

c언어 정리

1장 프로그래밍언어 개요

1. 프로그램(Program)이란 컴퓨터와 스마트폰에서 특정 목적의 작업을 수행하기 위한 관련 파일의 모임이라고 한다.
2. 프로그래머(Programmer)란 컴퓨터와 스마트폰 등의 정보기기에서 사용되는 프로그램을 만드는 사람을 말한다.
3. 프로그래밍 언어란 프로그램을 개발하기 위해 사용하는 언어이고 사람이 컴퓨터에게 지시할 명령어를 기술하기 위하여 만들어진 언어이다. 대표적인 언어로는 FORTAN, ALGOL, BASIC, COBOL, PASCAL, C, C++, VISUALBASIC, DELPHI, JAVA, OBJECTIVE-C, PERL, JSP, JAVASCRIPT, PYTHON, C#, GO등이 있다.
4. 최초의 프로그래밍 언어는 포트란이며 이를 개발한 사람은 1950년 중반 IBM에서 근무하는 28세 젊은 과학자인 존 배커스가 개발했다. 포트란은 수식변환기라는 의미의 약자로 공학과 과학 분야에서 계산 위주로 사용을 목적으로 개발된 프로그래밍 언어다. 하지만 최초의 프로그래머는 영국의 오거스타 에이다이며 현대 컴퓨터 프로그래밍 역사의 기원자이다.
5. 컴퓨터는 하드웨어와 소프트웨어로 구성되며 하드웨어의 구성요소로는 중앙처리장치(CPU), 주기억장치(main memory), 보조기억장치(second-ary memory), 입력장치(input device), 출력장치(output device)가 있다. 소프트웨어는 컴퓨터가 수행할 작업을 지시하는 전자적 명령어들의 집합으로 구성된 프로그램이며 소프트웨어도 크게 응용소프트웨어와 시스템 소프트웨어로 나눈다.시스템 소프트웨어는 컴퓨터가 잘 작동하도록 도와주는 기본 소프트웨어이고 응용소프트웨어는 인터넷,게임,동영상 보기 등과 같은 특정업무에 활용되는 소프트웨어를 말한다.
6. 운영체제는 컴퓨터 하드웨어 장치의 전반적인 작동을 제어하고 조정하며 사용자가 최대한 컴퓨터를 효율적으로 사용할 수 있도록 돕는 시스템 프로그램이다.
7. 기계어란 컴퓨터가 유일하게 바로 인식하는 언어이며 전기의 흐름을 표현하는 1과 흐르지 않음을 표현하는 0을 표현한다. 즉 사람이 프로그래밍 언어로 컴퓨터에게 명령을 내리기 위해서는 프로그래밍 언어를 기계어로 변환하는 통역사인 컴파일러가 필요하다.
8. 어셈블리어란 기계어를 프로그래머인 사람이 좀 더 이해하기 쉬운 기호 형태로 일대일 대응시킨 프로그래밍 언어이다. 이 어셈블리어로 작성된 프로그램을 기계어로 바꾸어 주는 프로그램이 어셈블러이다.

9. C언어는 1972년 데니스 리치가 개발한 프로그래밍 언어이다. 운영체제인 유닉스를 개발하기 위해 C언어를 개발했고 1970년 개발한 B언어에서 유래된 프로그래밍 언어이다.
10. 프로그래밍 언어 개발순서로는 ALGOL, BCPL, B, C, OBJECTIVE-C, C++, JAVA, C#, GO순이다.
11. C언어의 특징으로는 다소 배우기 어렵다는 단점이 있지만 절차지향 언어이며 간결하고 효율적인 언어이고 시스템의 세세한 부분까지 제어가 가능하며 포인터 및 메모리관리에 효율적으로 사용하여 실행속도가 빠르다는 장점이 있다. 또한 이식성이 좋다.
12. C언어를 배워야 하는 이유는 현장에서 많이 쓰이는 자바나 C#, OBJECTIVE-C 등의 뿌리가 C이므로 C언어를 알면 자바나 C#, OBJECTIVE-C 뿐만 아니라 그 이후의 프로그래밍 언어들은 습득하기가 매우 쉬워진다.
13. 아스키코드란 영문 알파벳을 사용하는 대표적인 문자 인코딩 방법이며 컴퓨터 뿐만 아니라 통신장비와 같은 문자를 사용하는 대부분의 장치에서 사용되며 아스키코드를 기반으로 하고 있는 장치들 간에는 이 표준에 의해서 서로 문자를 주고 받을 수 있다.
14. 유니코드는 전 세계 모든 문자를 컴퓨터에서 일관되게 표현하고 다룰 수 있도록 설계된 산업표준이다.
15. 알고리즘이란 어떠한 문제를 해결하기 위한 절차나 방법으로 명확히 정의된 유한개의 규칙과 절차의 모임이다.

2장 C 프로그래밍 첫걸음

1. 프로그래밍 구현과정 5단계로는 프로그램 구상 -> 소스편집 -> 컴파일 -> 링크 -> 실행이다.
2. 컴파일러는 고급언어인 프로그래밍 언어로 작성된 소스파일에서 기계어로 작성된 목적파일(object file)을 만들어내는 프로그램이다.
3. 링커는 하나 이상의 목적파일을 하나의 실행파일로 만들어주는 프로그램이다. 여러개의 목적파일을 연결하고 참조하는 라이브러리를 포함시켜 하나의 실행 파일을 생성하는데, 말뜻 그대로 이 과정을 링크 또는 링킹 이라고 한다.
4. 라이브러리란 프로그래머마다 새로 작성할 필요 없이 개발환경에서 미리 만들어 컴파일해 저장해 놓은 모듈을 라이브러리라 한다.

5. 빌드란 컴파일과 링크 과정을 하나로 합친 것을 말한다.
6. 디버깅이란 프로그램 개발 과정에서 발생하는 다양한 오류를 찾아 소스를 수정하여 다시 컴파일 링크 실행하는 과정을 말하며 이를 도와주는 프로그램을 디버거라고 한다.
7. IDE란 통합개발환경이며 프로그램 개발에 필요한 편집기, 컴파일러, 링커, 디버거 등을 통합하여 편리하고 효율적으로 제공하는 개발환경을 말한다.
8. 기본적인 출력문으로는 `puts("hello world!");` , `printf("hello world!");` 형식이 있다.
9. 솔루션은 프로젝트의 모임이며 프로젝트는 소스파일로 구성할 수 있다.
10. `main`을 `mein`으로 쓰는 링크오류 또는 문장 끝에 `;`을 누락시키는 컴파일 오류를 조심해야 한다.

3장 자료형과 변수

1. C프로그램은 적어도 `main()` 함수 하나는 구현되어야 응용 프로그램으로 실행될 수 있다. 즉 예시로 간단한 소스파일이다.

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

```
int main()
{
    puts("첫 c 프로그램");

    return 0;
}
```

2. 키워드란 예약어라고도 하며 미리 정의해놓은 단어를 말한다. 예시로 `auto`, `do`, `goto`, `signed`, `void`, `sizeof`, `static`, `while`, `for`, `enum`, `if`, `int`, `char`, `case`, `const` 등이 있다.
3. 식별자란 프로그래머가 자기 마음대로 정의해서 사용하는 단어들을 말한다. 예시로 `age`, `year` 등이 있으며 이 식별자는 규칙이 있다. 영문자, 숫자, 밑줄(`_`)로 구성되며 식별자의 첫 문자로 숫자가 나올수 없으며 키워드를 식별자로 사용할수 없다. 또한 대소문자를 구분하며 식별자 중간에 공백이 들어갈수 없다.
4. indentation 즉 들여쓰기는 프로그램의 이해력을 돕는데 매우 중요함으로 적극적으로 활용

하는게 좋다.

5. 주석은 타인은 물론이고 자신을 위해서라도 반드시 필요하며 소스를 보는 모든 사람이 이해할 수 있도록 도움이 되는 설명이다. 한줄 주석사용법은 //이며 여러줄을 주석처리 하려면 /* */으로 사용하면 된다.

6. 변수는 저장공간이라고 하며 선언되는 자료형에 따라 변수의 저장공간 크기와 저장되는 자료값의 종류가 결정된다. 변수를 사용하기 위해서는 원칙적으로 사용전에 변수선언 과정이 반드시 필요하다. 예시로 double height; 가 있으며 자료형 double에 height로 실수를 저장하는 공간을 만드는 것이다.

7. 대입 연산자 '='는 오른쪽에 위치한 값을 이미 선언된 왼쪽 변수에 저장한다라는 의미이다. 예시로

```
int age;  
age = 20;
```

이렇게 하면 변수 age에 정수 20을 저장하게 된다.

8. signed 자료형이란 0과 음수를 모두 표현 할수 있고 예시로 signed int가 있다면 signed 는 생략되고 int로 쓸수 있다.

9. unsigned 자료형은 0과 양수만을 저장할수 있으며 unsigned short , unsigned long으로 쓰인다.

10. 정수 자료형의 표현범위로는 표가 있다.

음수지원 여부	자료형	크기	표현범위
부호가 있는 정수형 signed(음수만)	signed short	2바이트	$-2^{15} \sim 2^{15} - 1$
	signed int	4바이트	$-2^{31} \sim 2^{31} - 1$
	signed long	4바이트	$-2^{31} \sim 2^{31} - 1$
부호가 없는 정수형 unsigned(0과양수만)	unsigned short	2바이트	$2^{16} - 1$
	unsigned int	4바이트	$2^{32} - 1$
	unsigned long	4바이트	$2^{32} - 1$

11. 부동소수 자료형은 float, double, long double 세가지가 있다.

자료형	크기	정수의 유효자릿수	표현범위
float	4바이트	6 ~ 7	1 . 1 7 5 4 9 4 3 5 1 E - 3 8 F 에 서 3.402823466E+38F까지
double	8바이트	15 ~ 16	2.22507385850720114E-308에서 1.7976931348623158E+308까지
long double	8바이트	15 ~ 16	2.2250738585072014E-308에서 1.7976931348623158E+308까지

12. 문자형 자료형으로는 char, signed char, unsigned char 세가지가 있다.

자료형	저장공간 크기	표현범위
char	1바이트	-128에서 127까지
signed char	1바이트	-128에서 127까지
unsigned char	1바이트	0에서 255까지

13. 연산자 size of를 이용하면 자료형, 변수, 상수의 저장공간 크기를 바이트 단위로 알수 있다. 예제로 소스파일이 있다.

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

```
int main(void)
```

```
{
printf("자료형: 크기(바이트)\n");
printf("char :%d %d\n", sizeof(char), sizeof(unsigned char));
printf("short :%d %d\n", sizeof(short), sizeof(unsigned short));
printf("int :%d %d\n", sizeof(int), sizeof(200));
printf("long :%d %d\n", sizeof(long), sizeof(300L));
printf("          long long :%d %d\n", sizeof(long long), sizeof(900LL));
printf("float :%d %d\n", sizeof(float), sizeof 3.14F);
printf("double :%d %d\n", sizeof(double), sizeof 3.14);
printf("          long double :%d %d\n", sizeof(long double), sizeof 3.24L);

return 0;
}
```

```

D:\Windows\system32\cmd.exe
자료형 : 크기<바이트>
      char :1 1
     short :2 2
       int :4 4
       long :4 4
    long long :8 8
       float :4 4
      double :8 8
 long double :8 8
계속하려면 아무 키나 누르십시오 . . .
  
```

14. 오버플로와 언더플로가 있다. 자료형의 범주에서 벗어난 값을 저장하면 오버플로 또는 언더플로가 발생한다. 예시로 소스파일이 있다.

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

```
int main(void)
```

```
{
```

```
    unsigned char uc = 255 + 1;
```

```
    shorts = 32767 + 1;
```

```
    float      min = 1.175E-50;
```

```
    float      max = 3.403E39;
```

```
    printf("%u\n", uc); //오버플로 발생
```

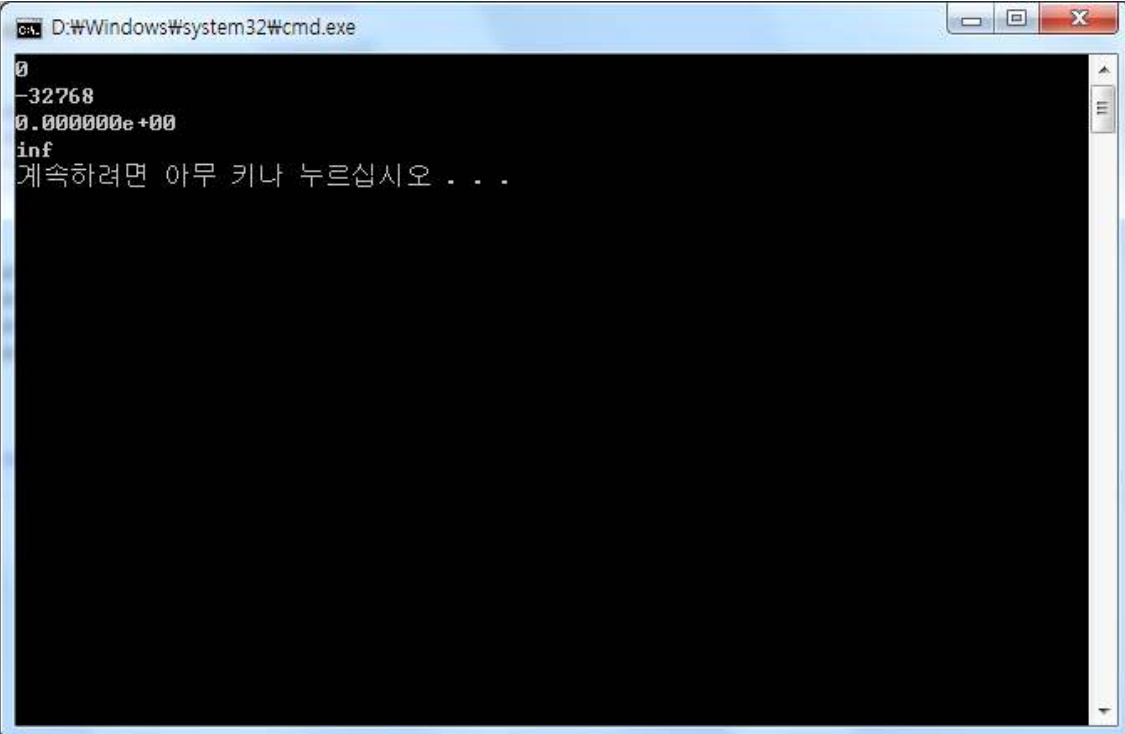
```
    printf("%d\n", s); //오버플로 발생
```

```
    printf("%e\n", min); // 언더플로 발생
```

```
    printf("%f\n", max); // 오버플로 발생
```

```
    return 0;
```

```
}
```



```
CA. D:\Windows\system32\cmd.exe
0
-32768
0.000000e+00
inf
계속하려면 아무 키나 누르십시오 . . .
```

15. 상수는 이름없이 있는 그대로 표현한 자료값이나 이름이 있으나 정해진 하나의 값으로 사용되는 자료값을 말한다. 크게 리터럴 상수와 심볼릭 상수로 구분한다.

구분	표현방법	설명	예
리터럴 상수 (이름이 없는 상수)	정수형 실수형 문자 문자열 상수	다양한 상수를 있는 그대로 기술	3 2 , 0 2 5 , 0xf3,10u,100L, 30LL, 3.2F, 3.15E3, 'A', '\n','\0','\24',' \x2f',"c 언어 “,” 프로그래밍 언어 \n"
심볼릭 상수 (이름이 있는 상수)	const 상수	키워드 const를 이용한 변수 선언과 같으며 수정할수 없는 변수 이름으로 상수정의	const double PI = 3.141592;
	매크로 상수	전처리기 명령어 #define으로 다양한 형태를 정의	#define PI 3.141592
	열거형 상수	정수 상수 목록 정의	enum bool {false , true};

16. 이스케이프 문자란 알람 소리와 새로운 줄 등을 의미하는 특수한 문자에 한하여 \n과 같이 역슬래쉬 \와 문자의 조합으로 표현하는 문자를 이스케이프 문자라 한다.

제어문자 이름	영문 표현	코드값 (십진수)	\ddd (팔진수)	제어문자 표현	의미
널문자	NULL	0	\000	\0	아스키코드 0번
경고	BEL(bell)	7	\007	\a	경고음이 울림
백스페이스	BS(Back Space)	8	\010	\b	커서를 한문자 뒤로 이동
수평탭	HT(Horizontal Tab)	9	\011	\t	커서를 수평으로 다음 탭만큼 이동
개행문자	LF(Line Feed)	10	\012	\n	커서를 다음줄로 이동
수직탭	VT(Vertical Tab)	11	\013	\v	수직으로 이동하여 탭만큼 이동
폼피드	FF(Form Feed)	12	\014	\f	새 페이지의 처음으로 이동
캐리지 리턴	CR(Carriage Return)	13	\015	\r	커서를 현재 줄의 처음으로 이동
큰따옴표	Double quote	34	\042	\"	“ 문자
작은따옴표	single quote	39	\047	\'	‘ 문자

역슬래쉬	Backslash	92	\134	\\	\문자
------	-----------	----	------	----	-----

17. 이진수 8진수 16진수 표현방식은 이렇다.

2진수 : printf("%d ", 10); 8진수 : printf("%d ", 010); 16진수 : printf("%d ", 0x1b);

18. 지수표현 방식은

실수 e 또는 E를 사용하여 10의 지수표현 방식으로 나타낼 수 있다. 즉 3.14E+2는 3.14×10^2 (314.000000)을 나타낸다. 반대로 3.14-2는 3.14×10^{-2} (0.031400)로 나타낸다.

19. 실수형 리터럴 상수는 float, double, long double의 자료형으로 나뉜다. 즉 소수는 double 유형이며 float상수는 숫자 뒤에 f나 F를 붙인다. long double 상수는 숫자뒤에 L또는 l을 붙여 표시한다. 그러나 l은 숫자1과 구분이 어려우므로 대문자 L을 사용한다.

리터럴상수 접미어가 표로 있다.

구분	표현방법	접미어	예
정수형 상수	unsigned (int)	u	1000000u
		U	24563U
	unsigned long	ul	87252987ul
		UL	10000000UL
	unsigned longlong	ull	1000000000000000ull
		ULL	2400000000000000ULL
실수형 상수	float	l	-209871
		L	76528876L
	long double	ll	1000000000000000ll
		LL	-2450000000000000LL
	float	f	3.14f
		F	3354.9876F
실수형 상수	long double	l	4356.9876l
		L	5634.987276L

20. 심볼릭 const상수는 변수를 수정할수 없게 하고 변수선언 시 자료형 또는 변수 앞에 키워드 const가 놓이면 이 변수는 심볼릭 상수가 된다. 상수는 변수선언 시 반드시 초기값을 저장해야한다. 예시로 const double RATE = 0.03;을 선언하면 RATE변수 0.03을 수정할수 없다.

21. 열거형 상수 enum은 정수형 상수목록 집합을 정의하는 자료형이다.

예시로 enum DAY {SUN, MON, TUE, WED, THU, FRI, SAT} 처럼 표현한다.

0 1 2 3 4 5 6

4장 전처리와 입출력

1. 전처리 지시자 `#include`는 헤더파일을 삽입하는 지시자이다. `stdio.h`를 사용하기 위해 사용한다.

주요 헤더파일로는 표가 있다.

헤더파일	파일이름	파일내용
<code>stdio.h</code>	STanDard Input Output (표준 입출력)	표준 입출력 함수와 상수
<code>stdlib.h</code>	STanDard LiBrary(표준함수)	주요 메모리 할당 함수와 상수
<code>math.h</code>	<code>math</code>	수학관련 함수와 상수
<code>string.h</code>	<code>string</code>	문자열 관련 함수와 상수
<code>time.h</code>	<code>Time</code>	시간관련 함수와 상수
<code>ctype.h</code>	<code>Character TYPE</code>	문자관련 함수와 상수
<code>limits.h</code>	<code>limits</code>	정수 상수 등 여러 상수
<code>float.h</code>	<code>float</code>	부동소수에 관련된 각종 상수

2. 전처리 지시자 `#define`은 매크로 상수를 정의하는 지시자이다.

예시로 소스파일이 있다.

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

```
#define KPOP 50000000
```

```
#define PI 3.14
```

```
#define PRT printf("매크로상수예제종료\n")
```

```
int main(void)
```

```
{
```

```
double radius = 2.83;
```

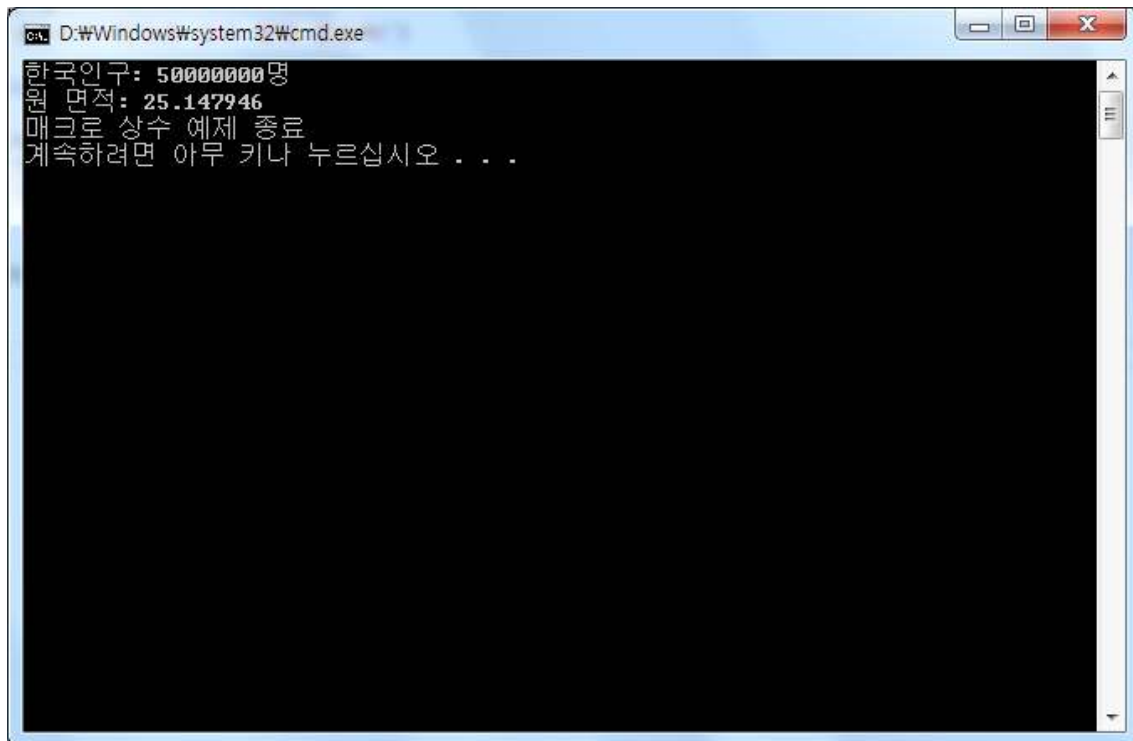
```
printf("한국인구: %d명\n", KPOP);
```

```
printf("원면적: %f\n", radius * radius * PI);
```

```
PRT
```

```
return 0;
```

```
}
```



3. 출력함수 printf()는 일련의 문자 및 값으로 서식을 지정하여 표준 출력인 stdout에 출력하는 함수이다. printf()의 인자는 크게 형식문자열과 출력할 목록으로 구분되어 출력목록의 각 항목을 형식문자열에서 %d와 같이 %로 시작하는 형식지정자 순서대로 서식화하여 그 위치에 출력한다. 예시로

```
int term = 15
printf(" %d의 두배는 %d입니다.\n", term, 2*term);
```

이렇게 하면 15의 두배는 30입니다.가 출력된다.

4. 정수의 십진수출력은 %d와 %i이고 8진수는 %o 0이붙는(leading) 출력을 하려면 #을 삽입하여 %#o를 이용한다. 16진수는 %x를 이용하며 0x를 붙여 출력하려면 #을 삽입하여 %#x또는 %#X를 이용한다. 예시로 소스파일이 있다.

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

```
int main(void)
```

```
{
```

```
int n = 16, ret_value = 0;
```

```
ret_value = printf("Hello!\n");
```

```
printf("printf() 반환값(출력된문자수): %d\n", ret_value);
```

```
ret_value = printf("출력값: %d %i %o %#o %#x %#X\n", n, n, n, n, n, n);
```

```
printf("반환값(출력된문자수): %d\n", ret_value);
```

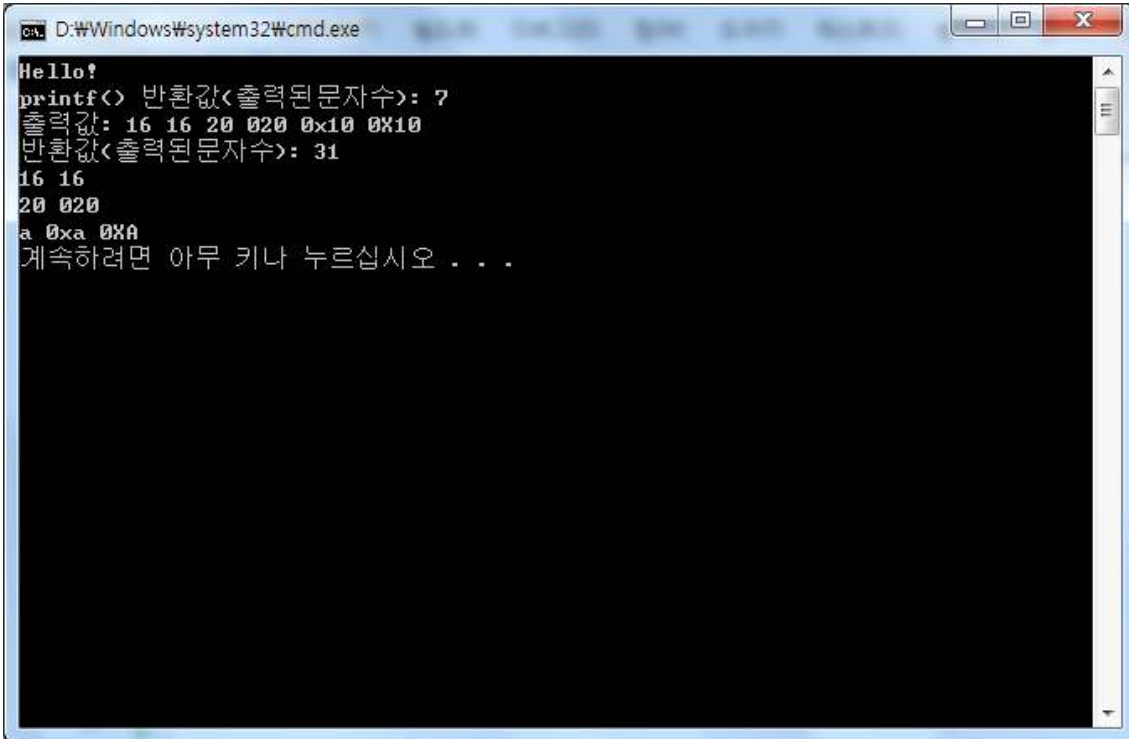
```
printf("%d %i\n", 16, 16);
```

```
printf("%o %#o\n", 16, 16);
```

```
printf("%x %#x %#X\n", 10, 10, 10);
```

```
return 0;
```

```
}
```



```

D:\Windows\system32\cmd.exe
Hello!
printf() 반환값(출력된문자수): 7
출력값: 16 16 20 020 0x10 0X10
반환값(출력된문자수): 31
16 16
20 020
a 0xa 0XA
계속하려면 아무 키나 누르십시오 . . .
```

5. 실수를 위한 출력 %f도있다. 실수의 간단한 출력을 위한 형식지정자는 %f이며 %f는 실수를 기본적으로 3.400000와 같이 소수점 6자리까지 출력한다. 함수 printf()에서 실수출력으로 %f와 %lf는 같이 사용된다. 예시로 소스파일이 있다.

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

```
int main(void)
```

```
{
```

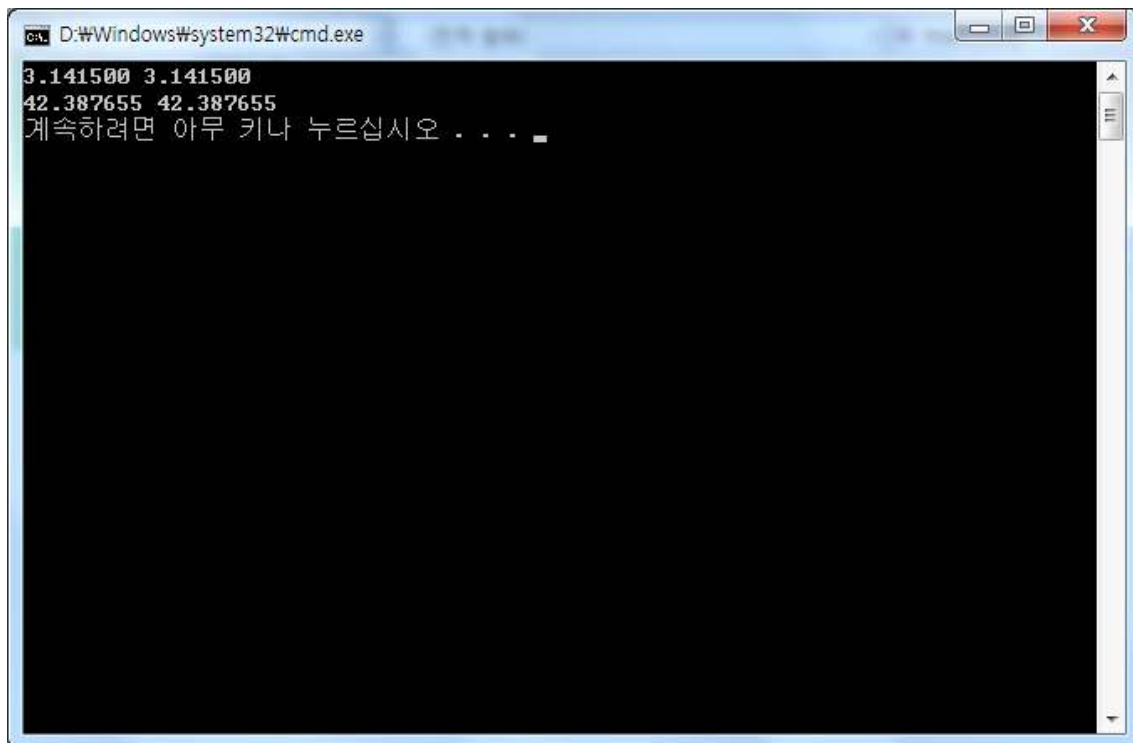
```
floatf = 3.1415F;
```

```
doubled = 42.3876547;
```

```
printf("%f %lf\n", f, f);
```

```
printf("%f %lf\n", d, d);
```

```
return 0;
}
```



6. 출력폭을 지정할수 있는데 예로 %8d는 십진수 8자리를 폭에 출력한다. 지정한 출력폭이 출력할 내용보다 넓으면 정렬은 기본적으로 오른쪽이다. 다음은 정수 7629를 %8d로 출력한 모습이다.

				7	6	2	9
--	--	--	--	---	---	---	---

```
printf("%8d\n", 7629);
```

반대로 왼쪽으로 정렬 시키려면 %-8d로 출력하면된다.

7	6	2	9				
---	---	---	---	--	--	--	--

```
printf("%-8d\n", 7629);
```

부동소수 %10.3f도 출력할수 있다.

				3	2	.	3	6	9
--	--	--	--	---	---	---	---	---	---

```
printf("%10.3f\n", 32.369);
```

반대로 왼쪽 정렬을 시키려면 %-10.3f로 하면된다.

3	2	.	3	6	9				
---	---	---	---	---	---	--	--	--	--

```
printf("%-10.3f\n", 32.369);
```

7. 함수 scanf()는 입력함수로서 %s와 %d같은 동일한 형식지정자를 사용하며 주소연산자 &를 꼭 사용해줘야 한다. scanf()는 출력함수 printf()와 달리 %f와 %lf를 구분한다. %f는 float에 쓰이고 %lf는 double에 쓰인다.

예시로 이렇게 선언한다. `int year = 0;`

```
scanf("%d", &year);
```

8. 다양한 형식지정자가 있다.

형식지정자	콘솔 입력값의 형태	입력변수 인자 유형
%d	십진수로 인식	정수형 int변수에 입력값 저장
%i	십진수로 인식하며, 단 입력값에 0이 앞에 붙으면 8진수로 0x가 붙으면 16진수로 인식하여 저장	정수형 int변수에 입력값 저장
%u	unsigned int로 인식	정수형 unsigned int변수에 입력값 저장
%o	8진수로 인식	정수형 int변수에 입력값 저장
%x, %X	16진수로 인식	정수형 int변수에 입력값 저장
%f	부동소수로 인식	부동소수형 float변수에 입력값 저장
%lf	부동소수로 인식	부동소수형 double변수에 입력값 저장
%e, %E	지수 형태의 부동소수로 인식	부동소수형 float변수에 입력값 저장
%c	문자로 인식	문자형 char변수에 입력값 저장
%s	문자열로 인식	문자열 저장할 배열에 입력값 저장
%p	주소값으로 인식	정수형 unsigned int 변수에 입력값 저장

9. 함수 scanf()의 경고

visual c++ 2005 이후부터 함수 scanf()는 보안문제로 더 이상 사용을 권하지 않고 있으며 사용시엔 오류가 발생하기 때문에 꼭 `#define _CRT_SECURE_NO_WARNINGS`를 기술해야 한다.

5장 연산자

1. 산술연산자 +, -, *, /, %

더하기, 빼기, 곱하기, 나누기, 나머지 연산자이며 밑에 예제 소스가있다.

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

```
int main(void)
```

```
{
```

```
printf("3 + 4 ==> %d\n", 3 + 4);
```

```
printf("3.4 - 4.3 ==> %f\n", 3.4 - 4.3);
```

```
printf("3.4 * 4.3 ==> %f\n", 3.4 * 4.3);
```

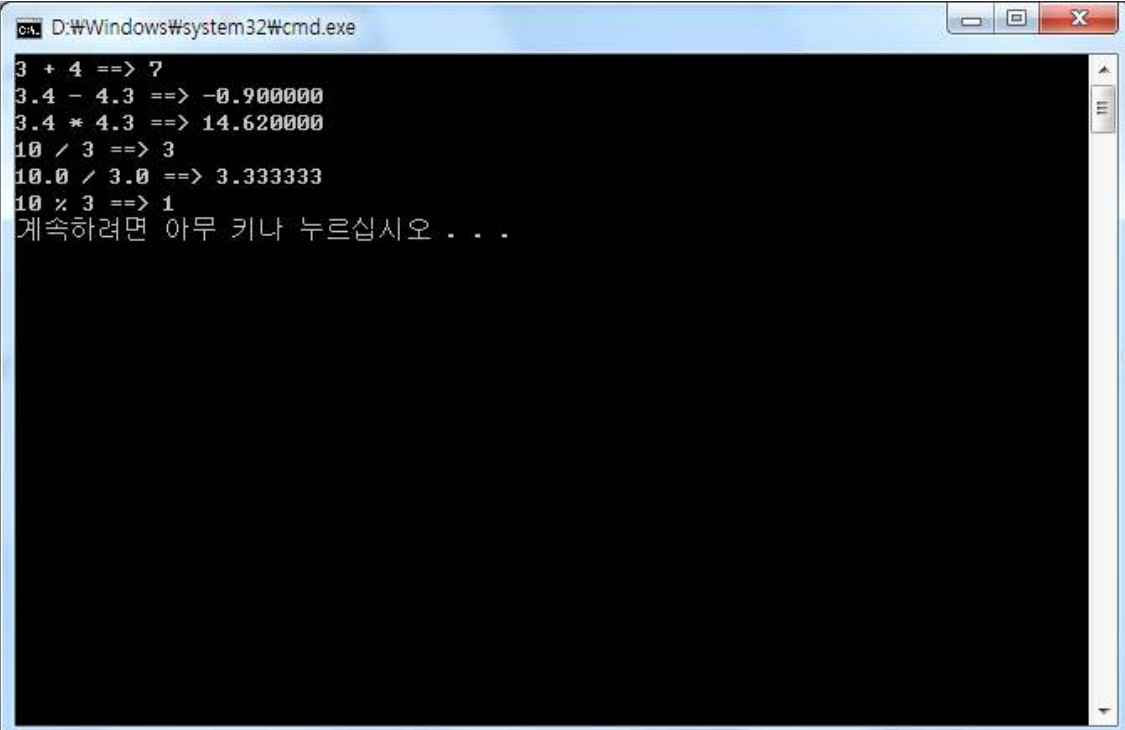
```
printf("10 / 3 ==> %d\n", 10 / 3);
```

```
printf("10.0 / 3.0 ==> %f\n", 10.0 / 3.0);
```

```
printf("10 %% 3 ==> %d\n", 10 % 3);
```

```
return 0;
```

```
}
```



The screenshot shows a Windows command prompt window titled "D:\Windows\system32\cmd.exe". The output of the program is displayed as follows:

```
3 + 4 ==> 7
3.4 - 4.3 ==> -0.900000
3.4 * 4.3 ==> 14.620000
10 / 3 ==> 3
10.0 / 3.0 ==> 3.333333
10 %% 3 ==> 1
계속하려면 아무 키나 누르십시오 . . .
```

2. 대입연산자 =

대입연산자는 =으로 연산자 오른쪽의 연산값을 변수에 저장하는 연산자이다. 대입연산자의 왼쪽부분에는 반드시 하나의 변수만이 올 수 있다. 이 하나의 변수를 왼쪽을 의미하는 left 단어에서 l-value라 하며 오른쪽에 위치하는 연산식의 값을 right단어에서 r-value라 한다. 밑에 예제소스가 있다.

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

```
int main(void)
```

```
{
```

```
int a, b, c;
```

```
a = b = c = 5;
```

```
printf("a = a + 2 ==> %d\n", a = a + 2);
```

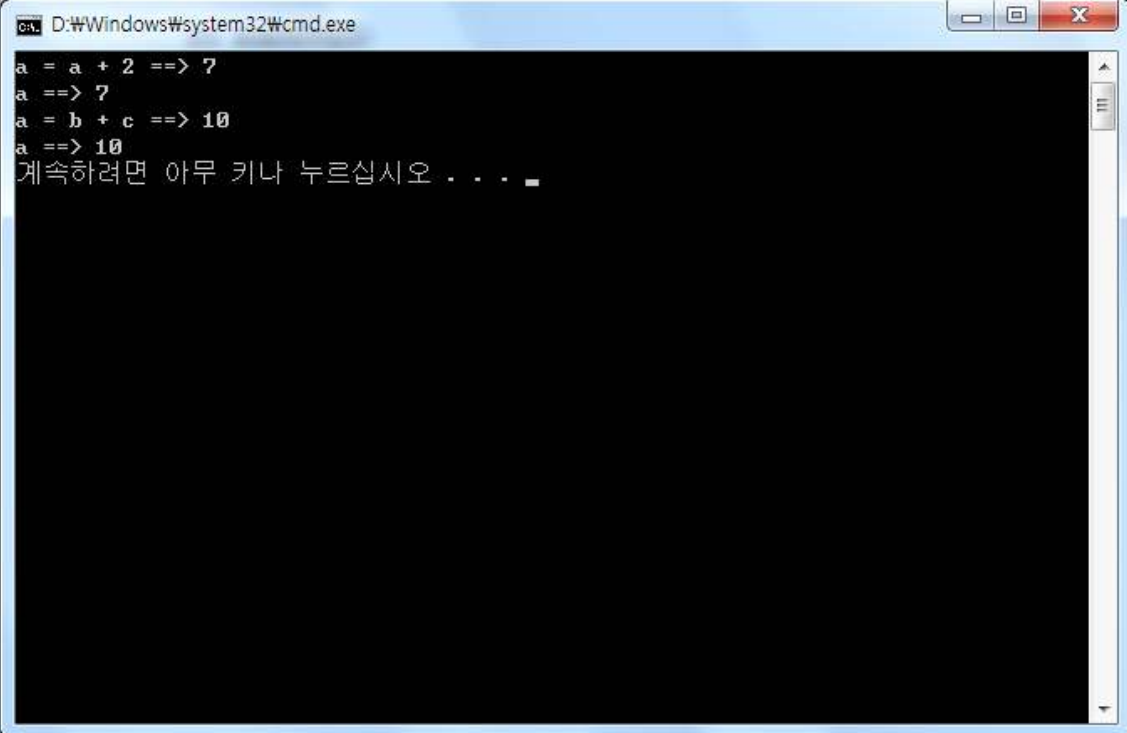
```
printf("a ==> %d\n", a);
```

```
printf("a = b + c ==> %d\n", a = b + c);
```

```
printf("a ==> %d\n", a);
```

```
return 0;
```

```
}
```



```
cmd. D:\Windows\system32\cmd.exe
a = a + 2 ==> 7
a ==> 7
a = b + c ==> 10
a ==> 10
계속하려면 아무 키나 누르십시오 . . .
```

3. 축약 대입연산자

$a = a + b$ 는 간결하게 $a += b$ 로 쓸 수 있다. 마찬가지로 $a = a - b$ 는 간결하게 $a -= b$ 로 쓸 수 있다. 이와같이 산술연산자와 대입연산자를 이어 붙인 연산자 $+=, -=, *=, /=, %=$ 을 축약대입연산자라 한다. 밑에 예제소스가 있다.

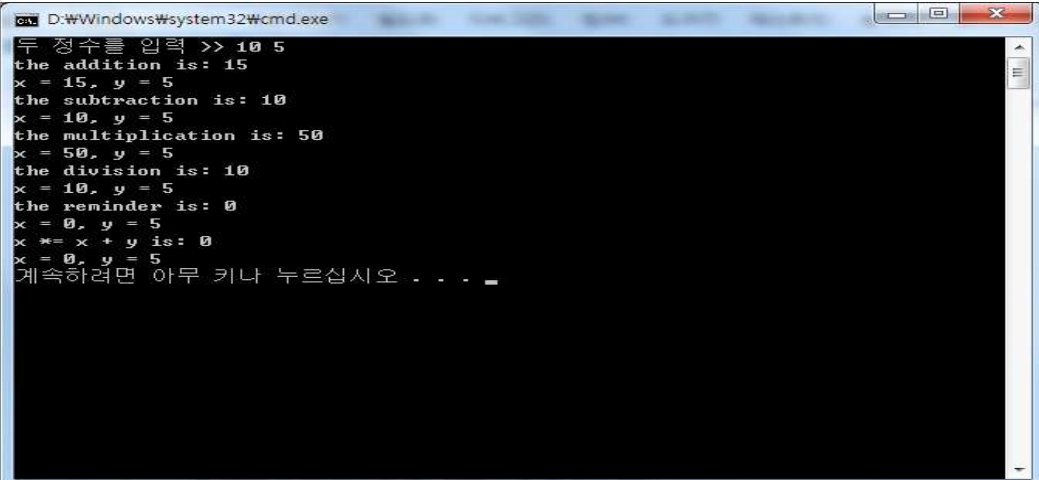
```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>

int main(void)
{
    int x = 5, y = 10;

    printf("두정수를입력>> ", &x, &y);
    scanf("%d%d", &x, &y);

    printf("the addition is: %d\n", x += y);
    printf("x = %d, y = %d\n", x, y);
    printf("the subtraction is: %d\n", x -= y);
    printf("x = %d, y = %d\n", x, y);
    printf("the multiplication is: %d\n", x *= y);
    printf("x = %d, y = %d\n", x, y);
    printf("the division is: %d\n", x /= y);
    printf("x = %d, y = %d\n", x, y);
    printf("the remainder is: %d\n", x %= y);
    printf("x = %d, y = %d\n", x, y);
    printf("x *= x + y is: %d\n", x *= x + y);
    printf("x = %d, y = %d\n", x, y);

    return 0;
}
```



```
cmd: D:\Windows\system32\cmd.exe
두 정수를 입력 >> 10 5
the addition is: 15
x = 15, y = 5
the subtraction is: 10
x = 10, y = 5
the multiplication is: 50
x = 50, y = 5
the division is: 10
x = 10, y = 5
the remainder is: 0
x = 0, y = 5
x *= x + y is: 0
x = 0, y = 5
계속하려면 아무 키나 누르십시오 . . .
```


4. 증감연산자 ++, --

변수값을 1증가시키거나 1감소시키는 문장이다. 예시로 `n++`은 ++이뒤에 있으므로 후위연산자이며 1이 증가되기 전 값이 연산 결과값이다. 반대로 `++n`은 전위 연산자로 1증가된 값이 연산 결과 값이다. `n--` 와 `--n`도 같은 원리이다. 밑에 예제소스가 있다.

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

```
int main(void)
```

```
{
```

```
int m = 10, n = 5;
```

```
int result;
```

```
result = m++ + --n;
```

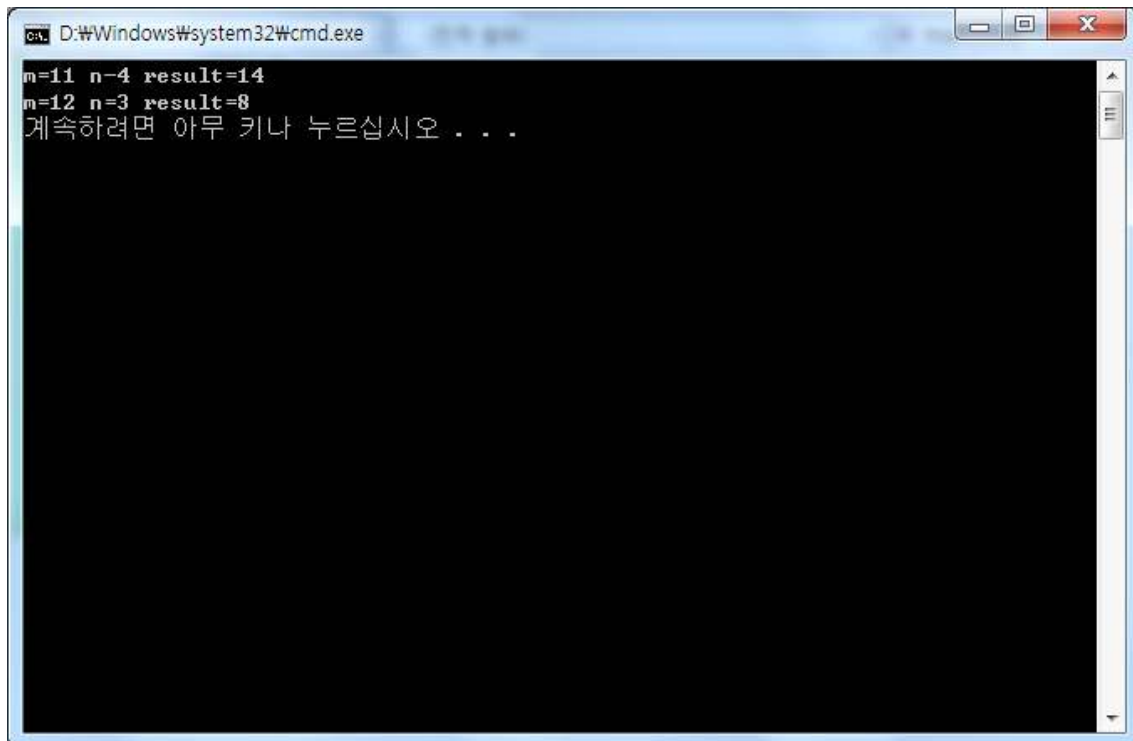
```
printf("m=%d n=%d result=%d\n", m, n, result);
```

```
result = ++m - n--;
```

```
printf("m=%d n=%d result=%d\n", m, n, result);
```

```
return 0;
```

```
}
```



```
cmd D:\Windows\system32\cmd.exe
m=11 n=4 result=14
m=12 n=3 result=8
계속하려면 아무 키나 누르십시오 . . .
```

5. 관계 연산자

관계연산자는 두 피연산자의 크기를 비교하기 위한 연산자이다. 관계연산자의 연산값은 비교 결과가 참이면 1, 거짓이면 0이다. 밑에 예제 소스가 있다.

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

```
int main(void)
```

```
{
```

```
printf("(3 > 4) 결과값: %d\n", (3 > 4));
```

```
printf("(3 > 4.0) 결과값: %d\n", (3 < 4.0));
```

```
printf("( 'a' <= 'b' ) 결과값: %d\n", ('a' <= 'b'));
```

```
printf("( 'z' <= 'a' ) 결과값: %d\n", ('z' <= 'a'));
```

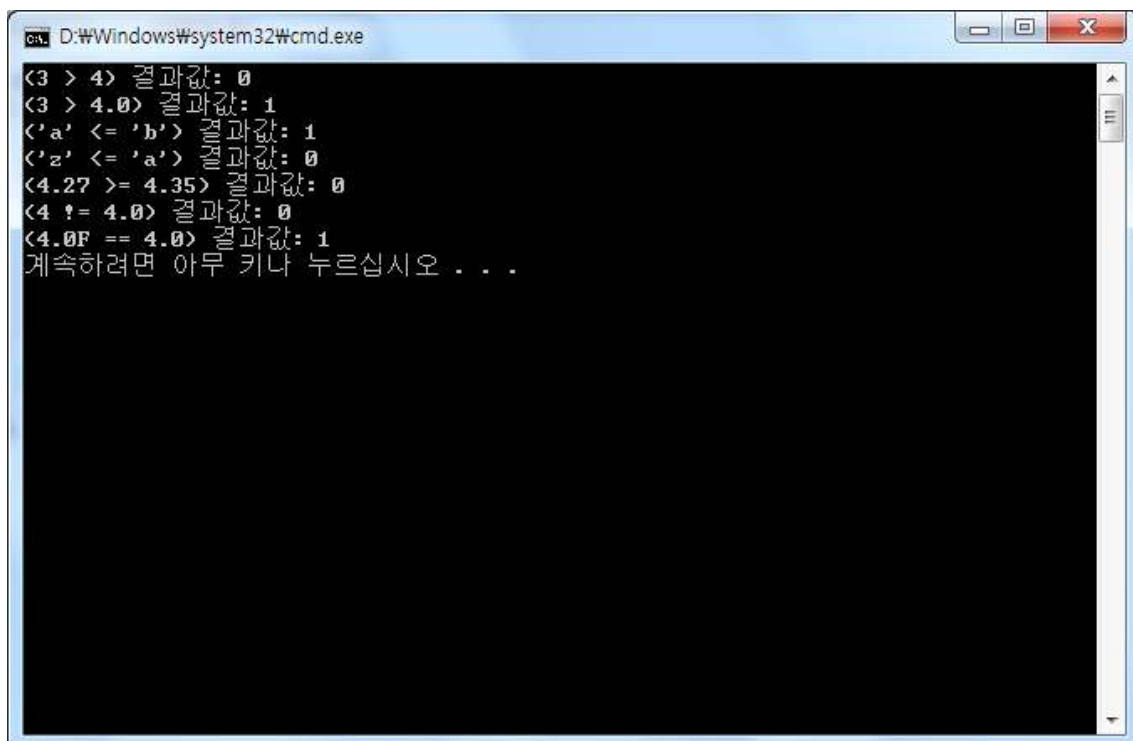
```
printf("(4.27 >= 4.35) 결과값: %d\n", (4.27 >= 4.35));
```

```
printf("(4 != 4.0) 결과값: %d\n", (4 != 4.0));
```

```
printf("(4.0F == 4.0) 결과값: %d\n", (4.0F == 4.0));
```

```
return 0;
```

```
}
```



```
ca. D:\Windows\system32\cmd.exe
<3 > 4> 결과값: 0
<3 > 4.0> 결과값: 1
<'a' <= 'b'> 결과값: 1
<'z' <= 'a'> 결과값: 0
<4.27 >= 4.35> 결과값: 0
<4 != 4.0> 결과값: 0
<4.0F == 4.0> 결과값: 1
계속하려면 아무 키나 누르십시오 . . .
```

6. 논리 연산자

&&, ||, ! 세가지 연산자가 있으며 각각 and, or, not의 논리연산을 의미한다. 결과가 참이면 1 거짓이면 0을 반환한다. 0이 아닌 모든 것은 참이다. 밑에 예제소스가 있다.

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

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

```
    char null = '\0', a = 'a'
```

```
    int zero = 0, n = 10;
```

```
    double dzero = 0.0, x = 3.56;
```

```
    printf("%d ", !zero);
```

```
    printf("%d ", zero && x);
```

```
    printf("%d\n", dzero || null);
```

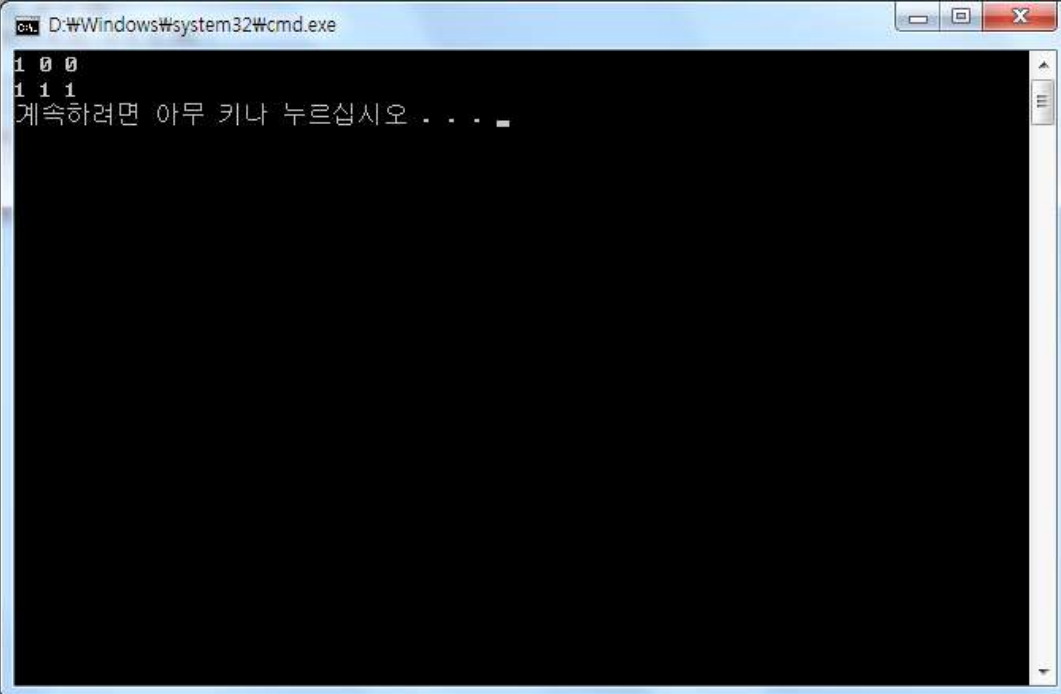
```
    printf("%d ", n && x);
```

```
    printf("%d ", a || null);
```

```
    printf("%d\n", "java" && "C Lang");
```

```
    return 0;
```

```
}
```



The screenshot shows a Windows command prompt window titled "D:\Windows\system32\cmd.exe". The output of the program is displayed as follows:

```
1 0 0
1 1 1
계속하려면 아무 키나 누르십시오 . . .
```

7. 단축평가

논리연산자 &&와 ||는 피연산자 두 개 중에서 왼쪽 피연산자만으로 논리연산 결과가 결정된다면 오른쪽 피연산자는 평가하지 않는다. 밑에 예제소스가 있다.

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

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

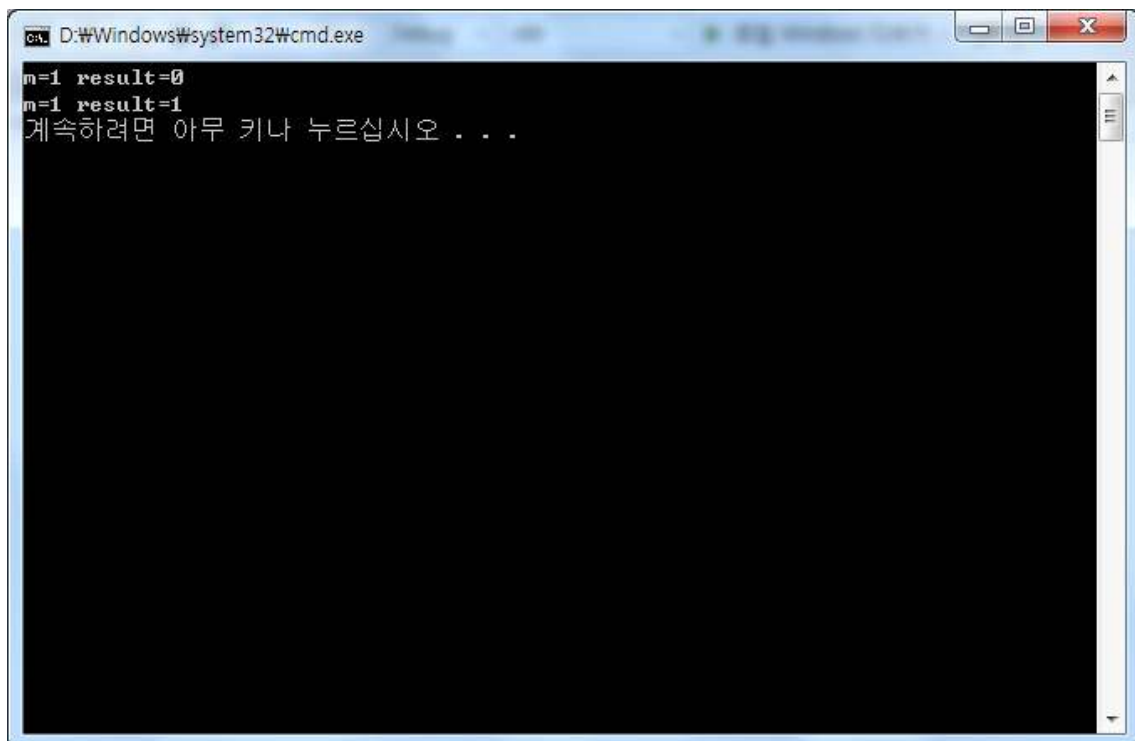
```
    int a = 10, b = 5, m = 1;
    int result;
```

```
    result = (a < b) && (m++ == 1);
    printf("m=%d result=%d\n", m, result);
```

```
    result = (a > b) || (--m == 0);
    printf("m=%d result=%d\n", m, result);
```

```
    return 0;
```

```
}
```



```
cmd. D:\Windows\system32\cmd.exe
m=1 result=0
m=1 result=1
계속하려면 아무 키나 누르십시오 . . .
```

8. 조건 연산자

조건 연산자는 조건에 따라 주어진 피연산자가 결과값이 되는 삼항연산자이다.

즉 $\max a$ 가 b 보다 크면 참은 a 이고 거짓이면 b 라는 뜻이다. $\max a > b ? a : b$ 밑에 예제소스가 있다.

```
#define _CRT_SECURE_NO_WARNINGS
```

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

```
int main(void)
```

```
{
```

```
int a = 0, b = 0;
```

```
printf("두정수를입력>>");
```

```
scanf("%d%d", &a, &b);
```

```
printf("최대값: %d ", (a > b) ? a : b);
```

```
printf("최소값: %d\n ", (a < b) ? a : b);
```

```
printf("절대값: %d ", (a > b) ? a : -a);
```

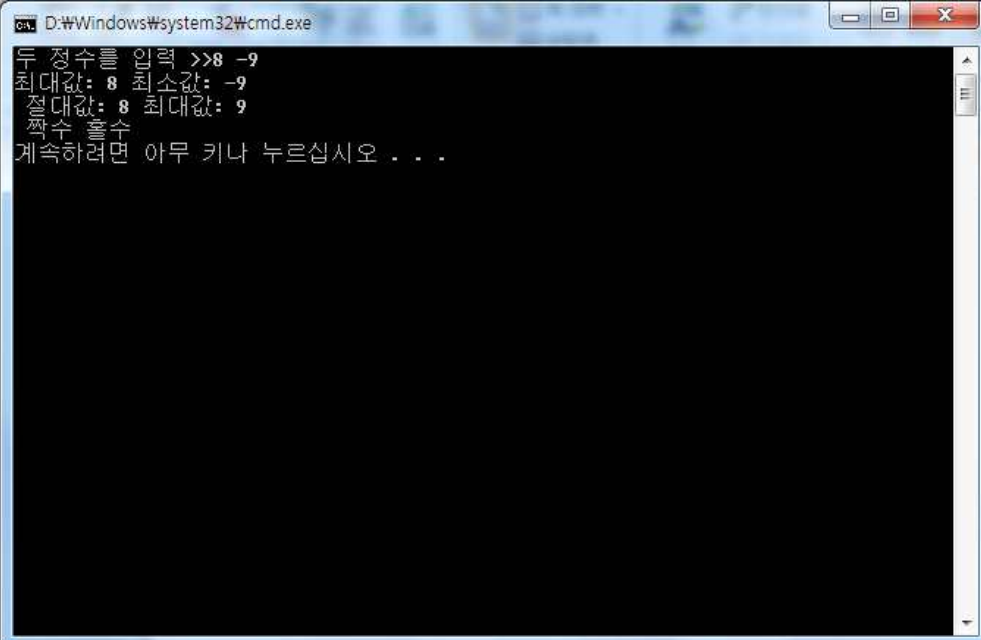
```
printf("최대값: %d\n ", (b > 0) ? b : -b);
```

```
((a % 2) == 0) ? printf("짝수") : printf("홀수");;
```

```
printf("%s\n", ((b % 2) == 0) ? "짝수" : "홀수");
```

```
return 0;
```

```
}
```



```
ca D:\Windows\system32\cmd.exe
두 정수를 입력 >>8 -9
최대값: 8 최소값: -9
절대값: 8 최대값: 9
짝수 홀수
계속하려면 아무 키나 누르십시오 . . .
```

9. 비트논리 연산자

피연산자 정수값을 비트 단위로 논리 연산을 수행하는 연산자로 $\&$, $|$, \wedge , \sim 4가지이다.

연산자	연산자 이름	사용	의미
$\&$	비트 AND	$op1 \ \& \ op2$	비트가 모두 1이면 결과는 1, 아니면 0
$ $	비트 OR	$op1 \ \ op2$	비트가 적어도 하나가 1이면 결과는 1, 아니면 0
\wedge	비트 배타적 OR(XOR)	$op1 \ \wedge \ op2$	비트가 서로 다르면 결과는 1, 같으면 0
\sim	비트 NOT 또는 보수 (COMPLEMENT)	$\sim op1$	비트가 0이면 결과는 1, 0이면 1

밑에 예제 소스가 있다.

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

```
int main(void)
```

```
{
```

```
int x = 127;
```

```
printf("%5d    -> %08x\n", x, x);
```

```
printf("x & 1 -> %08x\n", x & 1);
```

```
printf("x | 1 -> %08x\n", x | 1);
```

```
printf("x ^ 1 -> %08x\n", x ^ 1);
```

```
printf("~(-1) -> %08x\n", ~(-1));
```

```
printf(" ~1    -> %08x\n", ~1);
```

```
return 0;
```

```
}
```

```

D:\Windows\system32\cmd.exe
127    -> 0000007f
x & 1 -> 00000001
x | 1 -> 0000007f
x ^ 1 -> 0000007e
~(-1) -> 00000000
~1    -> ffffffff
계속하려면 아무 키나 누르십시오 . . .
  
```

10. 콤마 연산자

콤마 연산자 ,는 왼쪽과 오른쪽 연산식을 각각 순차적으로 계산하며 결과값은 가장 오른쪽에서 수행한 연산의 결과이다. 예로 $3 + 4, 5 - 10$ 이있다면 결과는 오른쪽 연산을 수행한 -5 가 나온다. 밑에 예제소스가 있다.

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

```
int main(void)
```

```
{
```

```
int a = 100, b = 50, c;
```

```
printf("%d ", sizeof(short));
```

```
printf("%d ", sizeof a);
```

```
printf("%d ", sizeof 3.5F);
```

```
printf("%d\n", sizeof 3.14);
```

```
c = ++a, b++;
```

```
printf("%d %d %d\n", a, b, c);
```

```
c = (3 + a, b * 2);
```

```
printf("%d %d %d\n", a, b, c);
```

```
return 0;
```

```
}
```

A screenshot of a Windows command prompt window titled "D:\Windows\system32\cmd.exe". The window has a black background and white text. The output of the program is displayed as follows:

```
2 4 4 8
101 51 101
101 51 102
계속하려면 아무 키나 누르십시오 . . .
```

6장 조건

1. if문

- 조건식 cond가 0이 아니면 참인 stmt를 실행하고 0이면 거짓인 stmt를 실행하지 않는다.

```
if (cond)          if(grade >= 3.2)
    stmt:          {
    next:           printf("회사에 지원가능");
                    }
                    printf("졸업을 축하합니다");
```

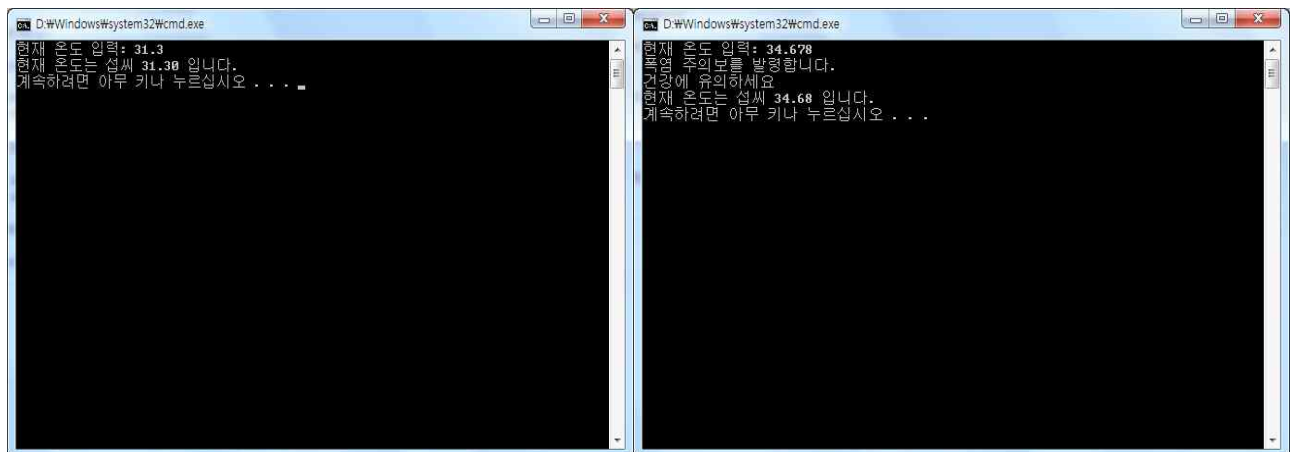
밑에 예제소스가 있다.

```
#define _CRT_SECURE_NO_WARNINGS
```

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

```
int main(void)
{
    double temperature;
    printf("현재온도입력: ");
    scanf("%lf", &temperature);

    if (temperature >= 32.0)
    {
        printf("폭염주의보를발령합니다.\n");
        printf("건강에유의하세요\n");
    }
    printf("현재온도는섭씨%.2f 입니다.\n", temperature);
    return 0;
}
```



2. if else문

조건문 if (cond) stmt1; else stmt2;는 조건 cond를 만족하면 stmt1을 실행하고, 조건 cond를 만족하지 않으면 stmt2를 실행하는 문장이다.

```
if (cond)                if(n % 2 == 0)
    stmt1;                printf("짝수");
else                      else
    stmt2;                printf("홀수");
next;                     printf("입니다.\n");
```

밑에 예제소스가 있다.

```
#define _CRT_SECURE_NO_WARNINGS
```

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

```
int main(void)
```

```
{
```

```
int n;
```

```
printf("정수입력: ");
```

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

```
if (n % 2)
```

```
printf("홀수");
```

```
else
```

```
printf("짝수");
```

```
printf("입니다.\n");
```

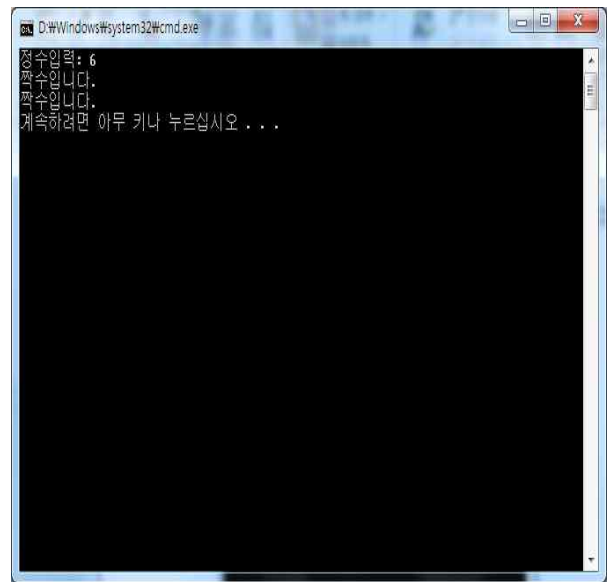
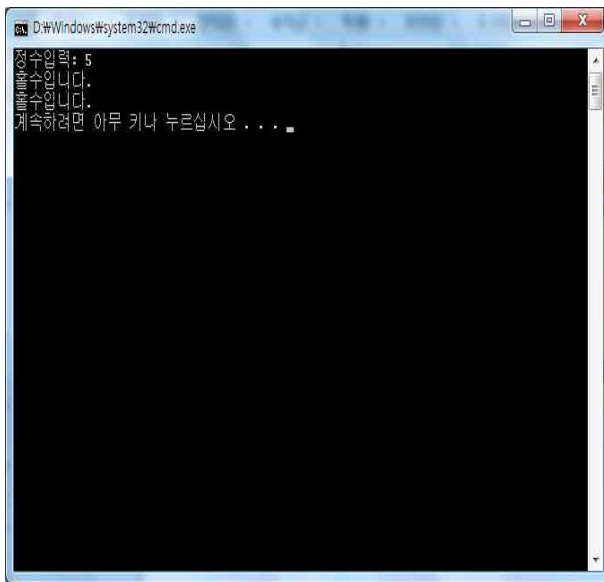
```
//조건연산자이용
```

```
(n % 2) ? printf("홀수") : printf("짝수");
```

```
printf("입니다.\n");
```

```
return 0;
```

```
}
```



3. if else if문

if (cond1)	if (point >= 90)
stmt1;	printf("A\n");
else if (cond2)	else if (point >= 80)
stmt2;	printf("B\n");
else if (cond3)	else if (point >= 70)
stmt3;	printf("C\n");
else	else if (point >= 60)
stmt4;	printf("D\n");
next;	else
	printf("F\n");

밑에 예제소스가 있다.

```
#define _CRT_SECURE_NO_WARNINGS
```

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

```
int main(void)
```

```
{
```

```
double gpa;
```

```
printf("평균평점입력: ");
```

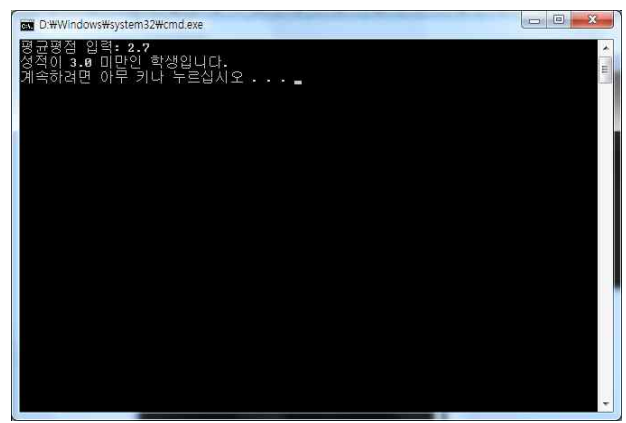
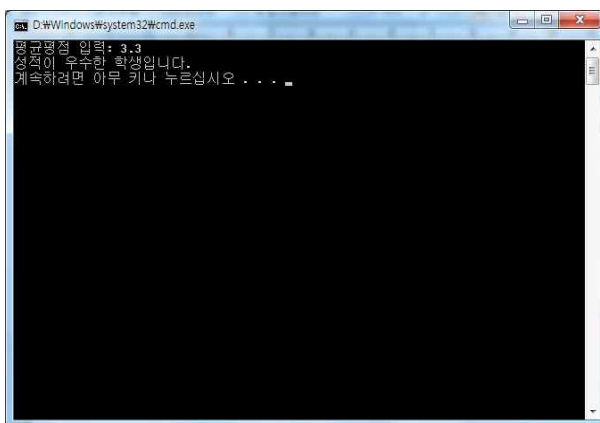
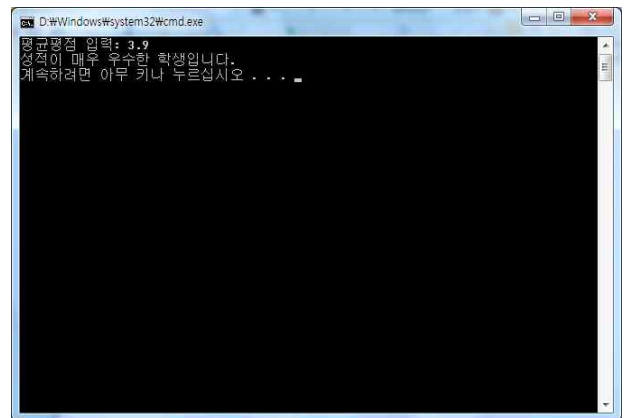
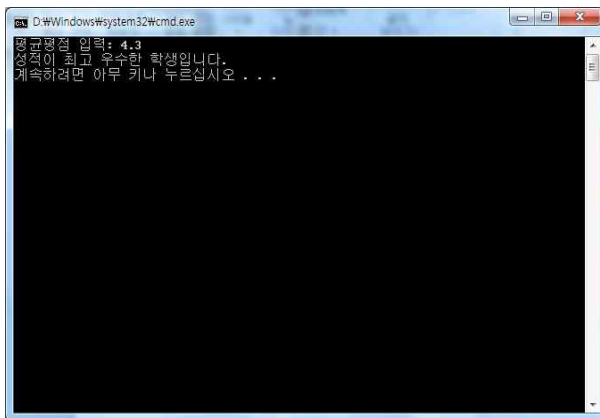
```
scanf("%lf", &gpa);
```

```

if (gpa >= 4.3)
printf("성적이최고우수한학생입니다.\n");
else if (gpa >= 3.8)
printf("성적이매우우수한학생입니다.\n");
else if (gpa >= 3.0)
printf("성적이우수한학생입니다.\n");
else
printf("성적이3.0 미만인학생입니다.\n");

return 0;
}

```



4. switch문

문장 if else가 여러번 계속 반복되는 구분을 좀 더 간략하게 구현할수 있다. switch문은 주어진 연산식이 문자형 또는 정수형이라면 그 값에 따라 case의 상수값과 일치하는 부분의 문장들을 수행하는 선택 구문이다. break을 만나면 switch문은 종료된다.

```
switch (exp) {  
    case 상수1:  
        stmt1:  
        break;  
  
    case 상수2:  
        stmt2:  
        break;  
  
    case 상수3:  
        stmt3:  
        break;  
  
    default:  
        stmt4:  
        break;  
}
```

밑에 예제소스가 있다.

```
#define _CRT_SECURE_NO_WARNINGS  
  
#include <stdio.h>  
  
int main(void)  
{  
    double x, y, result;  
    int op;  
  
    printf("두실수입력: ");  
    scanf("%lf %lf", &x, &y);  
    printf("연산종류번호선택1(+), 2(-), 3(*), 4(/): ");  
    scanf("%d", &op);  
  
    switch (op) {  
        case 1:
```

```

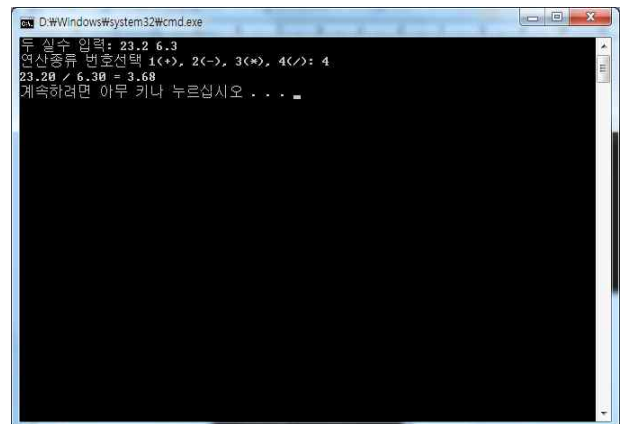
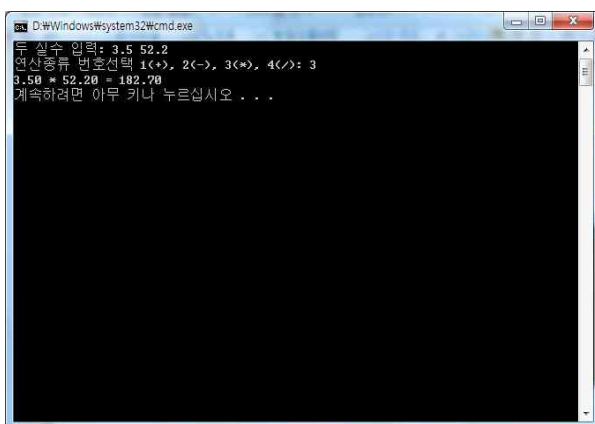
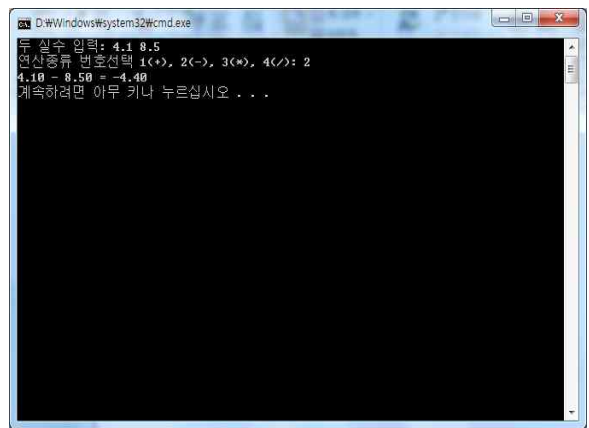
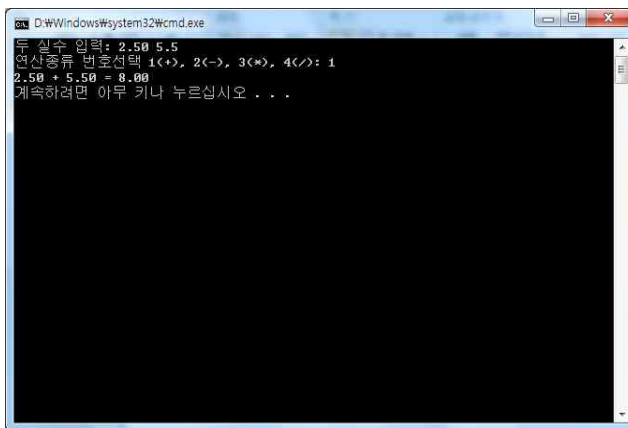
printf("%.2f + %.2f = %.2f\n", x, y, x + y);
break
case 2:
printf("%.2f - %.2f = %.2f\n", x, y, x - y);
break

case 3:
printf("%.2f * %.2f = %.2f\n", x, y, x * y);
break

case 4:
printf("%.2f / %.2f = %.2f\n", x, y, x / y);
break

default:
printf("번호를잘못선택했습니다.\n");
break
}
return 0;
}

```



7장 반복

1. while문

while (cond) stmt;는 반복조건인 cond를 평가하여 0이아니면 참 반복몸체인 stmt를 실행하고 다시 반복조건 cond를 평가하여 while문 종료 시까지 반복한다.

```
int count = 1;
```

```
while (cond)
```

```
    stmt1;
```

```
next;
```

```
while (count <= 3)
```

```
{
```

```
    printf("c 언어 재미있네요!\n");
```

```
    count++;
```

```
};
```

밑에 예제소스가 있다.

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

```
int main(void)
```

```
{
```

```
int count = 1;
```

```
while (count <= 3)
```

```
{
```

```
printf("C 언어재미있네요!\n");
```

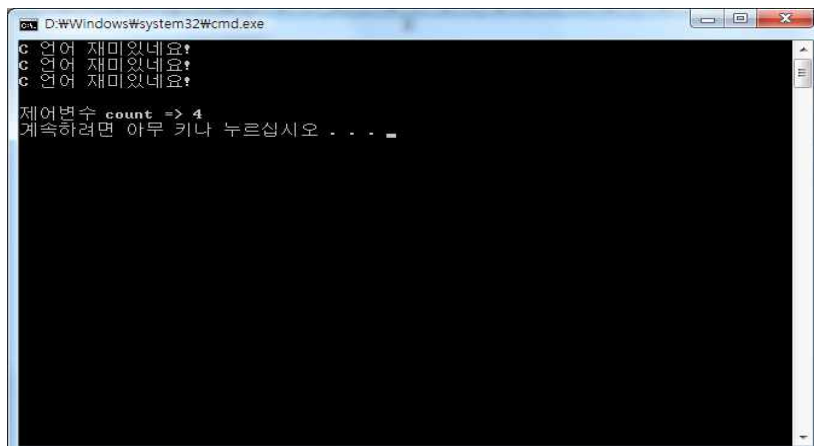
```
count++;
```

```
};
```

```
printf("\n제어변수count => %d\n", count);
```

```
return 0;
```

```
}
```



2. do while문

do while문은 반복문체 수행 후에 반복조건을 검사한다. 즉 do stmt; while (cond)는 가장 먼저 stmt를 실행한 이후 반복조건인 cond를 평가하여 0이 아니면 참 다시 반복문체인 stmt;를 실행하고 0이면 거짓 do while문을 종료한다.

do	do
stmt;	{
while(cond);	printf("양의 정수 또는 0(종료)을 입력: ");
next;	scanf("%d", &input);
	} while (input != 0);

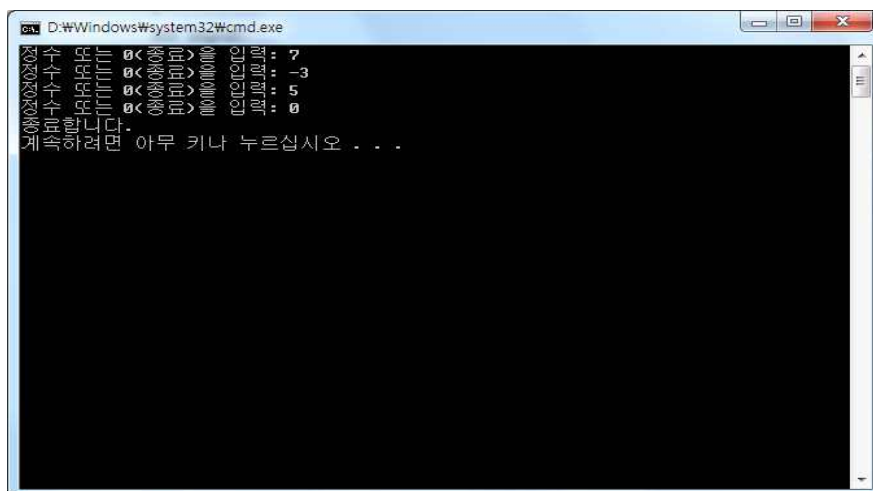
밑에 예제소스가 있다.

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

```
int main(void)
{
    int input;

    do
    {
        printf("정수또는0(종료)을입력: ");
        scanf("%d", &input);
    } while (input != 0);
    puts("종료합니다.");

    return 0;
}
```



3. for문

for(init; cond; inc) stmt;에서 init에서는 초기화가 이루어지고 cond에서는 반복조건 검사를 하며 inc는 증감을 수행한다.

for(init; cond; inc)	for (i=1; i<=10; I++)
	{
stmt;	printf("%3d", i);
next;	}

밑에 예제소스가 있다.

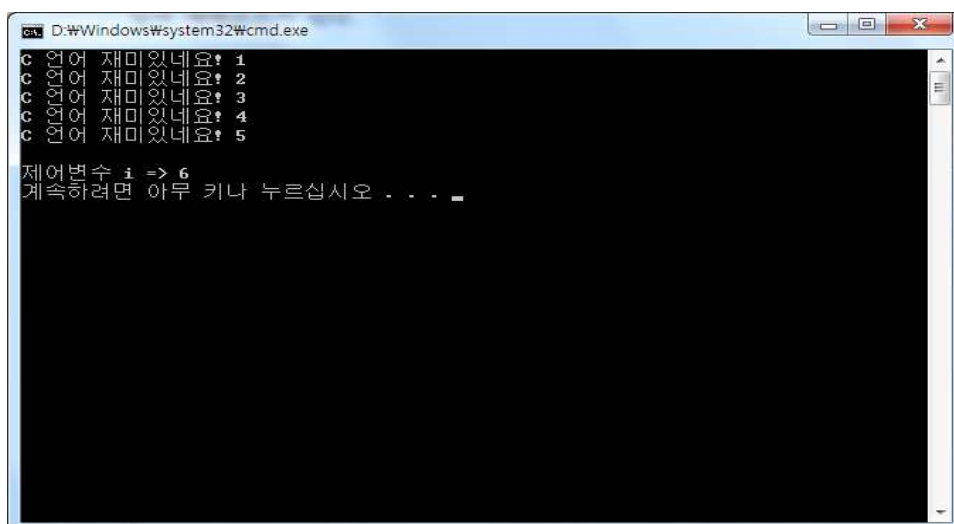
```
#include <stdio.h>
#define MAX 5

int main(void)
{
    int i;

    for (i = 1; i <= MAX i++)
    {
        printf("C 언어재미있네요! %d\n", i);
    }

    printf("\n제어변수i => %d\n", i);

    return 0;
}
```



```
D:\Windows\system32\cmd.exe
C 언어 재미있네요! 1
C 언어 재미있네요! 2
C 언어 재미있네요! 3
C 언어 재미있네요! 4
C 언어 재미있네요! 5

제어변수 i => 6
계속하려면 아무 키나 누르십시오 . . .
```


4. 분기문

분기문은 정해진 부분으로 바로 실행을 이동(jump)하는 기능을 수행한다. c가 지원하는 분기문으로는 break, continue, goto, return 문이 있다.

첫째 break문은 반복내부에서 반복을 종료할 때 사용한다.

for(; ;)	while(...)	do
{	{	{
break;	break;	...
}	}	...
next;	next;	} while(...);
		next;

둘째 continue문은 반복의 시작으로 이동하여 다음 반복을 실행하는 문장이다.

```
while ( cond 1 )
{
    continue;

for (init; cond; inc)
{
    continue;

stmt1;
stmt2;
}
}
```

셋째 goto문은 레이블이 위치한 다음 문장으로 실행순서를 이동하는 문장이다.

```
loop:
printf("%3d", count); //문장 goto문은 무조건 lable이 있는곳으로 이동하여 실행한다.
if (++count <= 10)
goto loop;
printf("프로그램을 종료합니다.");
```

5. 중첩된 for문

반복문 내부에 반복문이 또 있는 구문을 중첩된 반복문(nested loop)이라 한다.

밑에 예제소스가 있다.

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

```
int main(void)
```

```
{
```

```
int m, n;
```

```
for (m = 1; m <= 5; m++)
```

```
{
```

```
printf("m = %-2d\n", m);
```

```
for (n = 1; n <= 7; n++)
```

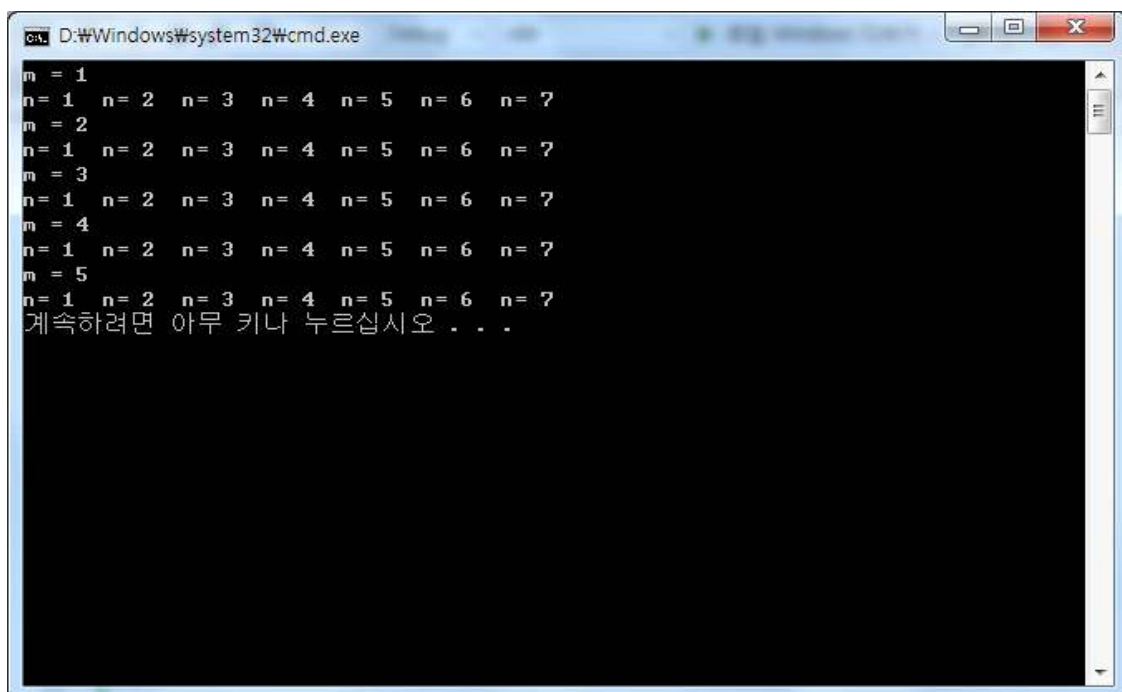
```
printf("n= %-3d", n);
```

```
puts("");
```

```
}
```

```
return 0;
```

```
}
```



```
C:\> D:\Windows\system32\cmd.exe
m = 1
n = 1 n = 2 n = 3 n = 4 n = 5 n = 6 n = 7
m = 2
n = 1 n = 2 n = 3 n = 4 n = 5 n = 6 n = 7
m = 3
n = 1 n = 2 n = 3 n = 4 n = 5 n = 6 n = 7
m = 4
n = 1 n = 2 n = 3 n = 4 n = 5 n = 6 n = 7
m = 5
n = 1 n = 2 n = 3 n = 4 n = 5 n = 6 n = 7
계속하려면 아무 키나 누르십시오 . . .
```

8장 포인터 기초

1. 메모리주소와 주소연산자 &

- 주소개념은 메모리공간인 8비트인 1바이트마다 고유한 주소가 있다.
메모리주소는 0부터 바이트마다 1씩 증가한다. 메모리 주소는 저장 장소인 변수이름과 함께 기억장소를 참조하는 또 다른 방법이다.
- 주소연산자는 &를 사용하며 피연산자인 변수의 메모리 주소를 반환하는 주소연산자이다.
밑에 예제소스가 있다.

```
#define _CRT_SECURE_NO_WARNINGS
```

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

```
int main(void)
```

```
{
```

```
int input;
```

```
printf("정수입력: ");
```

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

```
printf("입력값: %d\n", input);
```

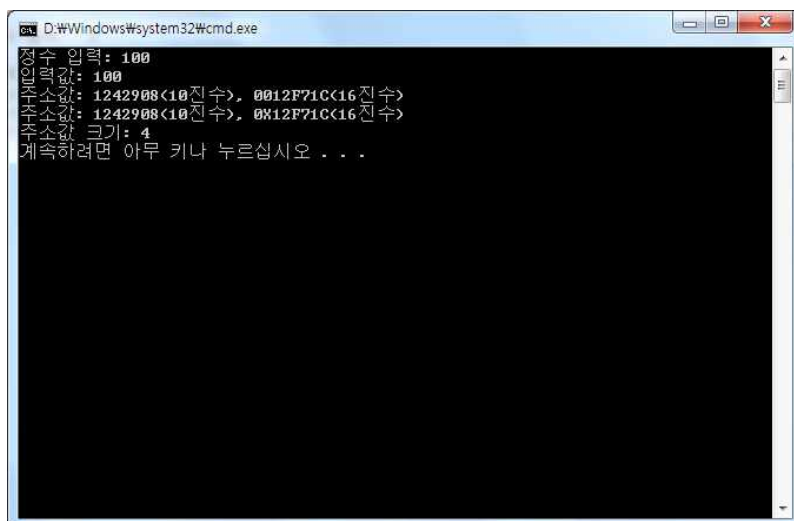
```
printf("주소값: %u(10진수), %p(16진수)\n", (int)&input, &input);
```

```
printf("주소값: %d(10진수), %#X(16진수)\n", (unsigned)&input, (int)&input);
```

```
printf("주소값 크기: %d\n", sizeof(&input));
```

```
return 0;
```

```
}
```



```
D:\Windows\system32\cmd.exe
정수 입력: 100
입력값: 100
주소값: 1242908<10진수>, 0012F71C<16진수>
주소값: 1242908<10진수>, 0X12F71C<16진수>
주소값 크기: 4
계속하려면 아무 키나 누르십시오 . . .
```

2. 포인터 변수 선언

포인터 변수 선언은 자료형과 포인터 변수 이름 사이에 연산자 *를 삽입한다. 즉 다음변수 선언에서 `ptrint`, `ptrshort`, `ptrchar`, `ptrdouble`은 모두 포인터 변수이며 간단히 포인터라고 한다.

`int data = 100;` <- 변수 `data`는 정수 `int`를 저장하는 일반변수

`int *ptrint;` <- 변수 `ptrint`는 정수 `int`를 저장하는 일반변수의 주소를 저장하는 포인터 변수

`ptrint = &data;` <- 포인터 `ptrint`는 이제 'data를 가리킨다'

포인터의 변수는 변수종류에 상관없이 크기가 모두 4바이트이다.

밑에 예제소스가 있다.

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

```
int main(void)
```

```
{
```

```
int data = 100;
```

```
int *ptrint;
```

```
ptrint = &data;
```

```
printf("변수명주소값저장값\n");
```

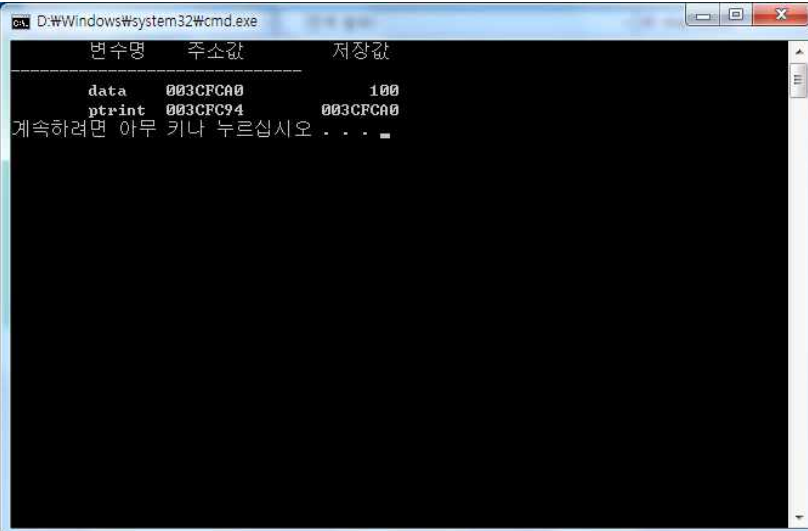
```
printf("-----\n");
```

```
printf("data%p%8d\n", &data, data);
```

```
printf("ptrint%p%p\n", &ptrint, ptrint);
```

```
return 0;
```

```
}
```



```
D:\Windows\system32\cmd.exe
변수명  주소값  저장값
-----
data    003CFCa0  100
ptrint  003CFC94  003CFCa0
계속하려면 아무 키나 누르십시오 . . .
```

3. 다양한 포인터 변수 선언과 간접 연산자*

여러 포인터 변수선언

여러개의 포인터 변수를 한번에 선언하기 위해서는 다음과 같이 콤마 이후에 변수마다 *를 앞에 기술해야 한다.

```
int *ptr1, *ptr2, *ptr3; //ptr1, ptr2, ptr3 모두 int형 포인터임
```

```
int *ptr1, ptr2, ptr3; //ptr1은 int형 포인터나 ptr2와 ptr3은 int형 변수임
```

또한 초기값을 대입하지 않으면 쓰레기값이 들어가므로 포인터 변수에 지정할 특별한 초기값이 없는 경우에 0번 주소값인 NULL로 초기값을 지정한다.

```
int *ptr = NULL;
```

마지막으로 자료유형(void *)는 아직 유보된 포인터이므로 모든 유형의 포인터 값을 저장할 수 있는 포인터 형이다.

```
#define NULL ((void *)0)
```

밑에 예제소스가 있다.

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

```
int main(void)
```

```
{
```

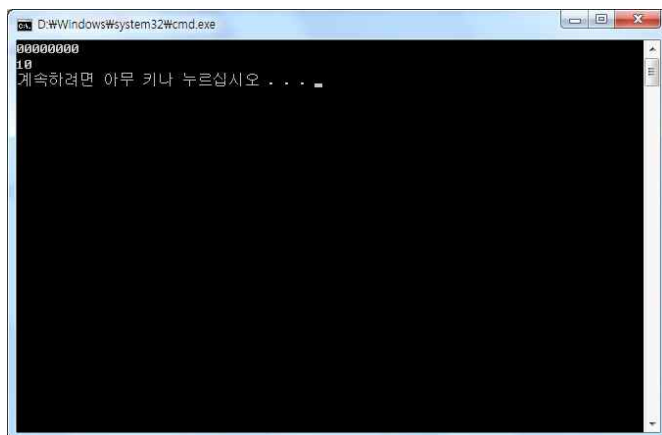
```
int *ptr1, *ptr2, data = 10;
```

```
ptr1 = NULL
```

```
printf("%p\n", ptr1);
```

```
printf("%d\n", data);
```

```
return 0;
```



4. 간접연산자 *

포인터 변수가 갖는 주소로 그 주소의 원래 변수를 참조하는 것이다.

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

```
int *p;
```

```
printf("간접참조 출력: %d \n", *p);
```

```
*p = 200;
```

즉 포인터p가 가리키는 변수가 data라면 *p는 변수 data를 의미한다.

밑에 예제소스가 있다.

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

```
int main(void)
```

```
{
```

```
int data = 100;
```

```
char ch = 'A'
```

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

```
char *ptrchar = &ch;
```

```
printf("간접참조출력: %d %c\n", *p, *ptrchar);
```

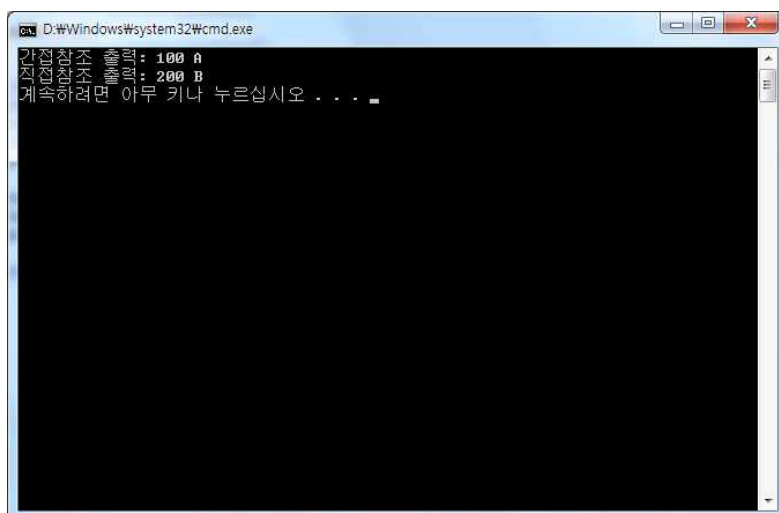
```
*p = 200;
```

```
*ptrchar = 'B'
```

```
printf("직접참조출력: %d %c\n", data, ch);
```

```
return 0;
```

```
}
```



5. 명시적 형변환

포인터변수는 동일한 자료형끼리만 대입이 가능하다. 만일 대입문에서 포인터의 자료형이 다르면 경고가 발생한다. 포인터 변수는 자동으로 형변환이 불가능하므로 사용자가 직접 변환해주는 것이 명시적 형변환이다.

```
int value = 0x61626364;
int *pi = &value;
char *pc = &value;
```

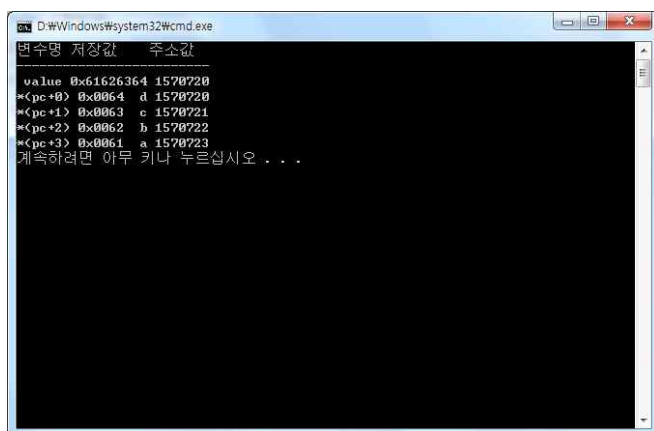
밑에 예제소스가 있다.

```
#include <stdio.h>

int main(void)
{
    int value = 0x61626364;
    int *pi = &value;
    char *pc = (char *)&value;

    printf("변수명 저장값 주소값\n");
    printf("-----\n");
    printf(" value %0#x %u\n", value, pi);

    for (int i = 0; i <= 3; i++)
    {
        char ch = *(pc + i);
        printf("(pc+%d) %0#6x %2c %u\n", i, ch, ch, pc + i);
    }
    return 0;
}
```



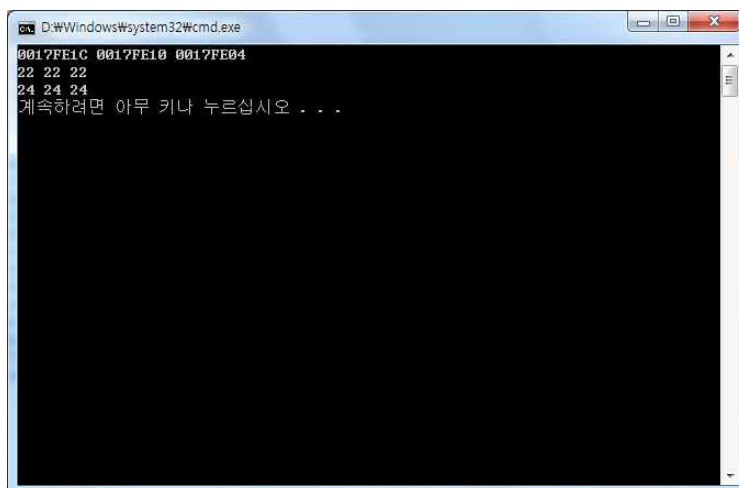
6. 이중포인터

포인터 변수의 주소값을 갖는 변수를 이중 포인터라 한다.

```
int i = 20;  
int *pi = &i;  
int **dpi = &pi;
```

밑에 예제소스가 있다.

```
#include <stdio.h>  
  
int main(void)  
{  
    int i = 20;  
    int *pi = &i;  
    int **dpi = &pi;  
  
    printf("%p %p %p\n", &i, &pi, &dpi);  
  
    *pi = i + 2;  
    printf("%d %d %d\n", i, *pi, **dpi);  
  
    **dpi = *pi + 2;  
    printf("%d %d %d\n", i, *pi, **dpi);  
  
    return 0;  
}
```



7. 간접 연산자와 증감 연산자 활용

간접연산자 *는 증감 연산자 ++, --와 함께 사용하는 경우가 많다. 밑에 표로 우선순위가 있다.

우선순위	단항연산자	설명	결합성(계산방향)
1	a++ a--	후위증가, 후위감소	->(좌에서 우로)
2	++a --a & * 주소 간접 또는 역참조	전위증가, 전위감소 주소 간접 또는 역참조	<-(우에서 좌로)

표를 정리하면

*p++는 *(p++)으로(*p)++과 다르다.

+++p와 ++(*p)는 같다.

***p는 *(++p)는 같다.

밑에 예제소스가 있다.

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

```
int main(void)
```

```
{
```

```
int i;
```

```
int *pi = &i;
```

```
int **dpi = &pi;
```

```
*pi = 5;
```

```
*pi += 1;
```

```
printf("%d\n", i);
```

```
printf("%d\n", (*pi)++);
```

```
printf("%d\n", *pi);
```

```
*pi = 10;
```

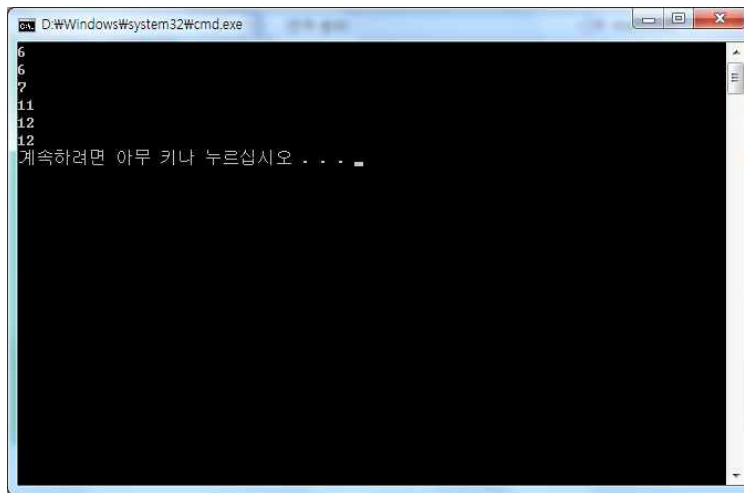
```
printf("%d\n", +++*pi);
```

```
printf("%d\n", +***dpi);
```

```
printf("%d\n", i);
```

```
return 0;
```

```
}
```



8. 포인터 상수

키워드 `const`를 이용하는 변수 선언은 변수를 상수로 만들 듯이 포인터 변수도 포인터 상수로 만들 수 있다. 2가지 방식이 있다.

```
int I = 10, j = 20;
```

1. `const int *pi = &i;`

2. `int const *pi = &i;`

1번과 2번은 같은 방식이다.

3. `int* const pi = &i;`

밑에 예제소스가 있다.

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

```
int main(void)
```

```
{
```

```
int i = 10, j = 20;
```

```
const int *p = &i;
```

```
p = &j;
```

```
printf("%d\n", *p);
```

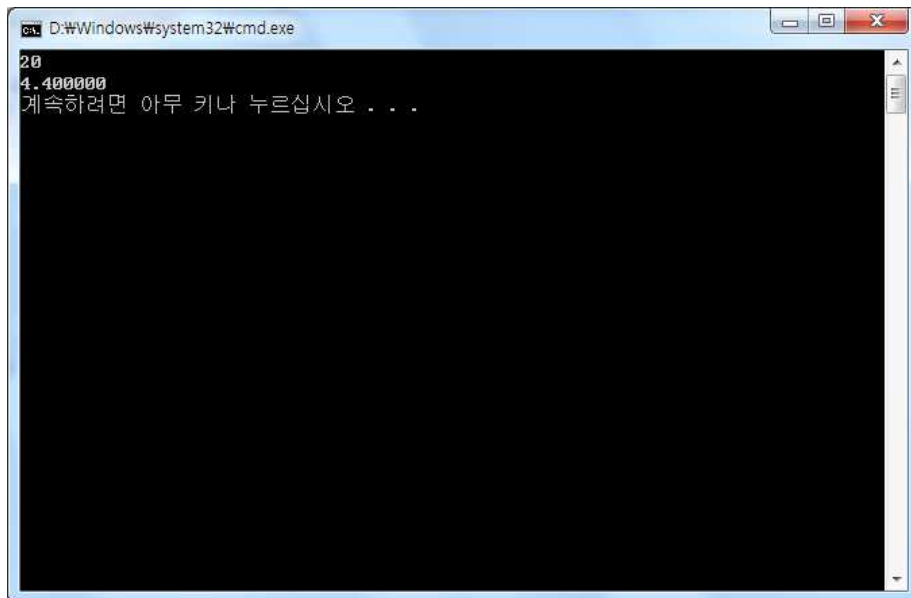
```
double d = 7.8, e = 2.7;
```

```
double * const pd = &d;
```

```
*pd = 4.4;
```

```
printf("%f\n", *pd);
```

```
return 0;
}
```



9장 배열

1. 배열정의

배열은 동일한 자료 유형이 여러 개 필요한 경우에 유용한 자료 구조이다. 즉 배열은 한 자료유형의 저장공간인 원소를 동일한 크기로 지정된 배열크기만큼 확보한 연속된 저장공간이다.

2. 배열선언

배열선언은 `int data[10];`과 같이 원소자료유형 배열이름[배열크기];로 한다. 배열 선언시 초기값 지정이 없다면 반드시 배열크기는 양의정수로 명시되어야 한다. 하지만 변수와 const상수로는 배열의 크기를 지정할 수 없다.

이렇게 선언한다.

원소자료형 배열이름[배열크기];

int score[10];

3. 배열원소 접근

배열선언 후 배열원소에 접근하려면 배열이름 뒤에 대괄호 사이 첨자(index)를 이용한다. 배열에서 유효한 첨자의 범위는 0부터 (배열크기-1)까지 이며 첨자의 유효 범위를 벗어나

원소를 참조하면 문법오류 없이 실행오류가 발생한다.

배열선언 시 대괄호 안의 수는 배열 크기이다. 그러나 선언 이후 대괄호 안의 수는 원소를 참조하는 번호인 첨자라는 것을 명심하자.

밑에 for문을 이용한 배열원소 일괄출력 소스가 있다.

```
#include <stdio.h>

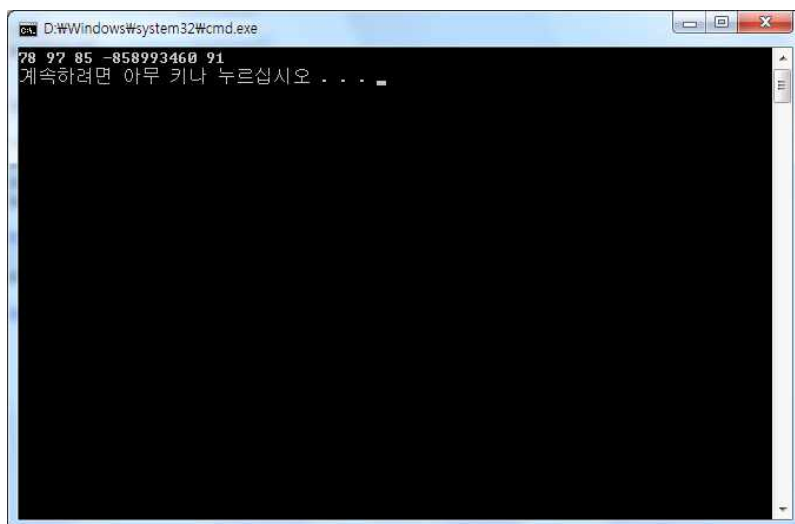
#define SIZE 5
int main(void)
{

    int score[SIZE];

    score[0] = 78;
    score[1] = 97;
    score[2] = 85;
    score[4] = 91;

    for (int i = 0; i < SIZE; i++)
        printf("%d ", score[i]);
    printf("\n");

    return 0;
}
```



4. 배열 초기화

배열크기가 크면 원소값을 일일이 저장하는 일도 쉽지 않기 때문에 배열을 선언하면서 원소값을 손쉽게 저장하는 배열선언 초기화 방법을 제공한다. 배열선언 초기화 구문은 배열선언을 하면서 대입연산자를 이용해 중괄호 사이에 여러 원소값을 쉼표로 구분하여 기술하는 방법이다.

이런식으로 초기화한다.

```
int grade[4] = {98, 88, 92, 95};
```

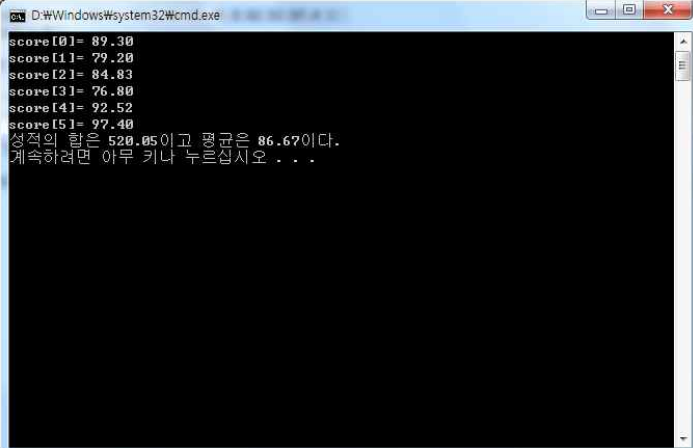
밑에 예제소스가 있다.

```
#include <stdio.h>
#define SIZE 6

int main(void)
{
    double score[] = { 89.3, 79.2, 84.83, 76.8, 92.52, 97.4 };
    double sum = 0;

    for (int i = 0; i < SIZE i++)
    {
        sum += score[i];
        printf("score[%d]= %.2f\n", i, score[i]);
    }
    printf("성적의 합은 %.2f이고 평균은 %.2f이다.\n", sum, sum / SIZE);

    return 0;
}
```



```
D:\Windows\system32\cmd.exe
score[0]= 89.30
score[1]= 79.20
score[2]= 84.83
score[3]= 76.80
score[4]= 92.52
score[5]= 97.40
성적의 합은 520.05이고 평균은 86.67이다.
계속하려면 아무 키나 누르십시오 . . .
```

5. 이차원 배열

테이블 형태의 구조를 나타낼 수 있으므로 행과 열의 구조로 표현한다.

중간고사	기말고사
95	85
90	88
86	90
88	78

6. 이차원 배열선언

원소자료형 배열이름[배열행크기][배열열크기];

int score[RSIZE][CSIZE];

밑에 예제소스가 있다.

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

```
#define ROWSIZE 2
```

```
#define COLSIZE 3
```

```
int main(void)
```

```
{
```

```
int td[ROWSIZE][COLSIZE];
```

```
td[0][0] = 1; td[0][1] = 2; td[0][2] = 3;
```

```
td[1][0] = 4; td[1][1] = 5; td[1][2] = 6;
```

```
printf("반복문for를이용하여출력\n");
```

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

```
{
```

```
for (int j = 0; j < COLSIZE j++)
```

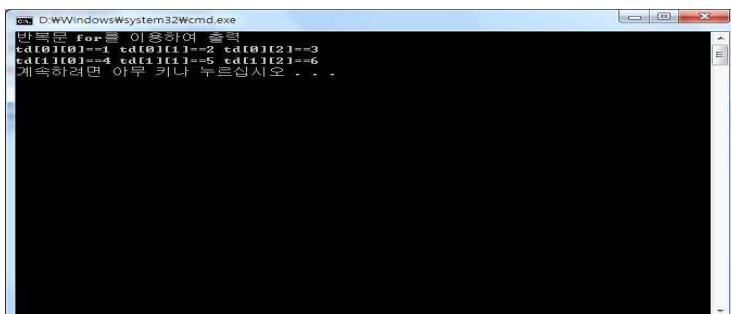
```
printf("td[%d][%d]==%d ", i, j, td[i][j]);
```

```
printf("\n");
```

```
}
```

```
return 0;
```

```
}
```



7. 이차원배열 초기화

이차원 배열을 선언하면서 초기값을 지정하는 방법은 중괄호를 중첩되게 이용하는 방법과 일차원 배열 같이 하나의 중괄호를 사용하는 방법이 있다.

```
int score[2][3] = {{30, 44, 67},{87, 43, 56}};
```

밑에 예제소스가 있다.

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

```
#define ROWSIZE 2
```

```
#define COLSIZE 3
```

```
int main(void)
```

```
{
```

```
int td[][3] = { { 1 }, { 1, 2, 3 } };
```

```
printf("반복문for를이용하여출력\n");
```

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

```
{
```

```
for (int j = 0; j < COLSIZE j++)
```

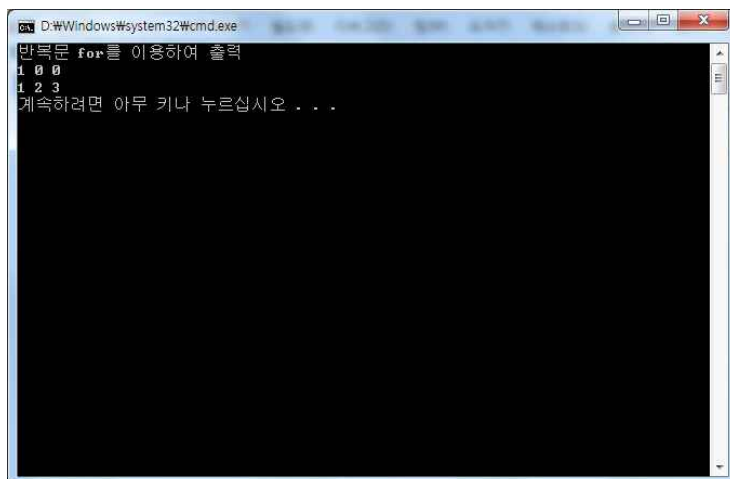
```
printf("%d ", td[i][j]);
```

```
printf("\n");
```

```
}
```

```
return 0;
```

```
}
```



8. 3차원 배열선언

```
int thread[2][2][3]; //2*2*3=12개 원소의 삼차원 배열
thread[0][0][0] = 1; //첫 번째 원소
thread[0][0][1] = 1; // 두 번째 원소
thread[0][0][2] = 1; // 세 번째 원소
thread[0][0][0] = 1; // 네 번째 원소
...
thread[1][1][2] = 1; // 열두 번째(마지막) 원소
```

9. 3차원 배열초기화

[강좌 1]	중간	기말	int score[2][4][2] = {
학생1	95	85	{ { 95, 85 },
학생2	85	83	{ 85, 83 },
학생3	92	75	{ 92, 75 },
학생4	90	88	{ 90, 88 }},

[강좌 2]	중간	기말	{ { 88, 77 },
학생1	88	77	{ 72, 95 },
학생2	72	95	{ 88, 92 },
학생3	88	92	{ 93, 83 },
학생4	93	83	};

밑에 예제소스가 있다.

```
#include <stdio.h>

#define ROWSIZE 4
#define COLSIZE 2
int main(void)
{
    int score[][ROWSIZE][COLSIZE] = {
        { { 95, 85 },
          { 85, 83 },
          { 92, 75 },
          { 90, 88 } },
        { { 88, 77 },
          { 72, 95 },
          { 88, 92 },
          { 93, 83 } }
    };
}
```



```

for (int i = 0; i < 2; i++)
{
    if (i == 0) printf("[강좌1]");
    else printf("[강좌2]");
    printf("%11s%7s\n", "중간", "기말");

    for (int j = 0; j < ROWSIZE j++)
    {
        printf("%10s%2d", "학생", j + 1);
        for (int k = 0; k < COLSIZE k++)
            printf("%6d ", score[i][j][k]);
        printf("\n");
    }
    printf("\n");
}
return 0;
}

```

```

cmd. D:\Windows\system32\cmd.exe
[강좌 1]
학생 1      95      85
학생 2      85      83
학생 3      92      75
학생 4      90      88

[강좌 2]
학생 1      88      77
학생 2      72      95
학생 3      88      92
학생 4      93      83

계속하려면 아무 키나 누르십시오 . . .

```

10. 포인터 배열

일반 변수의 배열이 있듯이 포인터 배열이란 주소값을 저장하는 포인터를 배열원소로 하는 배열이다.

```
int a = 5, b = 7, c = 9;
int *pa[3];
pa[0] = &a;    pa[1] = &b;    pa[2] = &c;
```

11. 포인터 배열 선언

자료형 *변수이름[배열크기];

```
int *pary[5];
char *ptr[4];
float a, b, c;
double *dary[5] = {NULL};
float *ptr[3] = {&a, &b, &c};
```

밑에 예제소스가 있다.

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

```
#define SIZE 3
int main(void)
{
    int *pary[SIZE] = { NULL };
    int a = 10, b = 20, c = 30;
```

```
pary[0] = &a;
pary[1] = &b;
pary[2] = &c;
```

```
for (int i = 0; i < SIZE i++)
    printf("*pary[%d] = %d\n", i, *pary[i]);
```

```
for (int i = 0; i<SIZE i++)
{
    scanf("%d", pary[i]);
    printf("%d, %d, %d\n", a, b, c);
}
return 0;
}
```

```
ca. D:\Windows\system32\cmd.exe
*par1[0] = 10
*par1[1] = 20
*par1[2] = 30
21
21, 20, 30
15
21, 15, 30
8
21, 15, 8
계속하려면 아무 키나 누르십시오 . . .
```

11. 배열포인터 선언

일차원 배열과 이차원 배열 포인터의 변수선언

원소자료형 *변수이름;

변수이름 = 배열이름;

또는

원소자료형 *변수이름 = 배열이름;

int a[] = {8, 2, 8, 1, 3};

int *p = a;

원소자료형 (*변수이름)[배열열크기];

변수이름 = 배열이름;

또는

원소자료형 (*변수이름)[배열열크기] = 배열이름;

int ary[][4] = {5, 7, 6, 2, 7, 8, 1, 3};

int (*ptr)[4] = ary;

밑에 예제소스가 있다.

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

```
int main(void)
```

```
{
```

```
int a[] = { 8, 2, 8, 1, 3 };
```

```
int *p = a;
```

```
printf("%2d, %2d\n", *(p + 1), *(p + 4));
```

```
printf("%2d, %2d\n", p[1], p[4]);
```

```
printf("sizeof(a) = %d, sizeof(p)=%d\n", sizeof(a), sizeof(p));
```

```
printf("%2d\n", **p);
```

```
int ary[4] = { 5, 7, 6, 2, 7, 8, 1, 3 };
```

```
int(*ptr)[4] = ary;
```

```
printf("%2d, %2d\n", **ary, **ptr++);
```

```
printf("%2d, %2d\n", *(ary + 1), *(ptr++));
```

```
ptr = ary;
```

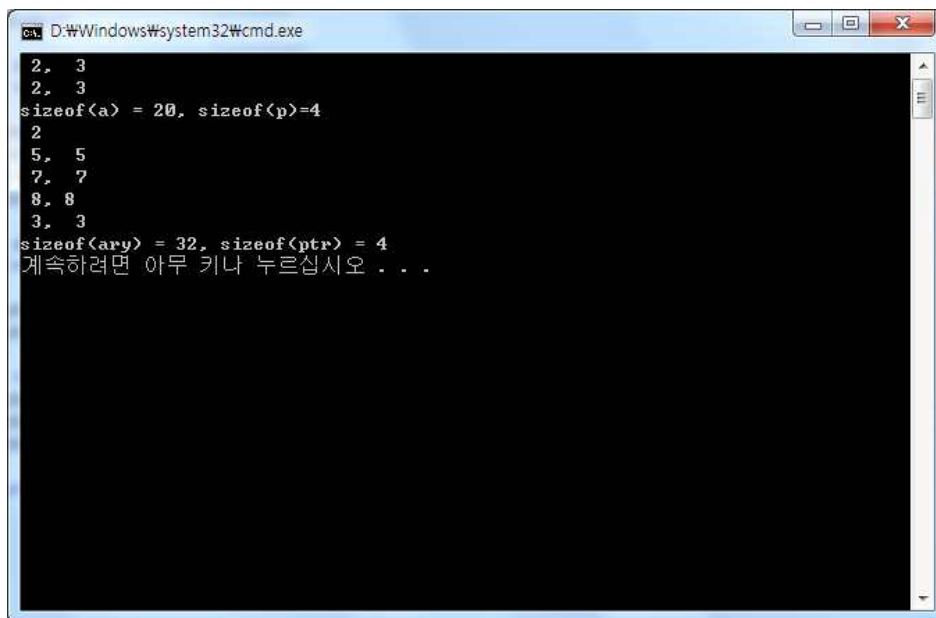
```
printf("%2d,%2d\n", *(ary[1] + 1), *(ptr[1] + 1));
```

```
printf("%2d, %2d\n", (*(ary + 1) + 3), (*(ptr + 1) + 3));
```

```
printf("sizeof(ary) = %d, sizeof(ptr) = %d\n", sizeof(ary), sizeof(ptr));
```

```
return 0;
```

```
}
```



```
D:\Windows\system32\cmd.exe
2, 3
2, 3
sizeof(a) = 20, sizeof(p)=4
2
5, 5
7, 7
8, 8
3, 3
sizeof(ary) = 32, sizeof(ptr) = 4
계속하려면 아무 키나 누르십시오 . . .
```

12. 배열 크기 연산

배열 크기 계산방법

연산자 sizeof를 이용한다. (sizeof(배열이름) / sizeof(배열원소))의 결과는 배열크기이다.

sizeof(배열이름)은 배열의 전체 공간의 바이트 수이다.

sizeof(배열원소)는 배열원소 하나의 바이트 수이다.

밑에 예제소스가 있다.

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

```
int main(void)
```

```
{
```

```
int data[] = { 3, 4, 5, 7, 9 };
```

```
printf("%d %d\n", sizeof(data), sizeof(data[0]));
```

```
printf("일차원배열: 배열크기== %d\n", sizeof(data) / sizeof(data[0]));
```

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

```
printf("%d %d %d\n", sizeof(x), sizeof(x[0]), sizeof(x[1]), sizeof(x[0][0]));
```

```
int rowsize = sizeof(x) / sizeof(x[0]);
```

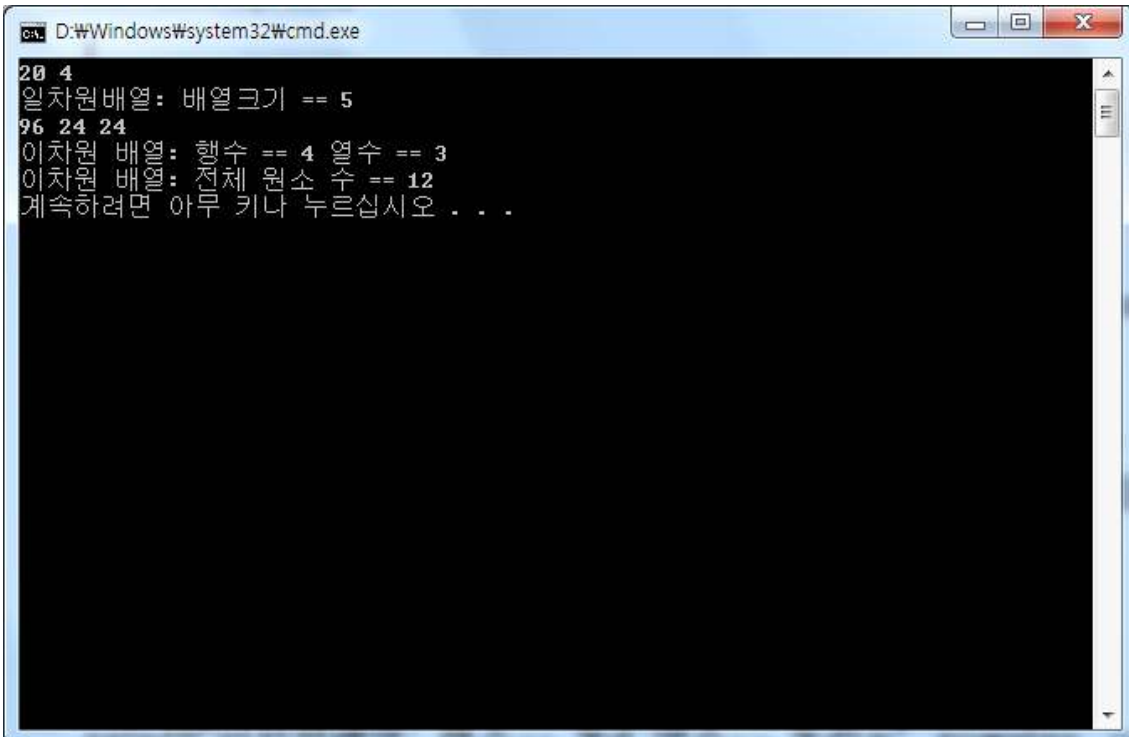
```
int colsize = sizeof(x[0]) / sizeof(x[0][0]);
```

```
printf("이차원배열: 행수== %d 열수== %d\n", rowsize, colsize);
```

```
printf("이차원배열: 전체원소수== %d\n", sizeof(x) / sizeof(x[0][0]));
```

```
return 0;
```

```
}
```



The screenshot shows a Windows command prompt window titled "D:\Windows\system32\cmd.exe". The output of the program is displayed as follows:

```
20 4
일차원배열: 배열크기 == 5
96 24 24
이차원 배열: 행수 == 4 열수 == 3
이차원 배열: 전체 원소 수 == 12
계속하려면 아무 키나 누르십시오 . . .
```

10장 함수기초

1. 함수개념

특정한 작업을 처리하도록 작성한 프로그램 단위를 함수라고 한다.

함수는 필요한 입력을 받아 원하는 어떤 기능을 수행한 후 결과를 반환하는 프로그램 단위이다.

함수는 라이브러리 함수와 사용자정의 함수로 구분한다.

또한 절차적 프로그래밍이다.

2. 함수정의

함수정의는 함수머리와 함수몸체로 구성된다.

함수헤더 { 반환형 함수이름 (매개변수 목록)	int add2(int a, int b)
{	{
...	int sum = a + b;
함수몸체 { 여러문장들;	return (sum);
return(반환연산식);	}
}	

3. 반환형과 return

C의 자료형 char, short, int, long, float, double 등과 같은 반환형이 있으며 자료형으로 사용가능하다. 만일 함수가 반환값이 없다면 반환형으로 void를 기술한다.

int findMin2(int x, int y)	void printMin(int a, int b)
{	{
int min= x < y ? x : y;	int min = a < b ? a : b;
return (min);	printf("%n",min);
}	return; //생략가능
	}

return문장은 함수에서 반환값을 전달하는 목적과 함께 함수의 작업종료를 알리는 문장이다.

4. 함수선언과 함수호출

함수실행

정의된 함수를 실행하려면 프로그램 실행중에 함수호출이 필요하다.

int a = 3, b=5;	int findMax2(int x, int y)
int max = findMax2(a,b);	{
int sum = add2(a,b);	int max = x > y ? x : y;
printMin(2,5);	return (max);
	}

밑에 예제소스가 있다.

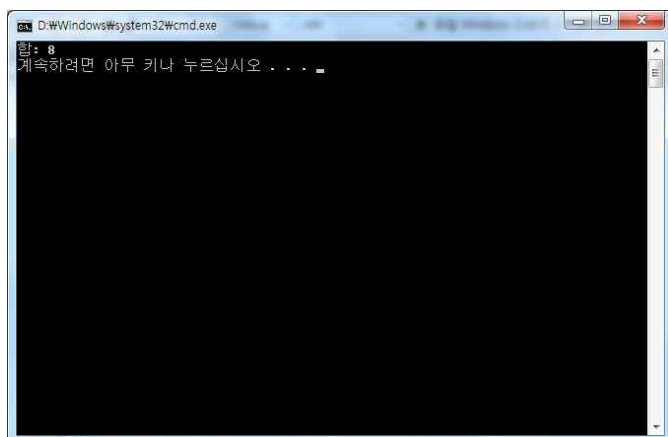
```
#include <stdio.h>
int main(void)
{
    int a = 3, b = 5;
    int add2(int a, int b);

    int sum = add2(a, b);
    printf("합: %d\n", sum);

    return 0;
}

int add2(int a, int b)
{
    int sum = a + b
    return (sum);
}

int findMin2(int x, int y)
{
    int min = x < y ? x : y
    return (min);
}
```



5. 매개변수 정의

함수 매개변수는 함수를 호출하는 부분에서 함수몸체로 값을 전달할 목적으로 이용된다. 함수정의에서 매개변수는 필요한 경우 자료형과 변수명의 목록으로 나타내며 필요없으면 키워드 void를 기술한다.

int sum = add2(a,b); //add2가 int sum을 호출한다.

밑에 예제소스가 있다.

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

```
int add2(int a, int b);
```

```
int findMax2(int, int);
```

```
void printMin(int, int);
```

```
int main(void)
```

```
{
```

```
int a = 3, b = 5;
```

```
int max = findMax2(a, b);
```

```
printf("최대: %d\n", max);
```

```
printf("합: %d\n", add2(a, b));
```

```
printMin(2, 5);
```

```
return 0;
```

```
}
```

```
int add2(int a, int b)
```

```
{
```

```
int sum = a + b
```

```
return (sum);
```

```
}
```

```
int findMax2(int a, int b)
```

```
{
```

```
int max = a > b ? a : b
```

```
return max;
```

```
}
```

```
int findMin2(int x, int y)
```

```
{
```

```
int min = x < y ? x : y
```



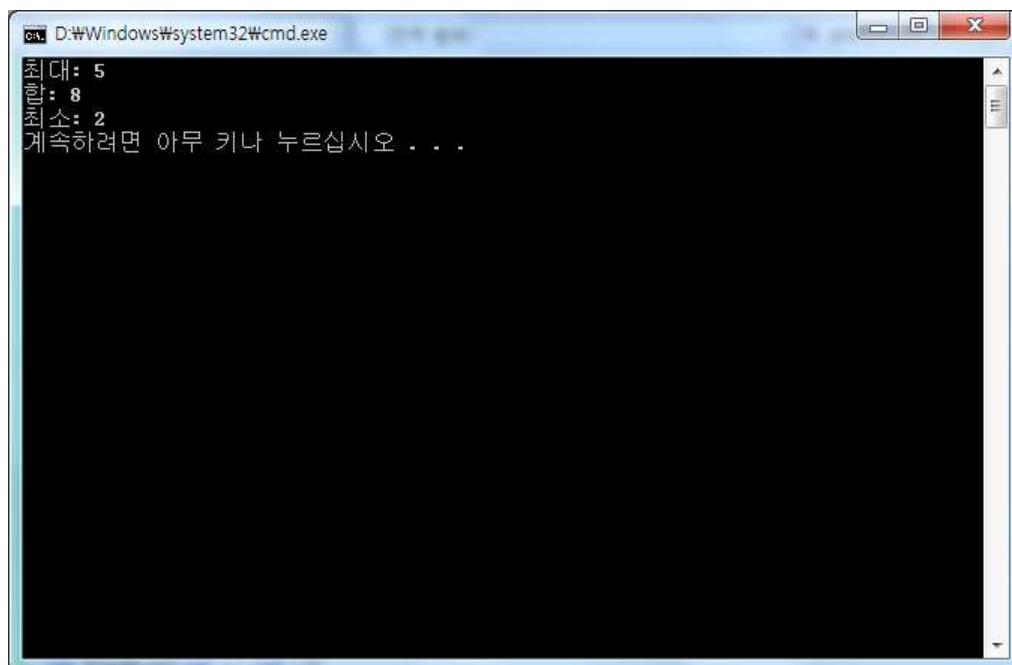
```

return min;
}

void printMin(int a, int b)
{
int min = a < b ? a : b
printf("최소: %d\n", min);

return ;
}

```



6. 함수에서의 배열 전달

함수헤더에 int ary[]로 기술하는 것은 int *ary로도 대체 가능하다.

int sumary(int ary[], int SIZE)	int sumaryf(int *ary, int SIZE)
{	{
...	...
}	}
int (i=0; I < SIZE; i++)	for(i=0; I < SIZE; i++)
{	{
sum += ary[i];	sum += *(ary + i);
}	}

for (i=0; i<SIZE; i++)	for (i=0; i<SIZE; i++)
{	{
sum += ary[i];	sum += *(ary++);
}	}

밑에 예제 소스가 있다.

```
#include <stdio.h>
int sumary(int *ary, int SIZE);

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

    int sum = 0;
    for (int i = 0; i < aryLength; i++)
        sum += *(point + i);

    printf("메인에서구한합은%d\n", sum);
    address = point;
    printf("함수sumary() 호출로구한합은%d\n", sumary(point, aryLength));
    printf("함수sumary() 호출로구한합은%d\n", sumary(&point[0], aryLength));
    printf("함수sumary() 호출로구한합은%d\n", sumary(address, aryLength));

    return 0;
}

int sumary(int *ary, int SIZE)
{
    int sum = 0;
    for (int i = 0; i < SIZE; i++)
    {
        sum += *ary++;
    }
    return sum;
}
```

```
cmd.exe D:\Windows\system32\cmd.exe
메인에서 구한 값은 755
함수 sumary<> 호출로 구한 값은 755
함수 sumary<> 호출로 구한 값은 755
함수 sumary<> 호출로 구한 값은 755
계속하려면 아무 키나 누르십시오 . . .
```

7. 재귀와 함수 구현

재귀특성

함수구현에서 자신 함수를 호출하는 함수를 재귀함수라 한다.

```
if (n <= 1)          int factorial(int num)
    n! = 1            {
else                  {
    n! = n * (n-1)!   if(num <= 1)
                      return 1;
                      else
                      return (num * factorial(num - 1));
                      }
}
```

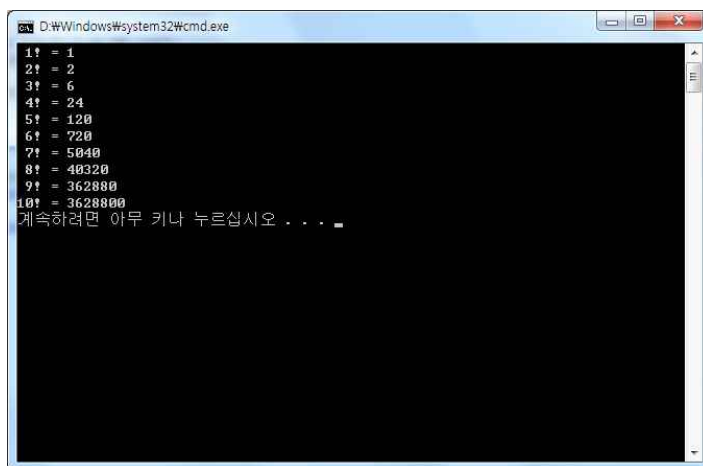
밑에 소스예제가 있다.

```
#include <stdio.h>
int factorial(int);

int main(void)
{
    for (int i = 1; i <= 10; i++)
        printf("%2d! = %d\n", i, factorial(i));
}
```

```
return 0;
}
```

```
int factorial(int number)
{
    if (number <= 1)
        return 1;
    else
        return (number * factorial(number - 1));
}
```



8. 함수rand()

특정한 나열 순서나 규칙을 가지지 않는 연속적인 임의의 수를 난수라 한다.

```
#include <stdlib.h>
int main(void)
{
    ...
    printf("%5d ", rand());
}
```

밑에 예제소스가 있다.

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

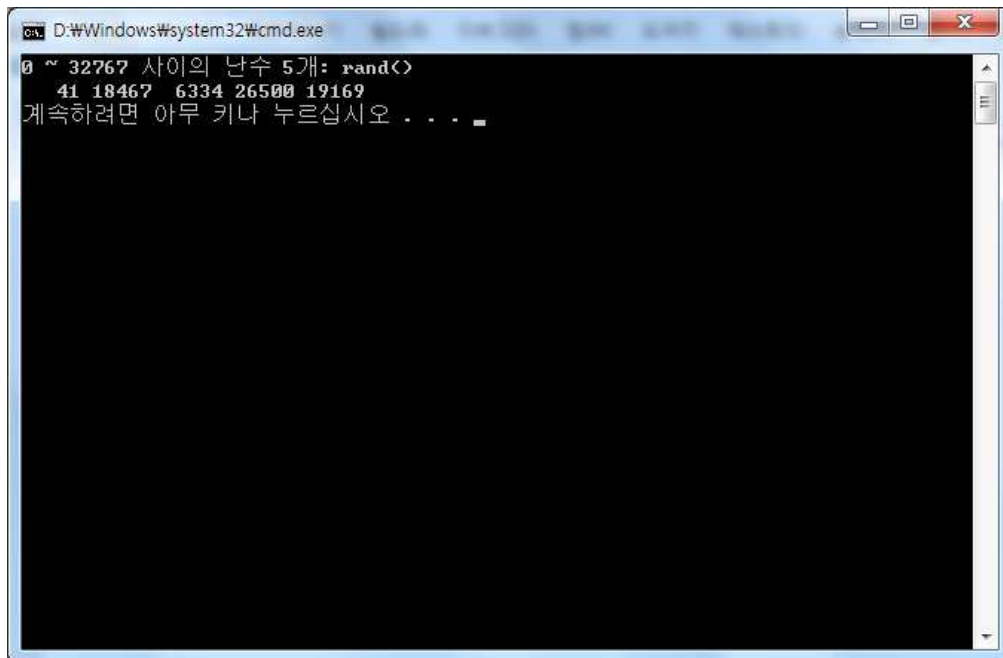
int main(void)
{
```

```

printf("0 ~ %5d »ÇÀÀÇ³-¼ö5³: rand()\n", RAND_MAX);
for (int i = 0; i < 5; i++)
printf("%5d ", rand());
puts("");

return 0;
}

```



```

D:\Windows\system32\cmd.exe
0 ~ 32767 사이의 난수 5개: rand()
 41 18467 6334 26500 19169
계속하려면 아무 키나 누르십시오 . . .

```

9. 함수 srand()

함수rand()와 다르게 매번 난수를 변화시켜 생성하면 시드값을 이용한다.
이 시드(seed)값은 난수를 다르게 만들기 위해 처음에 지정하는수로서 시드값이
달라지면 난수가 달라진다. 이처럼 seed값을 이용하는 함수가 srand()이다.

밑에 예제소스가 있다.

```

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

#include <time.h>
#define MAX 100
int main(void)

```

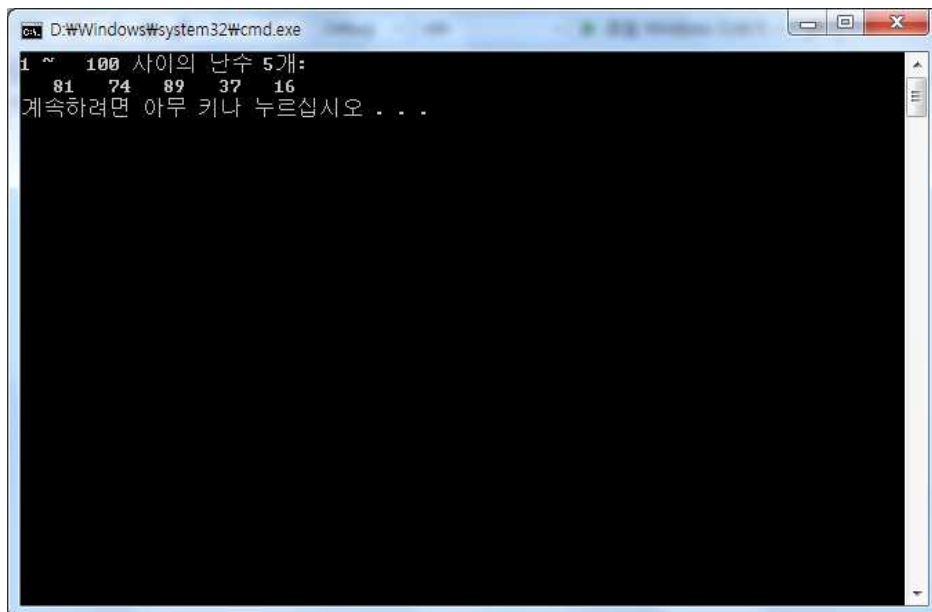
```

{
long seconds = (long)time(NULL);
srand(seconds);

printf("1 ~ %5d 사이의 난수 5개:\n", MAX);
for (int i = 0; i < 5; i++)
printf("%5d", rand() % MAX + 1);
puts("");

return 0;
}

```



10. math.h

수학 관련 함수를 사용하려면 헤더파일 math.h를 삽입해야 한다.

밑에 표가 있다.

함수	처리작업
double sin(double x)	삼각함수 sin
double cos(double x)	삼각함수 cos
double tan(double x)	삼각함수 tan
double sqrt(double x)	제곱근, square root(x)
double exp(double x)	e^x
double log(double x)	$\log_e(x)$
double log10(double x)	$\log_{10}(x)$
double pow(double x, double y)	x^y

double ceil(double x)	x보다 작지 않은 가장 작은 정수
double floor(double x)	x보다 크지 않은 가장 큰 정수
int abs(int x)	정수 x의 절대 값
double fabs(double x)	실수 x의 절대 값

밑에 예제소스가 있다.

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

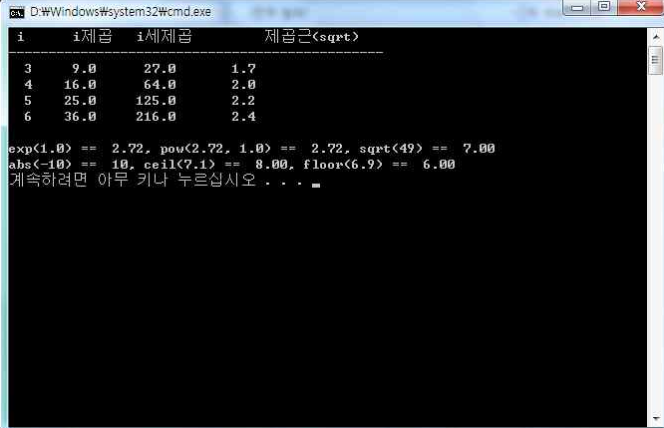
int main(void)

{

printf("iiÁ!öi¼¼Á!öÁ!ö±Ü(sqrt)\n");
printf("-----\n");
for (int i = 3; i < 7; i++)
printf("%3d %7.1f %9.1f %9.1f\n", i, pow(i, 2), pow(i, 3), sqrt(i));
printf("\n");

printf("exp(1.0) == %5.2f, ", exp(1.0));
printf("pow(2.72, 1.0) == %5.2f, ", pow(2.72, 1.0));
printf("sqrt(49) == %5.2f\n", sqrt(49));
printf("abs(-10) == %3d, ", abs(-10));
printf("ceil(7.1) == %5.2f, ", ceil(7.1));
printf("floor(6.9) == %5.2f\n", floor(6.9));

return 0;
}
```



```
D:\Windows\system32\cmd.exe
i      i제곱   i세제곱   제곱근<sqrt>
3      9.0     27.0     1.7
4      16.0    64.0     2.0
5      25.0    125.0    2.2
6      36.0    216.0    2.4

exp(1.0) == 2.72, pow(2.72, 1.0) == 2.72, sqrt(49) == 7.00
abs(-10) == 10, ceil(7.1) == 8.00, floor(6.9) == 6.00
계속하려면 아무 키나 누르십시오 . . .
```

11. 헤더파일 ctype.h

C언어에서 문자관련 함수는 헤더파일 ctype.h에 매크로가 정의되어 있다.
 밑에 표가있다.

함수원형	기능
isalpha(char)	영문자 검사
issupper(char)	영문 대문자 검사
islower(char)	영문 소문자 검사
isdigit(char)	숫자(0~9) 검사
isxdigit(char)	16진수 숫자(0~9, A~F, a~f)검사
isspace(char)	공백(' ', '\n', '\t', '\f', '\v', '\r')문자 검사
ispunct(char)	구두(빈칸이나 알파뉴메릭 제외한 출력문자) 문자검사
isalnum(char)	영문과 숫자(alphanumeric)(0~9, A~Z, a~z)검사
isprint(char)	출력 가능 검사
isgraph(char)	그래픽 문자 검사
iscntrl(char)	제어문자('\a', '\b', '\n', '\t', '\f', '\v', '\r')검사
toupper(char)	영문 소문자를 대문자로 변환
tolower(char)	영문 대문자를 소문자로 변환
toascii(char)	아스키 코드로 변환
_tolower(char)	무조건 영문 소문자로 변환
_toupper(char)	무조건 영문 대문자로 변환

밑에 예제소스가 있다.

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include<ctype.h>

void print2char(char);
int main(void)

{
char ch;

printf("알파벳(종료x) 또는다른문자입력하세요.\n");
do
{
printf("문자입력후Enter: ");
scanf("%c", &ch);
getchar();
if (isalpha(ch))
```



```

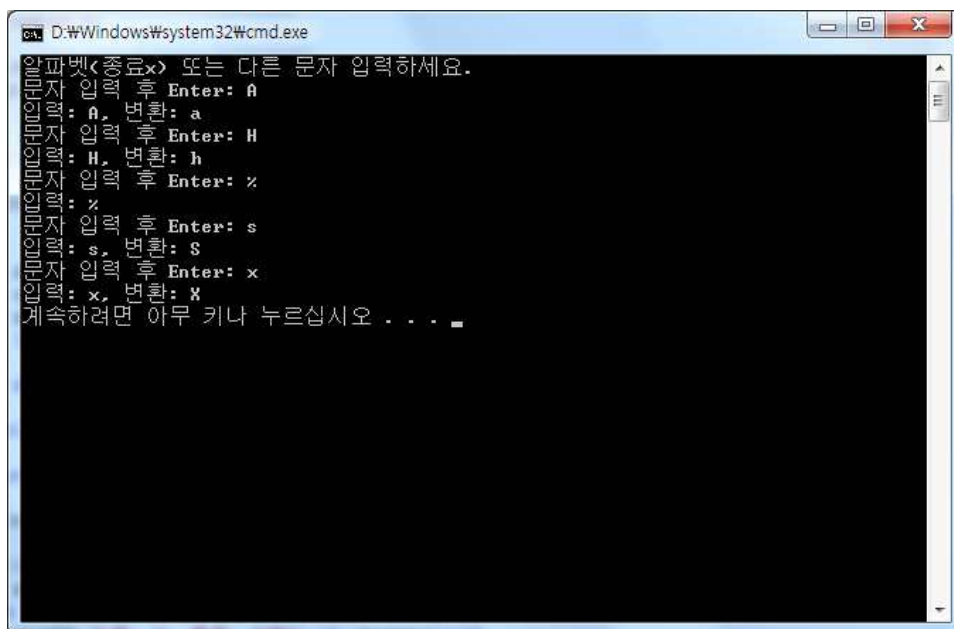
print2char(ch);
else
printf("입력: %c\n", ch);
} while (ch != 'x' && ch != 'X');

return 0;
}

void print2char(char ch)
{
if (isupper(ch))
printf("입력: %c, 변환: %c\n", ch, tolower(ch));
else
printf("입력: %c, 변환: %c", ch, toupper(ch));

return 0;
}

```



```

D:\Windows\system32\cmd.exe
파벳(중요) 또는 다른 문자 입력하세요.
입력 후 Enter: A
: A, 변환: a
입력 후 Enter: H
: H, 변환: h
입력 후 Enter: Z
: Z
입력 후 Enter: S
: S, 변환: S
입력 후 Enter: X
: X, 변환: X
계속하려면 아무 키나 누르십시오 . . .

```

12. 다양한 헤더파일

C언어는 다양한 라이브러리 함수가 있다.

헤더파일	처리작업
stdio.h	표준 입출력 작업
math.h	수학 관련 작업
string.h	문자열 작업
time.h	시간 작업
ctype.h	문자 관련 작업
stdlib.h	여러 유틸리티(텍스트를 수로 변환 등) 함수

-끝-