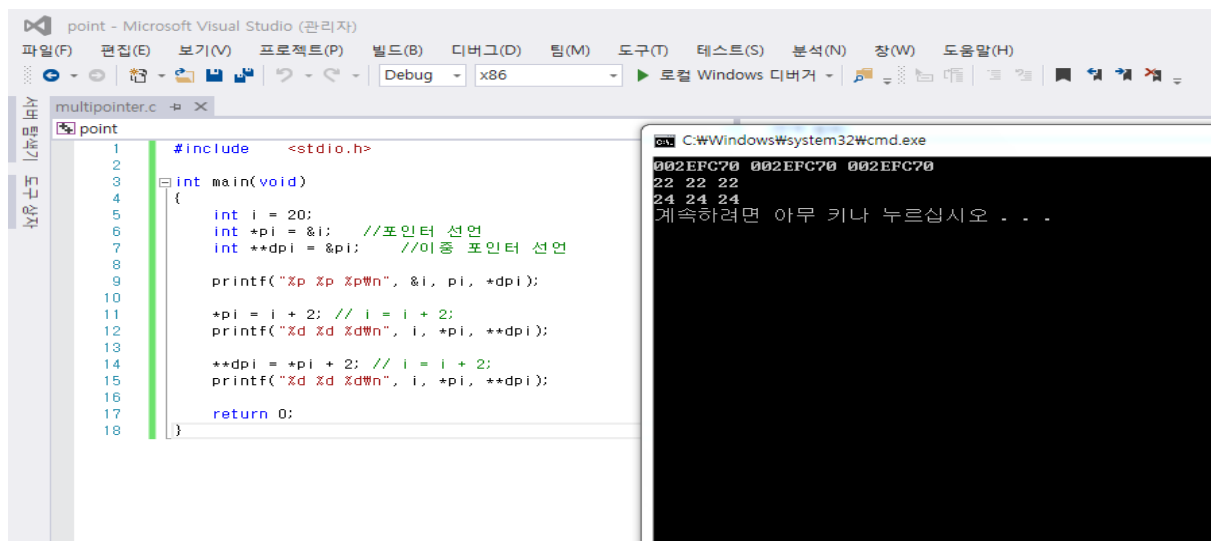


The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The output of the program is displayed, showing multiplication tables from 2 to 9. The output is formatted as follows:

```
=== 구구단 출력 ===
2단 출력
2*2 = 4 2*3 = 6 2*4 = 8 2*5 = 10 2*6 = 12 2*7 = 14 2*8 = 16 2*9 = 18
3단 출력
3*2 = 6 3*3 = 9 3*4 = 12 3*5 = 15 3*6 = 18 3*7 = 21 3*8 = 24 3*9 = 27
4단 출력
4*2 = 8 4*3 = 12 4*4 = 16 4*5 = 20 4*6 = 24 4*7 = 28 4*8 = 32 4*9 = 36
5단 출력
5*2 = 10 5*3 = 15 5*4 = 20 5*5 = 25 5*6 = 30 5*7 = 35 5*8 = 40 5*9 = 45
6단 출력
6*2 = 12 6*3 = 18 6*4 = 24 6*5 = 30 6*6 = 36 6*7 = 42 6*8 = 48 6*9 = 54
7단 출력
7*2 = 14 7*3 = 21 7*4 = 28 7*5 = 35 7*6 = 42 7*7 = 49 7*8 = 56 7*9 = 63
8단 출력
8*2 = 16 8*3 = 24 8*4 = 32 8*5 = 40 8*6 = 48 8*7 = 56 8*8 = 64 8*9 = 72
9단 출력
9*2 = 18 9*3 = 27 9*4 = 36 9*5 = 45 9*6 = 54 9*7 = 63 9*8 = 72 9*9 = 81
계속하려면 아무 키나 누르십시오 . . .
```

## 8 장-포인터 복습

### 1. 이중 포인터 사용 및 간접 참조



```
#include <stdio.h>

int main(void)
{
    int i = 20;
    int *pi = &i; //포인터 선언
    int **dpi = &pi; //이중 포인터 선언

    printf("%p %p %p\n", &i, pi, *dpi);

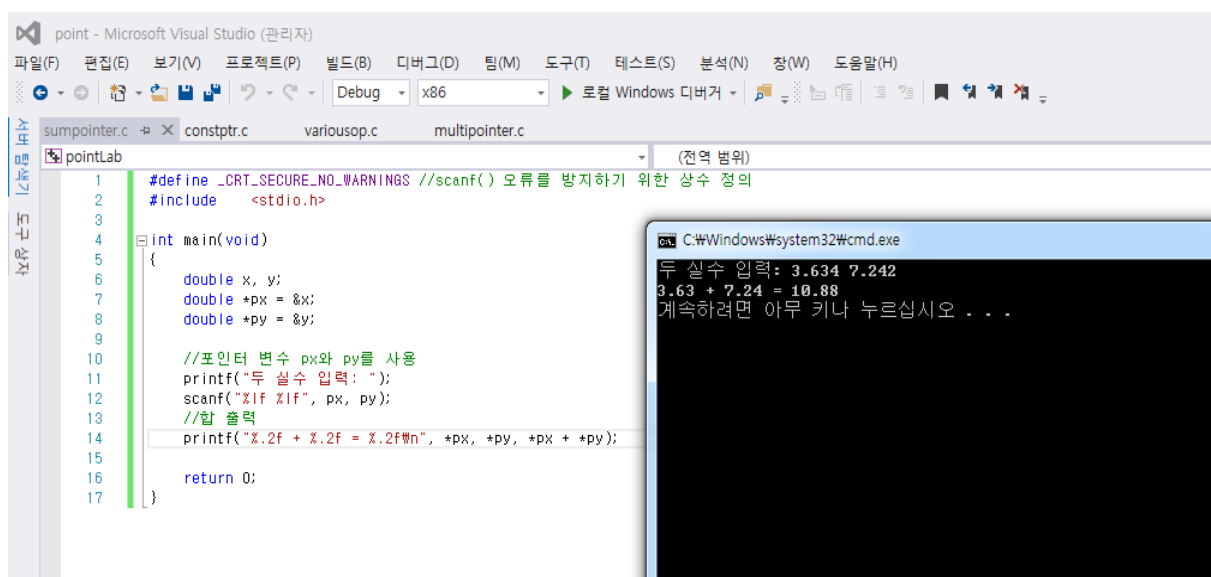
    *pi = i + 2; // i = i + 2;
    printf("%d %d %d\n", i, *pi, **dpi);

    **dpi = *pi + 2; // i = i + 2;
    printf("%d %d %d\n", i, *pi, **dpi);

    return 0;
}
```

002EFC70 002EFC70 002EFC70  
22 22 22  
24 24 24  
계속하려면 아무 키나 누르십시오 . . .

### 2. 포인터 연산



```
#define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
#include <stdio.h>

int main(void)
{
    double x, y;
    double *px = &x;
    double *py = &y;

    //포인터 변수 px와 py를 사용
    printf("두 실수 입력: ");
    scanf("%lf %lf", px, py);
    //합 출력
    printf("%.2f + %.2f = %.2f\n", *px, *py, *px + *py);

    return 0;
}
```

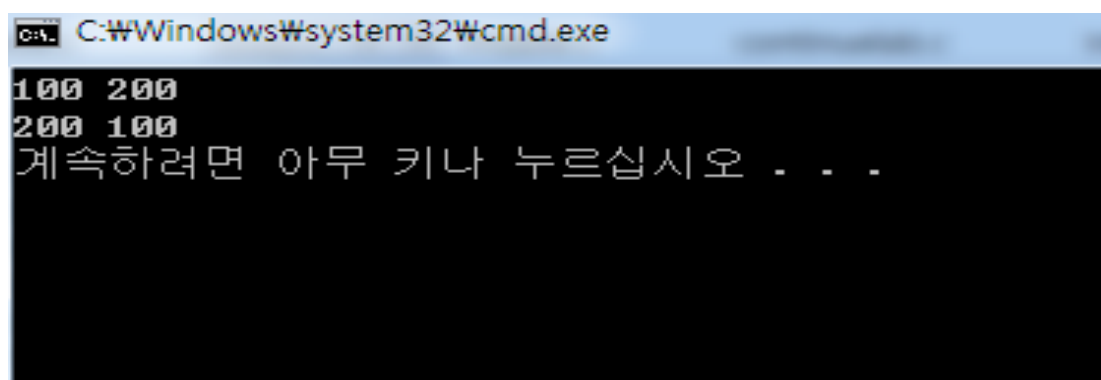
두 실수 입력: 3.634 7.242  
3.63 + 7.24 = 10.88  
계속하려면 아무 키나 누르십시오 . . .



### 3. 포인터를 이용해 변수의 값을 서로 교환



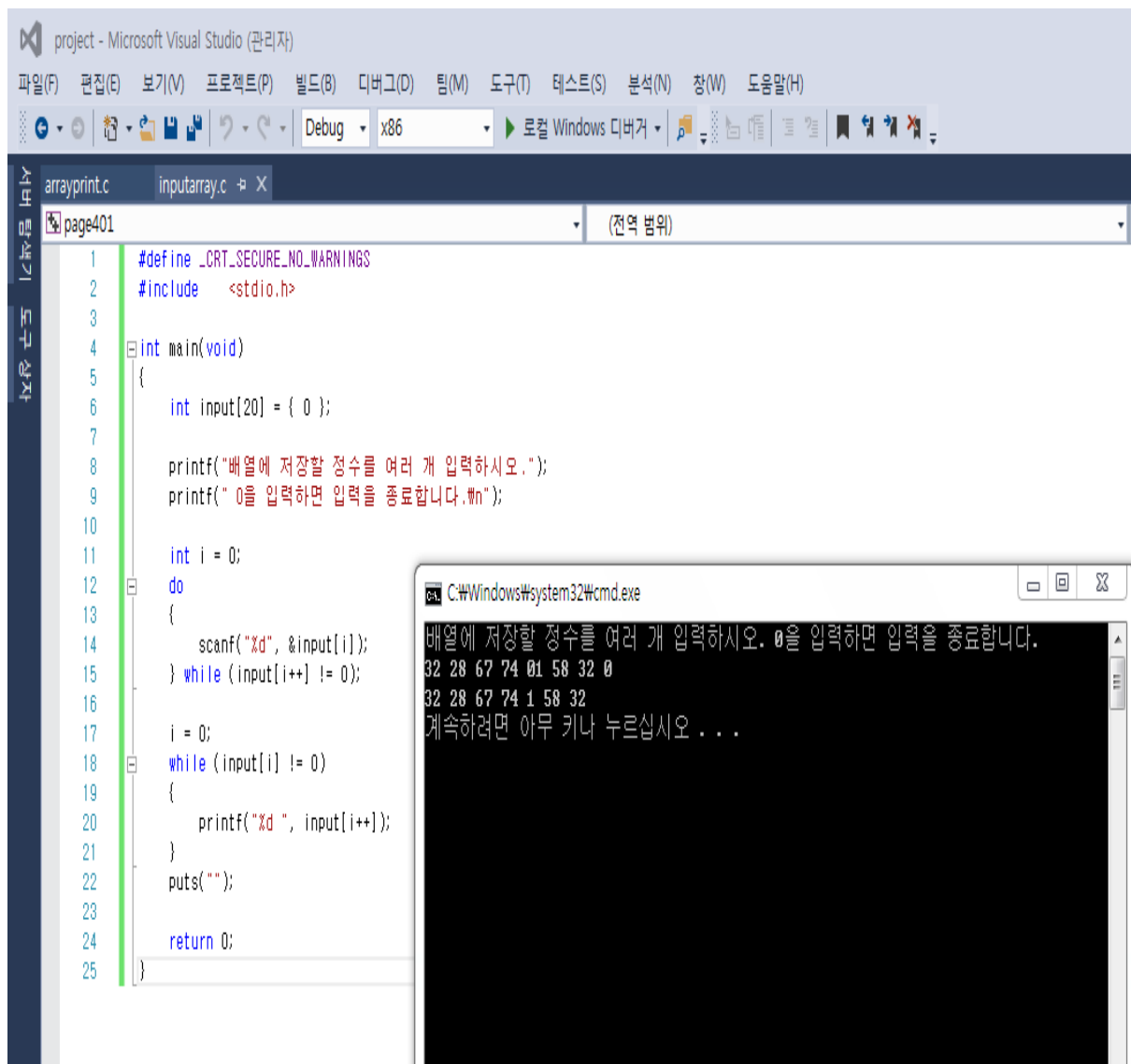
```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int m = 100, n = 200, dummy;
6      printf("%d %d\n", m, n);
7
8      int *p = &m;
9      dummy = *p;
10
11     *p = n;
12     p = &n;
13     *p = dummy;
14
15     printf("%d %d\n", m, n);
16
17     return 0;
18 }
```



```
C:\Windows\system32\cmd.exe
100 200
200 100
계속하려면 아무 키나 누르십시오 . . .
```

## 9 장-배열 복습

### 배열과 do while 문 활용



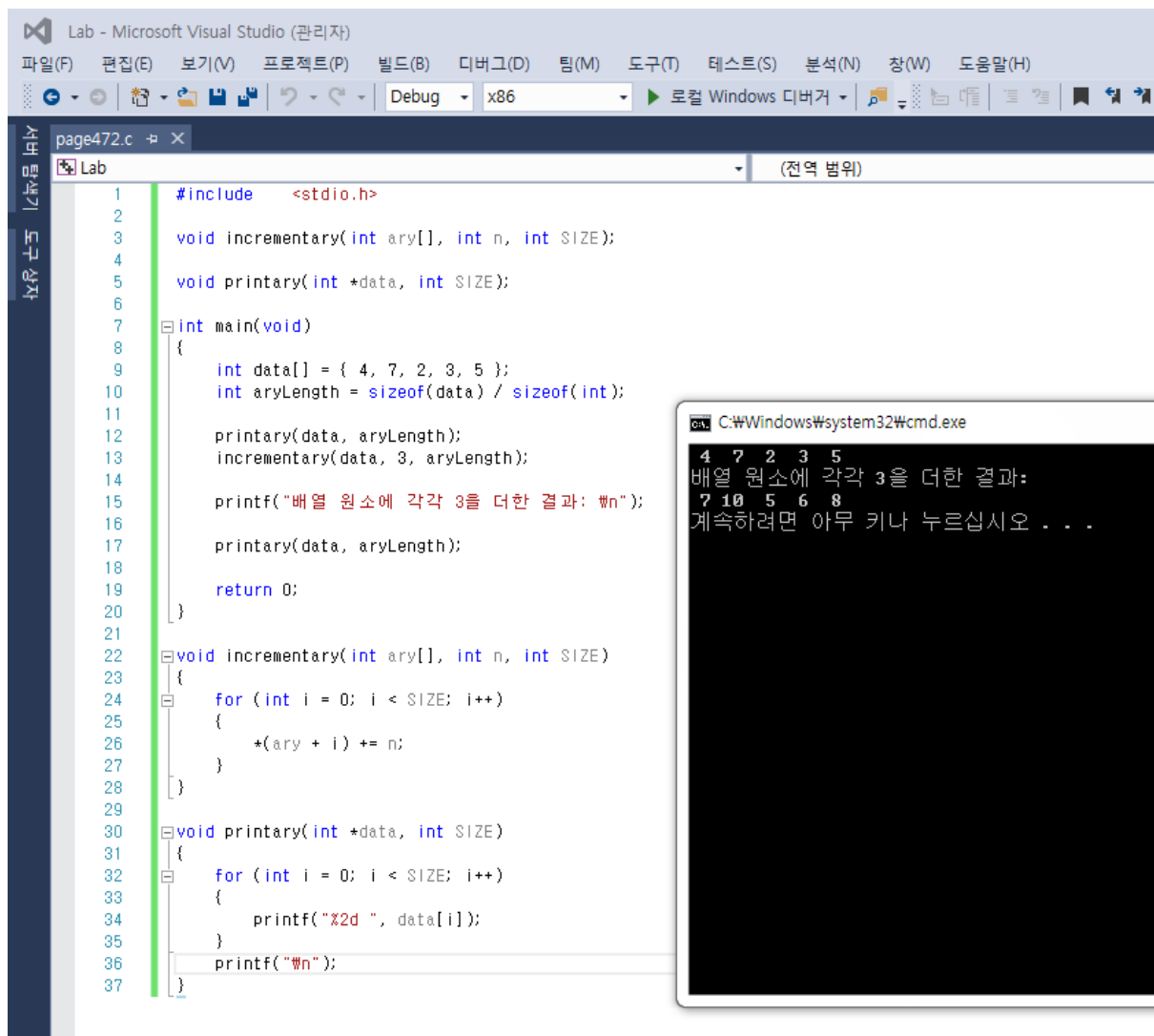
The screenshot shows the Microsoft Visual Studio IDE with a C program open in the editor. The program is named 'arrayprint.c' and is located in the 'project' folder. The code uses the 'Debug' configuration and 'x86' architecture. The program's logic is as follows:

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3
4  int main(void)
5  {
6      int input[20] = { 0 };
7
8      printf("배열에 저장할 정수를 여러 개 입력하시오.");
9      printf(" 0을 입력하면 입력을 종료합니다.\n");
10
11     int i = 0;
12     do
13     {
14         scanf("%d", &input[i]);
15     } while (input[i++] != 0);
16
17     i = 0;
18     while (input[i] != 0)
19     {
20         printf("%d ", input[i++]);
21     }
22     puts("");
23
24     return 0;
25 }
```

The output window shows the program's execution results. It displays the prompt "배열에 저장할 정수를 여러 개 입력하시오. 0을 입력하면 입력을 종료합니다." and the user's input "32 28 67 74 01 58 32 0". The program then prints the array contents: "32 28 67 74 1 58 32". The user's input "계속하려면 아무 키나 누르십시오 . . ." is also visible.

## 10 장-함수 복습

### 1.배열과 함수 사용해 배열 원소에 값 더하기



The image shows a screenshot of the Microsoft Visual Studio IDE. The main window displays a C program named `page472.c`. The program defines three functions: `incrementary`, `printary`, and `main`. The `main` function initializes an array `data` with values `{ 4, 7, 2, 3, 5 }`, calculates its length, and calls `printary` and `incrementary` (with an increment of 3) before calling `printary` again. The `incrementary` function uses a `for` loop to increment each element of the array by `n`. The `printary` function uses a `for` loop to print each element of the array.

```
1  #include <stdio.h>
2
3  void incrementary(int ary[], int n, int SIZE);
4  void printary(int *data, int SIZE);
5
6
7  int main(void)
8  {
9      int data[] = { 4, 7, 2, 3, 5 };
10     int aryLength = sizeof(data) / sizeof(int);
11
12     printary(data, aryLength);
13     incrementary(data, 3, aryLength);
14
15     printf("배열 원소에 각각 3을 더한 결과: \n");
16
17     printary(data, aryLength);
18
19     return 0;
20 }
21
22 void incrementary(int ary[], int n, int SIZE)
23 {
24     for (int i = 0; i < SIZE; i++)
25     {
26         *(ary + i) += n;
27     }
28 }
29
30 void printary(int *data, int SIZE)
31 {
32     for (int i = 0; i < SIZE; i++)
33     {
34         printf("%2d ", data[i]);
35     }
36     printf("\n");
37 }
```

A terminal window titled `C:\Windows\system32\cmd.exe` is open in the bottom right corner, showing the output of the program:

```
4 7 2 3 5
배열 원소에 각각 3을 더한 결과:
7 10 5 6 8
계속하려면 아무 키나 누르십시오 . . .
```



## 2. 난수 발생 함수를 사용하여 난수 맞추기

```
numberguess.c  X  myself.c
numberguess.c  (전역 범위)

1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  #define MAX 100
7
8  int main(void)
9  {
10     int guess, input;
11
12     srand((long)time(NULL));
13     guess = rand() % MAX + 1;
14
15     printf("1에서 %d 사이에서 한 정수가 결정되었습니다.\n", MAX);
16     printf("이 정수는 무엇일까요? 입력해 보세요. : ");
17
18     while (scanf("%d",&input))
19     {
20         if (input > guess)
21         {
22             printf("입력한 수보다 작습니다. 다시 입력하세요. :");
23         }
24         else if (input < guess)
25         {
26             printf("입력한 수보다 큼니다. 다시 입력하세요. :");
27         }
28         else
29         {
30             puts("정답입니다.");
31             break;
32         }
33     }
34     return 0;
35 }
36
```

7장 반복, 10장 함수를 발표하면서

복습한 자료입니다.

## 7장

### 1. do while문

#### Do While문은?

➡ 반복 몸체를 수행 후에  
반복 조건을 평가하는 구문

#### Do While문의 구성

Do{

반복몸체;

}While(반복조건);

# Do While문의 특징

반복 횟수가 정해지지 않은 구문에 유용

반복몸체가 최소 한 번은 실행(특별한 구문이 없다면)

센티널 값 검사에 유용

(반복을 종료시키는 특정한 값)

# Do While문의 실행순서

- 참일 경우

반복몸체  
다시 수행

- 거짓일 경우

Do While문  
종료





## 2.for문

### For문은?

→ **제어변수**의 초기화와 증가/감소를  
일정한 영역에서 하는 구문

(반복의 횟수를 제어하는 변수)

### For문의 구성

For(초기화; 반복조건; 증가/감소)



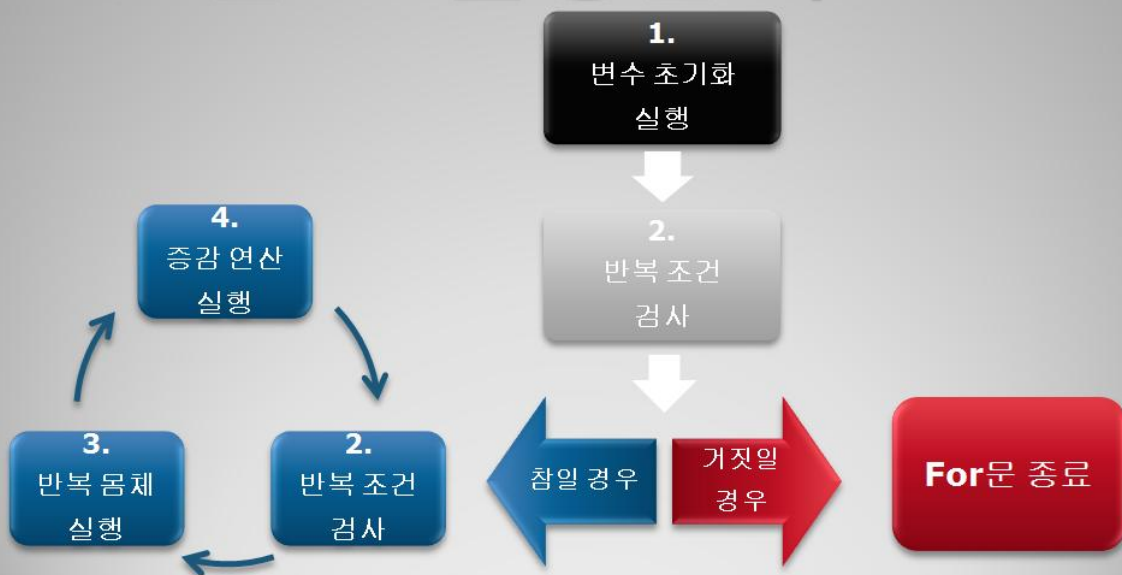
{반복몸체;}

# For문의 특징

1. 초기화는 한 번만 수행

2. 반복조건을 제거하면 반복을 무한히 함

# For문의 실행순서



## 10장

함수 매개변수의 활용

### 매개변수

- 함수를 호출하는 부분에서  
함수 몸체로 값을 전달하는 것이 목적인 변수

-함수 정의에서 매개 변수가

- 필요한 경우-자료형과 변수명으로 나타냄
- 필요하지 않은 경우-키워드 void를 기술함

### 반환 값

매개변수와는 반대로,  
함수가 작업을 수행한 후  
다시 함수를 호출한 영역으로  
결과 값을 전달할 때 이용

### 매개 변수와의 차이

매개변수는 여러 개를 사용 가능하지만  
반환 값은 하나만 이용 할 수 있음

## 형식매개변수

- 함수정의에서 기술되는 매개변수 목록의 변수

### • 특징

함수 내부에서만  
사용 가능한 변수

## 실매개변수

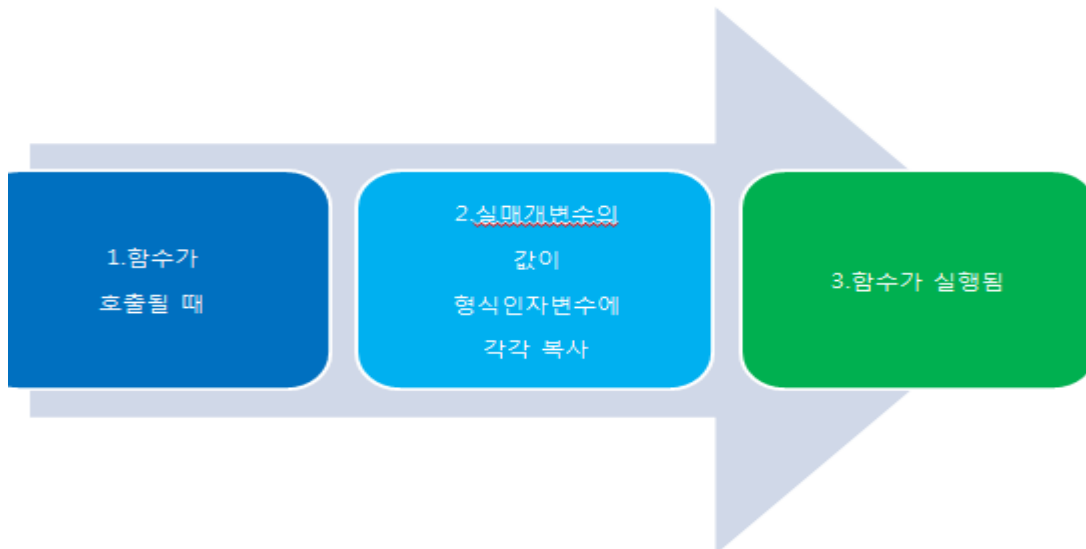
(실인자/인자)

- 함수를 호출할 때 기술되는 변수/값

### • 특징

- 1. 함수헤더에 정의된 자료 유형과 순서가 일치해야 함
- 2. 상수/변수/함수호출/수식이 가능

## 값에 의한 호출



따라서, 값에 의한 호출의 함수에서  
일반 매개변수의 값을 수정해도  
함수를 호출한 곳의 실매개변수의 값은 변하지 않음