

1. C언어 기본구조 및 입출력

1) C언어 기본 구조

```
1  #include <stdio.h>      // 헤더 파일 - 전처리문(컴파일 전 코드의 재구성).
2
3  void main()              // main함수.
4  {                        // { 함수 본체 }.
5      printf("korea\n");
6  }
```

2) 변수의 개념

```
1  #include <stdio.h>
2
3  void main()
4  {
5      int a = 3;           // 변수의 선언 => 타입명 변수명 [ = 초기값 ].
6      int b = a;           // - 변수의 사용 목적 : (1) 값의 변화. (2) 값의 기억.
7
8      printf("%d %d\n", a, b);
9  }
```

※ 기초 변수 개념

00000001	00000010	00000011	00000100
00000101	00000110	00000111	00001000
00001001	00001010	00001011	00001100
00001101	00001110	00001111	00010000

int a = 3 ;

※ 일반 수학에서의 대입 연산

$$\underline{X} + 5 = \underline{Y}$$

좌변값

우변값

※ C언어에서의 대입 연산과 초기화

$$\underline{X} = \underline{5}$$

좌변값

우변값

$$\underline{Y} = \underline{X + 5}$$

00000001	00000010	00000011	00000100
5			
00000101	00000110	00000111	00001000
00001001	00001010	00001011	00001100
00001101	00001110	00001111	00010000
10			

3) 기본 입출력

(1) printf

```
1  #include <stdio.h>
2
3  void main()
4  {
5      int Num = 123;
6
7      printf("==%d==\n", Num);
8      printf("==%5d==\n", Num);
9      printf("==%05d==\n", Num);
10     printf("==%-5d==\n", Num);
11
12     int value = 12;
13     char ch = 'S';
14
15     printf("값은 %d이며 문자는 %c이다.\n", value, ch);
16 }
```

(2) scanf_s

```
1  #include<stdio.h>
2
3  void main()
4  {
5      int a, b;
6      int sum;
7
8      printf("첫 번째 숫자를 입력하세요 : ");
9      scanf_s("%d", &a);
10     printf("두 번째 숫자를 입력하세요 : ");
11     scanf_s("%d", &b);
12
13     sum = a + b;
14     printf("입력한 두 수의 합은 %d입니다.\n", sum);
15 }
```

<프로젝트 실습>

<1단계>

-다음 출력 예시와 같이 학번과 점수를 입력 받아 출력하는 프로그램을 구현하라.

```
학번과 점수를 입력하시오.  
  
양아치 학번 : 1  
양아치 점수 : 50  
  
날라리 학번 : 2  
날라리 점수 : 70  
  
양아치 학번은 001, 점수는 50점입니다.  
날라리 학번은 002, 점수는 70점입니다.  
  
계속하려면 아무 키나 누르십시오 . . .
```

2. 변수의 종류

※변수의 자료형(기본형과 참조형)

기본형	정수형
	실수형
	문자형
	열거형
	논리형(bool) - C++부터 기본형에 추가
참조형	사용자 정의형(typedef)
	배열
	구조체
	공용체
	포인터

※ 논리형

- 기존 C에서는 사용자 정의형과 매크로 상수를 이용하던지 아니면, 열거형을 통해서 BOOL형을 정의하여 TRUE(1), FALSE(0)로 사용. 이후 C++부터 기본형에 포함되어 bool 타입의 true, false 형태의 값으로 사용.
- C에서 bool형 true, false를 사용하려면 "stdbool.h"헤더파일을 사용.

1) 정수

```
1  #include<stdio.h>
2  #include<limits.h>    // SHRT_MIN, SHRT_MAX, INT_MIN등의 정수 자료 최댓값, 최솟값 상수 정의 헤더 파일.
3
4  void main()
5  {
6      short a;           // signed short.
7      int b;             // signed int.
8      long c;            // signed long int. < long은 플랫폼에 정적인 반면 int는 플랫폼에 따라 가변적. >
9      long long d;       // signed long long int.
10     unsigned short a1;  // unsigned short int.
11     unsigned b1;        // unsigned int.
12     unsigned long c1;   // unsigned long int.
13     unsigned long long d1; // unsigned long long int.
14
15     printf("%d %d %d %d\n", sizeof(a), sizeof(b), sizeof(c), sizeof(d)); // sizeof( 대상 변수 or 상수 or 타입 ) :
16     printf("%d %d %d %d\n", sizeof(a1), sizeof(b1), sizeof(c1), sizeof(d1)); // -대상에 대한 크기를 byte단위로 리턴.
17
18     printf("(signed)short 범위 : %d ~ %d\n", SHRT_MIN, SHRT_MAX);
19     printf("unsigned short 범위 : %d ~ %d\n", 0, USHRT_MAX); // 부호가 없어 범위 최솟값 상수는 필요없어 정의되지 않음.
20
21     printf("(signed)int 범위 : %d ~ %d\n", INT_MIN, INT_MAX);
22     printf("unsigned (int) 범위 : %d ~ %u\n", 0, UINT_MAX); // "%u" : "%d"는 부호 있는 4byte 출력 서식인데 반해-
23                     // "%u"는 부호없는 4byte 출력 서식.
24
25     printf("(signed)long 범위 : %d ~ %d\n", LONG_MIN, LONG_MAX);
26     printf("unsigned long 범위 : %d ~ %u\n", 0, ULONG_MAX);
27
28     printf("(signed)long long 범위 : %lld ~ %lld\n", LLONG_MIN, LLONG_MAX); // "%lld"는 부호 있는 8byte 출력 서식.
29     printf("unsigned long long 범위 : %d ~ %llu\n", 0, ULLONG_MAX); // "%llu"는 부호없는 8byte 출력 서식.
30 }
```

2) 실수

```
1  #include<stdio.h>
2  #include<float.h>    // FLT_MIN, FLT_MAX, DBL_MIN, DBL_MAX등의 실수 자료 최댓값, 최솟값 상수 정의 헤더 파일.
3
4  void main()
5  {
6      float a; // 실수형은 정수와 달리 음수에 대한 보수 표현식이 아닌, 부호와 절댓치 표현식을 사용하여 단순히
7      double b; // -부호비트로 음수와 양수를 구분. 따라서 signed 키워드를 붙여봐야 조합불가 에러 처리되며 "FLT_MIN"
8               // -과 "DBL_MIN"은 정수와 달리 음수에 대한 최솟값이 아닌 마이너스(-) 거듭제곱에 대한 최솟값 정의.
9      printf("%d %d\n", sizeof(a), sizeof(b));
10
11     printf("float 범위 : %e ~ %e\n", FLT_MIN, FLT_MAX); // "%e" : 부동소수점 출력 서식.
12     printf("double 범위 : %e ~ %e\n", DBL_MIN, DBL_MAX);
13 }
```


3) 문자

```
1  #include<stdio.h>
2  #include<limits.h>      // CHAR_MIN, CHAR_MAX의 헤더 파일.
3
4  void main()
5  {
6      char ch = 'A';
7
8      printf("%dWn", sizeof(ch));
9      printf("%d %dWn", CHAR_MIN, CHAR_MAX);
10     printf("%d %cWn", ch, ch);      // 정수와 호환.
11 }
```

※ 아스키 코드표

10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자
0	0X00	NULL	32	0x20	SP	64	0x40	@	96	0x60	`
1	0X01	SOH	33	0x21	!	65	0x41	A	97	0x61	a
2	0X02	STX	34	0x22	"	66	0x42	B	98	0x62	b
3	0X03	ETX	35	0x23	#	67	0x43	C	99	0x63	c
4	0X04	EOT	36	0x24	\$	68	0x44	D	100	0x64	d
5	0X05	ENQ	37	0x25	%	69	0x45	E	101	0x65	e
6	0X06	ACK	38	0x26	&	70	0x46	F	102	0x66	f
7	0X07	BEL	39	0x27	'	71	0x47	G	103	0x67	g
8	0X08	BS	40	0x28	(72	0x48	H	104	0x68	h
9	0X09	HT	41	0x29)	73	0x49	I	105	0x69	i
10	0X0A	LF	42	0x2A	*	74	0x4A	J	106	0x6A	j
11	0X0B	VT	43	0x2B	+	75	0x4B	K	107	0x6B	k
12	0X0C	FF	44	0x2C	,	76	0x4C	L	108	0x6C	l
13	0X0D	CR	45	0x2D	-	77	0x4D	M	109	0x6D	m
14	0X0E	SO	46	0x2E	.	78	0x4E	N	110	0x6E	n
15	0X0F	SI	47	0x2F	/	79	0x4F	O	111	0x6F	o
16	0X10	DLE	48	0x30	0	80	0x50	P	112	0x70	p
17	0X11	DC1	49	0x31	1	81	0x51	Q	113	0x71	q
18	0X12	SC2	50	0x32	2	82	0x52	R	114	0x72	r
19	0X13	SC3	51	0x33	3	83	0x53	S	115	0x73	s
20	0X14	SC4	52	0x34	4	84	0x54	T	116	0x74	t
21	0X15	NAK	53	0x35	5	85	0x55	U	117	0x75	u
22	0X16	SYN	54	0x36	6	86	0x56	V	118	0x76	v
23	0X17	ETB	55	0x37	7	87	0x57	W	119	0x77	w
24	0X18	CAN	56	0x38	8	88	0x58	X	120	0x78	x
25	0x19	EM	57	0x39	9	89	0x59	Y	121	0x79	y
26	0x1A	SUB	58	0x3A	:	90	0x5A	Z	122	0x7A	z
27	0x1B	ESC	59	0x3B	;	91	0x5B	[123	0x7B	{
28	0x1C	FS	60	0x3C	<	92	0x5C	\	124	0x7C	
29	0x1D	GS	61	0x3D	=	93	0x5D]	125	0x7D	}
30	0x1E	RS	62	0x3E	>	94	0x5E	^	126	0x7E	~
31	0x1F	US	63	0x3F	?	95	0x5F	_	127	0x7F	DEL

4) 상수

(1) const

```
1  #include<stdio.h>
2
3  void main() {
4      const double PI = 3.14;
5      int radius;
6      //PI = 3.15;
7
8      scanf_s("%d", &radius);
9      printf("%d * 3.14 = %.2f\n", radius, radius*PI);
10 }
```

(2) define

```
1  #include<stdio.h>
2
3  #define CHARGE 5000    // 매크로 상수. 컴파일 시 치환. 타입 지정 불가.
4
5  void main() {
6      int time;
7
8      printf("사용 시간 입력 : ");
9      scanf_s("%d", &time);
10
11     printf("사용 요금은 %d원입니다.\n", time * CHARGE);
12 }
```


5) 사용자 정의 타입 typedef과 부호비트의 적용

```
1  #include<stdio.h>
2
3  /*
4   signed와 unsigned의 차이점은 데이터 확장에 따른 데이터 재 변환 여부. 양수인 경우에는 확장에 따른 차이점이 발생하지 않으나, 음수인 경우
5   signed는 데이터 확장에 따라 데이터의 재 변환을 가함. 하지만 unsigned의 경우 음수를 저장하여도 기본 보수 형태는 그대로 유지하나 이미
6   부호가 없는 형태로 명시하였으므로 데이터 확장을 하여도 보수 형태에 맞추어 재 변환을 가하지 않음.
7   */
8   typedef unsigned char uCh;
9   typedef unsigned int uint;
10
11 void main() {
12     char a = -10;    // "(signed) char"의 데이터 범위는 1byte(8bit) "-128 ~ 127". 리터럴 -10은 보수 형태인 "11110110"로 a에 저장. -
13                     // "%u" 출력서식 범위는 부호가 포함되지 않은 4byte(데이터 32bit) "0~4294967295"이고, "%d" 출력서식 범위는 -
14                     // 부호가 포함된 4byte(부호 1bit, 데이터 31bit) "-2147483648~2147483647". 저장 데이터는 1byte인데 반해 출력 -
15                     // 서식은 4byte 형태임에 따라 출력서식에 맞추어 데이터 변환. 즉, -10은 출력서식에 맞추어 4byte로 확장된 보수
16                     // -형태인 "11111111 11111111 11111111 11110110"로 확장 변환되어 출력. 따라서 "%u" 출력서식의 경우 첫 bit를-
17                     // 데이터 bit로 인식하여 그대로 절댓값 형태로 출력되나, "%d"의 경우 첫 bit를 부호로 인식하여 보수 전환을 거친
18                     // -절댓값에 부호를 더하여 출력.
19     printf("%u %d\n\n", a, a);
20
21
22     uCh b1 = -10;    // 리터럴 -10은 실제 보수 형태인 "11110110"로 b1에 저장. "%d" 출력서식에 의해 "00000000 00000000 00000000 11110110"로
23                     // -전환되어 "246"출력. 여기서 signed와 unsigned의 차이점 발생. 즉, 부호 있는 저장 데이터의 경우 출력데이터의 형식에
24                     // -맞추어 확장되어 변환되는 것을 알 수 있지만 부호가 없는 저장 데이터의 경우, 이미 데이터 타입이 부호가 없는 형태로
25                     // -명시한 상태이므로 -10에 대한 4byte형식으로 확장 재 변환하지 않고 기존 1byte 형식은 그대로 유지한 채 4byte형식에-
26                     // 맞추기 위해 3byte만 추가한 형태임을 알 수 있음.
27
28     uCh b2 = 256;    // 리터럴 "256"은 "1 00000000"이고 b2는 부호없는 1byte 타입이므로 Overflow가 발생되어
29                     // -"00000000"형태로 저장되므로 "0" 출력.
30     printf("%u %u\n\n", b1, b2);
31     printf("%d %d\n\n", b1, b2);
32
33
34     uint c1 = -2147483647; // c1과 c2모두 출력서식과 같은 4byte타입이므로 데이터 확장은 발생하지 않으며, 두 수 모두 음수 형태이므로 보수-
35     uint c2 = -200;      // 형태로 저장되어 "%u"의 경우 첫 bit를 데이터 bit로 인식하여 출력되고 "%d"의 경우 첫 bit를 부호 bit로 인식-
36                     // 하여 보수 전환을 거친 절댓값에 부호를 더하여 표시함으로써 음수 입력값 그대로 출력.
37     printf("%u %u\n\n", c1, c2);
38     printf("%d %d\n\n", c1, c2);
39 }
```

3. 연산자

1) 산술연산자

```
1 #include<stdio.h>
2
3 void main() {
4     int a = 20, b = 6;
5
6     printf("a + b = %d\n", a + b);
7     printf("a - b = %d\n", a - b);
8     printf("a * b = %d\n", a * b);
9     printf("a / b = %d\n", a / b);    // 동일 타입끼리의 연산 결과는 타입 유지. 즉, 정수와
10                                     // -정수의 연산 결과는 정수.
11     printf("a %% b = %d\n", a % b); // "%"는 서식지정을 위한 특수문자. 따라서 이러한 특수문자
12                                     // -자체를 출력하기 위해서는 "%"뒤에 해당 문자를 표기.
13
14
15     printf("a = a + b => %d\n", a = a + b);    // a와 b의 연산 결과를 다시 a에 대입(리턴).
16     printf("a = a - b => %d\n", a = a - b);
17     printf("a = a * b => %d\n", a = a * b);
18     printf("a = a / b => %d\n", a = a / b);
19     printf("a = a %% b => %d\n", a = a % b);
20
21
22     printf("a += b => %d\n", a += b);    // 복합 대입 연산자.
23     printf("a -= b => %d\n", a -= b);
24     printf("a *= b => %d\n", a *= b);
25     printf("a /= b => %d\n", a /= b);
26     printf("a %= b => %d\n", a %= b);
27
28     a = 20;
29     printf("a = a - (b + 4) => %d\n", a = a - (b + 4));
30     a = 20;
31     printf("a -= b + 4 => %d\n", a -= b + 4); // 29행의 연산에 대한 복합 대입식으로 변환한 당행의 표현식을 "a = a - b + 4"과 같은-
32                                     // 식으로 이해하여 "b + 4"를 괄호로 묶지 않으면 연산 결과가 달라지는 것으로 오해하는-
33                                     // 경우 발생. 복합 대입식의 경우, 괄호로 묶을 필요가 없는 것이 좌우변의 변수가 중복-
34                                     // 되는 것을 축약하여 표현할 뿐이지 실제 '='을 기준으로 중복 변수는 좌변에 위치할 뿐-
35                                     // 우변에 존재하지 않아 우변이 먼저 연산됨. 따라서 "우변의 값을 좌변의 변수와 연산한
36                                     // -후 다시 좌변의 변수에 대입"과 같은 형태로 처리.
```

2) 관계연산자와 논리연산자

```
1  #include<stdio.h>
2
3  void main() {
4      int n1 = 10, n2 = 5;
5      int n3 = n1;
6
7      printf("0 AND 0 = %d\n", 0 && 0);    // AND(&&) : 둘다 참이면 참. (0은 거짓, 1 참)
8      printf("0 AND 1 = %d\n", 0 && 1);
9      printf("1 AND 0 = %d\n", 1 && 0);
10     printf("1 AND 1 = %d\n\n", 1 && 1);
11
12     printf("0 OR 0 = %d\n", 0 || 0);    // OR(||) : 둘 중 하나만 참이면 참.
13     printf("0 OR 1 = %d\n", 0 || 1);
14     printf("1 OR 0 = %d\n", 1 || 0);
15     printf("1 OR 1 = %d\n\n", 1 || 1);
16
17     printf("NOT 0 => %d\n", !0);        // NOT(!) : 부정
18     printf("NOT 1 => %d\n\n", !1);
19
20     printf("n1 == n3 && n1 != n2  => %d\n", n1 == n3 && n1 != n2);
21     printf("n1 > n2 || n1 > n3  => %d\n", n1 > n2 || n1 > n3);
22     printf("!( n1 >= n2 && n1 >= n3 )  => %d\n", !(n1 >= n2 && n1 >= n3));
23 }
```

3) 증감연산자

```
1  #include<stdio.h>
2
3  void main() {
4      int a = 10;
5
6      a++;          // a = a + 1, a += 1 비교.
7      printf("%d\n", a);
8      printf("%d\n", ++a);    // 전위형. 선증가후 리턴.
9      printf("%d\n", a);
10     printf("%d\n", a--);    // 후위형. 리턴후 감소.
11     printf("%d\n", a);
12 }
```

4) 삼항 조건 연산자

```
1  #include<stdio.h>
2
3  void main() {
4      int n1 = 5, n2 = 6;
5
6      // 삼항 조건 연산자 형식 => ( 조건 ) ? 참 : 거짓;
7
8      (n1 % 2 == 1 && n2 % 2 == 1) ? printf("n1과 n2는 모두 홀수이다\n") : printf("n1과 n2 둘 중 하나는 짝수이다\n");
9  }
```


5) 비트연산자

```
1  #include<stdio.h>
2
3  void main()
4  {
5      char a = 10, b = 9;
6
7      printf("a & b => %d\n", a & b);    // & : 둘다 참이면 참.
8      printf("a | b => %d\n", a | b);    // | : 둘 중 하나만 참이면 참.
9      printf("a ^ b => %d\n", a ^ b);    // ^ : 서로 다르면 참.
10     printf(" ~ a => %d\n", ~a);        // ~ : not.
11 }
```

6) 캐스팅(Casting) 연산자

```
1  #include<stdio.h>
2  /*
3   < 캐스팅(Casting) - 형변환 >
4   기본형 데이터의 경우 크기가 작은 데이터를 큰 데이터에 대입하는 경우 명시적 캐스팅을 하지 않아도
5   묵시적 캐스팅이 되면서 자동 데이터 확장이 발생함. 반대로 큰 데이터를 작은 데이터에 대입하는 경우
6   묵시적 캐스팅으로 인한 자동 형변환이 발생되지 않으며 Overflow 발생으로 인한 자료의 손실 가능성에
7   따른 경고 발생. 이에 따른 자료의 손실을 감수하면서 명시적 캐스팅을 진행하는 경우는 경고 미발생.
8   또한 동일 데이터 타입끼리의 연산 결과는 데이터 타입을 유지하나, 데이터 타입이 다른 경우 상대적으로
9   큰 데이터 타입으로 확장된 연산 진행.
10 */
11 void main()
12 {
13     int i1 = 5;
14     int i2;
15     double d1 = 3.14;
16     double d2;
17
18     d2 = i1;                // 묵시적 형변환.
19     printf("%lf\n", d2);
20     d2 = (double)i1;        // 명시적 형변환.
21     printf("%lf\n", d2);
22
23
24     i2 = d1;                // 묵시적 형변환이 발생되지 않음에 따라 Overflow 발생으로 인한 데이터 손실
25                             // -가능성에 대한 경고 발생.
26     printf("%d\n", i2);
27     i2 = (int)d1;           // 데이터 손실을 감수하면서 강제 형변환을 명시하였으므로 데이터 손실 가능성
28     printf("%d\n", i2);    // -에 대한 경고 미발생.
29
30
31     printf("%d\n", i1 / i2);    // 동일한 데이터 타입에 대한 연산 결과는 데이터 타입 유지.
32     printf("%lf\n", (double)i1 / i2); // 정수형을 명시적 형변환을 통해 실수형으로 임시 변환함으로써-
33                                     // 큰 데이터 타입으로 확장된 연산 결과 리턴.
34 }
```

<프로젝트 실습>

<1단계>

-아래 결과 화면과 같이 정수 하나를 입력받고 비트연산자를 활용하여 부호를 변환하라.

```
정수 입력 : 7
비트연산자 활용 부호변환 : -7

계속하려면 아무 키나 누르십시오 . . .
```

<2단계>

-아래와 같은 코딩 후 출력 결과를 예측하라.

```
1  #include<stdio.h>
2
3  void main()
4  {
5      int n1 = 3, n2 = 5, n3;
6
7      n1 += (++n2);
8      n2 -= (n1--);
9      n3 = (++n1) - (n2++);
10
11     printf("n1=%d, n2=%d, n3=%d\n", n1, n2, n3);
12 }
```

<3단계>

-1~100사이의 자연수를 입력받아 범위에 해당 하면 “정확히 입력!!”, 범위를 벗어났으면 “입력 오류”로 출력하라.

-단, 관계 연산자와 삼항 조건 연산자 활용.

```
1 ~ 100사이의 자연수를 입력 :1

입력오류

계속하려면 아무 키나 누르십시오 . . .
```

```
1 ~ 100사이의 자연수를 입력 :99

정확히 입력!!

계속하려면 아무 키나 누르십시오 . . .
```


<4단계>

-아래와 같이 실행되도록 프로그래밍 하라. 단, 삼항 조건 연산자를 활용.

정수 2개를 입력 받아, 둘 중 큰 수를 출력하되 같으면 "같음."이라고 출력.

첫번째 정수 입력 : 4

두번째 정수 입력 : 9

큰 수는 9

계속하려면 아무 키나 누르십시오 . . .

정수 2개를 입력 받아, 둘 중 큰 수를 출력하되 같으면 "같음."이라고 출력.

첫번째 정수 입력 : 7

두번째 정수 입력 : 7

같음

계속하려면 아무 키나 누르십시오 . . .

<5단계>

-0000 ~ 1111 사이의 2진수를 입력받아 10진수로 변환하여 출력하라. 단, 해당 자릿수는 심볼릭(Symbolic) 상수로 지정.

-전역 매크로 상수 (1000자리 : THOUSAND, 100자리 : HUNDRED, 10자리 : TEN, 1자리 : UNIT)

-지역 상수 (8 : EIGHT, 4 : FOUR, 2 : TWO, 1 : ONE)

0000 ~ 1111 사이의 2진수 입력 : 1001

2진수 1001는 10진수 ==> 9

계속하려면 아무 키나 누르십시오 . . .

4. 제어문

1) 조건문

(1) if

```
1  #include<stdio.h>
2
3  void main()
4  {
5      int n;
6
7      printf("자연수 입력 : ");
8      scanf_s("%d", &n);
9      if (n > 0) {
10         printf("입력한 수 %d는 자연수입니다.\n", n);
11     }
12
13     printf("입력완료.\n");
14 }
```

(2) if_else

```
1  #include<stdio.h>
2
3  void main()
4  {
5      int n;
6
7      printf("자연수 입력 : ");
8      scanf_s("%d", &n);
9
10     if (n > 0) {
11         printf("입력한 수 %d는(은) 자연수입니다.\n", n);
12     }
13     else
14     {
15         printf("입력한 수 %d는(은) 자연수가 아닙니다.\n", n);
16     }
17 }
```

(3) if_else if

```
1  #include<stdio.h>
2
3  void main() {
4      int n;
5
6      printf("점수 입력 : ");
7      scanf_s("%d", &n);
8
9      if (n >= 90) {
10         printf("A학점\n");
11     }
12     else if (n >= 80) {
13         printf("B학점\n");
14     }
15     else if (n >= 70) {
16         printf("C학점\n");
17     }
18     else if (n >= 60) {
19         printf("D학점\n");
20     }
21     else {
22         printf("E학점\n");
23     }
24 }
```

(4) switch

```
1  #include<stdio.h>
2
3  void main() {
4      int i;
5
6      printf("회전수 입력( 1 ~ 5 회전까지만 ) : ");
7      scanf_s("%d", &i);
8
9      switch (i) {                // switch(평가 값 또는 변수){ 본체 }.
10     case 1:
11         printf("%d회전\n", i);
12         break;
13     case 2:                      // case 비교값(상수만 가능).
14         printf("%d회전\n", i);
15         break;
16     case 3:
17         printf("%d회전\n", i);
18         break;
19     case 4:
20         printf("%d회전\n", i);
21         break;
22     case 5:
23         printf("%d회전\n", i);
24         break;
25     default:                    // 생략 가능.
26         printf("회전수 범위를 벗어났습니다.\n");
27         break;
28     }
29 }
```

(5) switch(제한적 범위)

```
1  #include<stdio.h>
2
3  void main() {
4      int i;
5
6      printf("회전수 입력( 1 ~ 5 회전까지만 ) : ");
7      scanf_s("%d", &i);
8
9      switch (i) {
10         case 1:
11         case 2:
12         case 3:
13             printf("1 ~ 3회전 사이\n");
14             break;
15         case 4:
16         case 5:
17             printf("4 ~ 5회전 사이\n");
18             break;
19         default:
20             printf("회전수 범위를 벗어났습니다.\n");
21             break;
22     }
23 }
```

(6) switch(문자 상수)

```
1  #include<stdio.h>
2
3  void main() {
4      int a = 10, b = 5;
5      char op;
6
7      printf("연산자 입력( +, -, *, / ) : ");
8      scanf_s("%c", &op, 1);
9
10     switch (op) {
11         case '+':
12             printf("%d %c %d = %d\n", a, op, b, a + b);
13             break;
14         case '-':
15             printf("%d %c %d = %d\n", a, op, b, a - b);
16             break;
17         case '*':
18             printf("%d %c %d = %d\n", a, op, b, a * b);
19             break;
20         case '/':
21             printf("%d %c %d = %d\n", a, op, b, a / b);
22             break;
23
24         default:
25             printf("연산자 입력이 잘못되었습니다.\n");
26             break;
27     }
28 }
```

2) 반복문

(1) for

```
1  #include<stdio.h>
2
3  void main() {
4      for (int i = 1; i <= 10; i++) {    // for( 초기식; 조건식; 증감식 ) { 본체 }.
5          printf("%d번 출력.\n", i);    // -조건식이 생략되면 무한루프.
6      }
7  }
```

(2) for문을 이용한 누적합

```
1  #include<stdio.h>
2
3  void main() {
4      int tot = 0;
5      int n1, n2;
6
7      printf("작은 수 입력 : ");
8      scanf_s("%d", &n1);
9      printf("큰 수 입력 : ");
10     scanf_s("%d", &n2);
11
12     for (int i = n1; i <= n2; i++) {
13         tot += i;
14     }
15
16     printf("%d부터 %d까지의 합은 %d\n", n1, n2, tot);
17 }
```

(3) while

```
1  #include<stdio.h>
2
3  void main() {
4      int i = 1;
5
6      while (i < 11) {    // while(조건식) { 본체 }
7          printf("%d번 출력.\n", i++);    // -조건식이 0이 아니면 무한루프.
8      }
9  }
```


(4) while문을 이용한 누적합

```
1  #include<stdio.h>
2
3  void main() {
4      int tot = 0;
5      int n1, n2;
6      int i;
7
8      printf("작은 수 입력 : ");
9      scanf_s("%d", &n1);
10     printf("큰 수 입력 : ");
11     scanf_s("%d", &n2);
12
13     i = n1 - 1;
14
15     while (i < n2) {
16         i++;
17         tot += i;
18     }
19
20     printf("%d부터 %d까지의 합은 %d\n", n1, n2, tot);
21 }
```

(5) do_while

```
1  #include<stdio.h>
2
3  void main() {
4      int i = 1;
5
6      do                                     // do { 본체 } while(조건식);
7      {                                     // -조건식이 0이 아니면 무한루프.
8          printf("%d번 출력.\n", i++);
9      } while (i < 11);
10 }
```

(6) do_while을 이용한 누적합

```
1  #include<stdio.h>
2
3  void main() {
4      int tot = 0;
5      int n1, n2;
6      int i;
7
8      printf("작은 수 입력 : ");
9      scanf_s("%d", &n1);
10     printf("큰 수 입력 : ");
11     scanf_s("%d", &n2);
12
13     i = n1;
14
15     do {
16         tot += i;
17         i++;
18     } while (i <= n2);
19
20     printf("%d부터 %d까지의 합은 %d\n", n1, n2, tot);
21 }
```

3) 중첩루프

(1) 중첩루프 구조 분석

```
1  #include<stdio.h>
2
3  void main() {
4      for (int i = 1; i <= 5; i++)
5      {
6          for (int j = 1; j <= 5; j++)
7          {
8              printf("i : %d, j : %d\n", i, j);
9          }
10         puts("");
11     }
12 }
```

(2) 중첩루프를 활용한 구구단 출력

```
1  #include<stdio.h>
2
3  void main() {
4      for (int i = 1; i <= 9; i++)
5      {
6          for (int j = 1; j <= 9; j++)
7          {
8              printf("%d * %d = %2d ", i, j, i*j);
9          }
10         puts("");
11     }
12 }
```

4) break

```
1  #include<stdio.h>
2
3  void main() {
4      for (int i = 1; i <= 9; i++)
5      {
6          if (i >= 6) break;           // 5단 까지만 출력.
7
8          for (int j = 1; j <= 9; j++)
9          {
10             printf("%d * %d = %2d ", i, j, i*j);
11         }
12         puts("");
13     }
14 }
```

5) continue

```
1  #include<stdio.h>
2
3  void main() {
4      for (int i = 1; i <= 9; i++)
5      {
6          if (i % 2 == 0) continue;    // 홀수 구구단 출력.
7          for (int j = 1; j <= 9; j++)
8          {
9              printf("%d * %d = %2d ", i, j, i*j);
10             }
11             puts("");
12         }
13     }
```

6) 행렬 개념 활용

```
1  #include<stdio.h>
2
3  void main() {
4      for (int i = 1, v = 1; i <= 5; i++) // 출력줄의 줄 수를 가상의 행 개념으로 이해.
5      {
6          for (int j = 1; j <= 5; j++) // 출력 문자열을 열 개념으로 이해.
7          {
8              printf("%2d ", v++);
9          }
10         puts("");
11     }
12 }
```

<프로젝트 실습>

<1단계>

-아래 결과 화면과 같이 정수 두 개를 입력 받아 두 수의 차를 구하도록 구현하라.

```
< 두 정수의 차>
정수 하나 입력해봐!! : 95
정수 하나 더 입력해봐!! : 42

95, 42의 차는 : 53

계속하려면 아무 키나 누르십시오 . . .
```

<2단계>

-임의의 정수들을 입력 받아 다음과 같이 합을 구하는 프로그램을 구현하라.

-while문을 이용하고 '0'을 입력 받지 않는 한 정수를 계속 입력 받도록 하라.

```
< 입력받은 정수들의 합 - while >

정수 입력해봐~하나만(끝내려면 '0') : 1
정수 입력해봐~하나만(끝내려면 '0') : 2
정수 입력해봐~하나만(끝내려면 '0') : 3
정수 입력해봐~하나만(끝내려면 '0') : 4
정수 입력해봐~하나만(끝내려면 '0') : 5
정수 입력해봐~하나만(끝내려면 '0') : 10
정수 입력해봐~하나만(끝내려면 '0') : 0

합계 : 25

계속하려면 아무 키나 누르십시오 . . .
```

<3단계>

-아래의 결과 화면을 분석하여 구현하라.

```
출력할 줄 수 입력 : 10

1
12
123
1234
12345
123456
1234567
12345678
123456789
12345678910

계속하려면 아무 키나 누르십시오 . . .
```


-아래와 같이 짝수 구구단을 출력하라.
-단, 세부사항은 출력 형태를 분석하여 구현하라.

계속하려면 아무 키나 누르십시오 . . .

-아래 출력 결과를 분석하여 동일하게 출력되는 알고리즘을 구현하라.

계속하려면 아무 키나 누르십시오 . . .

계속하려면 아무 키나 누르십시오 . . .

<6단계>

-아래 출력 결과를 분석하여 동일하게 출력되는 알고리즘을 구현하라.

```
출력할 줄 수 입력(1보다 큰 자연수만 입력) : 5
*****
 * * * * *
  * * * * 
   * * *  
    * *   
     *    
      *
```

계속하려면 아무 키나 누르십시오 . . .

```
출력할 줄 수 입력(1보다 큰 자연수만 입력) : 12

*****
 *
  *
   *
    *
     *
      *
       *
        *
         *
          *
           *
            *
```

<7단계>

-아래 출력 결과를 분석하여 실현 가능한 프로그램을 구현하라.

[illegible]

5. 함수

1) 코드의 중첩

```
1  #include<stdio.h>
2
3  void main() {
4      int tot = 0;
5      int n1, n2;
6      int min, max;      // min : 작은 수 변수, max : 큰 수 변수.
7
8      printf("첫 번째 수 입력 : ");
9      scanf_s("%d", &n1);
10     printf("두 번째 수 입력 : ");
11     scanf_s("%d", &n2);
12
13     if (n1 > n2) {
14         min = n2;
15         max = n1;
16     }
17     else {
18         min = n1;
19         max = n2;
20     }
21
22     for (int i = min; i <= max; i++) {
23         tot += i;
24     }
25     printf("%d부터 %d까지의 합은 %d\n\n", min, max, tot);
26
27
28     tot = 0;      // 누적합 재 초기화.
29     printf("첫 번째 수 입력 : ");
30     scanf_s("%d", &n1);
31     printf("두 번째 수 입력 : ");
32     scanf_s("%d", &n2);
33
34     if (n1 > n2) {
35         min = n2;
36         max = n1;
37     }
38     else {
39         min = n1;
40         max = n2;
41     }
42
43     for (int i = min; i <= max; i++) {
44         tot += i;
45     }
46     printf("%d부터 %d까지의 합은 %d\n\n", min, max, tot);
47 }
```

2) 함수를 통한 코드의 통합

```
1  #include<stdio.h>
2
3  void from_to_tot() {           // 함수의 선언 : 리턴타입 함수명(형식인수) { 본체 }
4      int tot = 0;
5      int n1, n2;
6      int min, max;
7
8      printf("첫 번째 수 입력 : ");
9      scanf_s("%d", &n1);
10     printf("두 번째 수 입력 : ");
11     scanf_s("%d", &n2);
12
13     if (n1 > n2) {
14         min = n2;
15         max = n1;
16     }
17     else {
18         min = n1;
19         max = n2;
20     }
21
22     for (int i = min; i <= max; i++) {
23         tot += i;
24     }
25     printf("%d부터 %d까지의 합은 %d\n\n", min, max, tot);
26 }
27
28 void main() {
29     for (int i = 0; i < 3; i++)
30     {
31         from_to_tot();         // 함수 호출 : 함수명(실인수).
32     }                          // -함수로 코드를 통합하여 코드의 중복을 제거.
33 }                              // -함수 호출 이전에 먼저 함수가 선언되어야 함.
```


3) 형식인수와 실인수

```
1  #include<stdio.h>
2
3  void from_to_tot(int n1, int n2) {           // 당행의 n1, n2는 from_to_tot()함수의 지역 변수로
4      int tot = 0;                             // -23행의 메인 함수의 지역 변수와는 다름에 유의.
5      int min, max;
6
7      if (n1 > n2) {
8          min = n2;
9          max = n1;
10     }
11     else {
12         min = n1;
13         max = n2;
14     }
15
16     for (int i = min; i <= max; i++) {
17         tot += i;
18     }
19     printf("%d부터 %d까지의 합은 %d\n\n", min, max, tot);
20 }
21
22 void main() {
23     int n1, n2;
24
25     printf("< 두 정수의 합 > - 끝내려면 두 수 모두 0 입력\n\n");
26     for (;;)
27     {
28         printf("첫 번째 수 입력 : ");
29         scanf_s("%d", &n1);
30         printf("두 번째 수 입력 : ");
31         scanf_s("%d", &n2);
32         if (n1 == 0 && n2 == 0) break;
33
34         from_to_tot(n1, n2);           // 함수 호출에 의해 실인수 값이 형식인수에 대입. 함수 호출 -
35     }                                   // 시 실인수는 형식인수의 타입 및 갯수와 동일하게 적용해야 함.
36
37     printf("\n\n< 연 산 종 료 >\n\n");
38 }
```

4) 리턴타입

```
1  #include<stdio.h>
2
3  double avgFunc() {                // 리턴타입이 void가 아닌 타입이 지정된 경우 19행과 같이-
4      int n;                        // N개의 점수.                // 무조건 return 키워드를 통해 리턴 타입에 적용되는 값을-
5      int cnt = 0;                  // 입력 점수 카운트.        // 명시해야 함.
6      int jum;                      // 입력 점수.
7      int sum = 0;
8
9      printf("입력 받을 점수의 갯수 : ");
10     scanf_s("%d", &n);
11
12     do
13     {
14         printf("%d번째 점수 :", ++cnt);
15         scanf_s("%d", &jum);
16         sum += jum;
17     } while (cnt < n);
18
19     return sum / n;                // return이 호출된 후 함수 종료.
20 }
21
22 void main() {
23     double avg;
24
25     printf("N개의 점수를 입력 받아 평균을 구하는 알고리즘을 구현하세요.^^\n\n");
26     avg = avgFunc();
27
28     printf("\n\n평균 : %.1lf\n\n", avg);
29 }
```

5) retrain

```
1  #include<stdio.h>
2
3  void avgFunc() {
4      int n;                        // N개의 점수.
5      int cnt = 0;                  // 입력 점수 카운트.
6      int jum;                      // 입력 점수.
7      int sum = 0;
8
9      printf("입력 받을 점수의 갯수 : ");
10     scanf_s("%d", &n);
11
12     while (1) {
13         printf("%d번째 점수 :", ++cnt);
14         scanf_s("%d", &jum);
15         sum += jum;
16         if (cnt == n) {
17             printf("\n\n평균 : %.1lf\n\n", (double)sum / n);
18
19             return;                // return이 호출되면서 루프와 관계없이 함수 종료.-
20                                     // 함수의 리턴 타입이 void이므로 리턴 값은 미 설정.
21         }
22     }
23
24 void main() {
25     printf("N개의 점수를 입력 받아 평균을 구하는 알고리즘을 구현하세요.^^\n\n");
26     avgFunc();
27 }
```

6) 함수원형

```
1  #include<stdio.h>
2
3  int sum(int a, int b);    // 함수원형 : 22행의 함수 정의에서 본체를 빼고 ";"만 추가한 형태. 함수의
4                             // 정의는 호출전에 선언하는 것이 원칙이나, 이처럼 함수원형을 호출전에
5                             // 선언함으로써 위치에 상관없이 함수 호출이 가능.
6
7  int subtract(int, int);   // 함수원형 선언 시 리턴 타입과 형식인수의 타입 및 갯수가 함수 정의부와
8                             // 모두 일치해야 하나 형식인수의 변수는 당행과 같이 생략 가능. 이처럼
9                             // 함수원형의 형식인수 변수는 명확한 표현을 위한 명시적 기능을 가질뿐
10                             // 실제 변수로써 본연의 기능을 내포하지 않음으로 생략 또는 변경 가능.
11
12  int multiply(int firstNum, int secondNum);    // 함수원형에 대한 형식인수 변수는 명시적 기능.
13
14  void main() {
15      int n1 = 10, n2 = 5;
16
17      printf("%d과 %d의 합은 %d입니다.\n", n1, n2, sum(n1, n2));
18      printf("%d과 %d의 차는 %d입니다.\n", n1, n2, subtract(n1, n2));
19      printf("%d과 %d의 곱은 %d입니다.\n", n1, n2, multiply(n1, n2));
20  }
21
22  int sum(int a, int b) {
23      return a + b;
24  }
25
26  int subtract(int a, int b) {
27      return a - b;
28  }
29
30  int multiply(int a, int b) {
31      return a * b;
32  }
```

7) 재귀함수(재귀호출)

```
1  #include <stdio.h>
2
3  /*
4   < 예시 문제 > - 100부터 1까지의 합.
5   */
6
7  int sum(int n)
8  {
9      if (n >= 101) return 0;    // 재귀함수의 무한루프를 탈출하기 위한 종료 조건.
10
11     return n + sum(n + 1);      // 당 인수값에 차인수값을 실인수로 가지는 함수 호출 결과를 더한 값을 리턴. 즉, -
12 }                               // "1 + sum(2) -> 2 + sum(3) -> 3 + sum(4)... "과 같은 순서의 반복적인 형태로 함수
13                                // -호출 및 값을 리턴. 따라서 최초 연산시점은 9행의 조건에 의해 최종 호출 시점-
14                                // ("100 + sum(101)")이 되어 다음과 같이 호출 순서와 반대가 되는 역순 연산이 진행. -
15 void main()                    // "100 + sum(101) -> 99 + sum(100) -> 98 + sum(99) -> -> 97 + sum(98) ->..." -
16 {                               // "100 + 0 -> 99 + 100 -> 98 + 199 -> -> 97 + 297 ->..." -
17     printf("%d\n", sum(1));    // 이와 같이 자기 자신을 호출하는 재귀호출은 호출 순서에 따른 시점에서 봤을 때 -
18 }                               // 당함수 내에서 호출한 차함수의 결과 값을 리턴 받는 형태가 반복되어 9행의 탈출 -
19                                // 조건에 의해 최종 호출시점이 반대로 최초 연산 시점이 됨. 따라서 함수 호출과 실제
20                                // -연산 진행 순서가 역순이 되므로 100부터 1까지의 합을 구하기 위해서는 최초 호출 -
21                                // 시점에서 실인수를 1로 전달하여야만 최종 연산 대상이 되는 인수로 설정됨. 또한 -
22                                // 9행의 종료 조건에 설정된 인수 n은 호출 로직상 항상 차인수로 평가되므로 최종당 -
23                                // 인수가 100으로 끝나기 위해서는 차인수가 101로 설정되어야 역순 연산에 따른 최초 -
24                                // -연산 대상 항목이 100부터 시작됨. 연산 과정이 호출순서의 역순으로 진행됨에 따른-
25                                // 호출 시점에서의 누산합 과정을 분석해 보면 당호출 시점이 당누산합으로 평가되고 -
26                                // 차호출 시점이 전누산합으로 또한 차인수가 당항값으로 평가됨. 이처럼 연산 순서에 -
27                                // 따른 호출시점에서의 분석이 헛갈린다면, 연산 순서로의 시점으로만 분석하되 당함수-
28                                // 에서 리턴되는 연산 결과를 당누산합으로 평가함으로써 당인수가 당항값으로 평가되고
29                                // -차인수를 실인수로 하는 재귀 호출함수의 리턴 결과를 전누산합으로 평가.
```


8) 지역변수와 전역변수의 통용범위

```

1  #include<stdio.h>
2
3  int i;           // 전역 변수. 지역 변수는 블록 내로 통용 범위가 제한되는 반면,
4                  // -전역 변수는 선언한 소스 파일 전체에 통용. 또한 전역 변수는
5                  // -지역 변수와 달리 선언과 동시에 자동 초기화. 전역 변수는 -
6                  // 컴파일 시에 값이 결정되어야 하는 정적데이터 영역에 저장되어
7                  // 실행중에 값이 결정될 수 있는 다른 변수의 대입 자체가 불가.-
8                  // 즉, 상수값만 초기화 및 대입이 가능.
9
10 void outfunc() {
11     int i = 10;    // outfunc()함수의 지역 변수. 해당 지역 변수 선언에 의해 전역 변수 i는-
12                  // 가려지고 지역 변수 i로 인식. 단, 함수 블록이 종료되는 순간 지역 변수
13                  // -i도 소멸되어 외부 참조 불가.
14     int a = 15;
15     printf("outfunc함수 지역변수 i : %d\n\n", i);
16 }
17
18 void main() {
19     printf("전역변수 i : %d\n\n", i);
20
21     {
22         int i = 5;    // 블록이 종료되는 순간 지역 변수 i도 소멸되기 때문에 블록 내로 통용범위가-
23                     // -제한되고 따라서 외부에서 참조 불가.
24         printf("블록 지역변수 i : %d\n\n", i);
25     }
26
27     printf("전역변수 i : %d\n\n", i);
28
29     for (int i = 1; i < 4; i++)    // 당행의 지역 변수 i는 for문 내에서만 통용.
30     {
31         printf("for블록 지역변수 i : %d\n\n", i);
32     }
33
34     outfunc();
35
36     //printf("outfunc함수 지역변수 : %d\n\n", a);    // 13행의 outfunc()함수 내에 선언된 변수 a는 -
37                                                     // outfunc()함수의 지역 변수임에 따라 호출 불가
38                                                     // -하며 또한, 함수 호출 종료 후 소멸되어 외부
39                                                     // -에서 인식 불가.

```


9) 정적변수(static)

```
1  #include<stdio.h>
2
3  /*
4   전역 변수 : 통용 범위(소스 파일 전체), 프로그램 종료 시 소멸, 정적데이터 영역에 저장.
5   지역 변수 : 통용 범위(블록 내)           , 블록 종료 시 소멸      , 스택(Stack) 영역에 저장.
6   정적 변수 : 통용 범위(블록 내)           , 프로그램 종료 시 소멸, 정적데이터 영역에 저장.
7  */
8
9  void func() {
10     static int cnt;           // static 키워드를 이용한 정적 변수 선언. 선언 위치는 지역 변수와 동일.
11                               // -전역 변수와 같이 초기값 미 설정 시 자동 초기화가 이루어지되, 변수의
12     printf("%d\n", cnt++);    // -초기화는 최초 선언 시만 실행. 함수 종료 후에도 기억 공간이 유지되는
13 }                               // -전역 변수의 성질을 가짐. 또한 전역 변수와 동일한 정적데이터 영역에-
14                               // 저장되고 이로 인해 컴파일 시에 값이 결정되어야 하므로 상수의 대입만-
15                               // 가능하며 실행중에 값이 결정될 수 있는 변수의 대입 자체가 불가.
16
17 void main() {
18     func();
19     func();
20     func();
21     func();
22     func();
23
24     //printf("%d\n", cnt++);    // 통용 범위는 지역 변수와 같이 선언한 블록 내로 제한.
```

<프로젝트 실습>

<1단계>

- 아래 출력 결과와 같이 밑수와 지수를 입력받아 누승(거듭제곱)을 계산하는 알고리즘을 구현하라.
- 단, 밑수와 지수는 0이상의 숫자만 입력되는 것으로 가정.

```
밑수를 입력해라 : 2
지수를 입력해라 : 3
2의 3승은 8 이다.
계속하려면 아무 키나 누르십시오 . . .
```

<2단계>

- <1단계>를 아래의 조건과 출력 결과에 맞추어 수정하라.
- 입력과 누승 연산에 대한 출력을 무한 루프를 통해 구현하되, 밑수가 0이 입력되면 루프 종료.

```
밑수를 입력해라. 밑수가 '0'이면 종료: 2
지수를 입력해라 : 0
2의 0승은 1 이다.

밑수를 입력해라. 밑수가 '0'이면 종료: 2
지수를 입력해라 : 3
2의 3승은 8 이다.

밑수를 입력해라. 밑수가 '0'이면 종료: 3
지수를 입력해라 : 3
3의 3승은 27 이다.

밑수를 입력해라. 밑수가 '0'이면 종료: 0
계속하려면 아무 키나 누르십시오 . . .
```

<3단계>

- 재귀함수를 이용하여 5부터 38까지의 누적 합을 구하는 프로그램을 구현하라.
- 연산 순서는 5부터 시작해서 38로 종료되어야 하며, 실제 연산 순서가 호출 순서의 역순인지 검증하기 위해 연산 결과 출력 전 "5 + 6 + 7 + 8 + ... + 37 + 38"과 동일한 형태로 출력.
- 아래의 출력 형태를 참조하여 구현.

```
<루프 이용 검증> 5부터 38까지의 합은 : 731
5 + 6 + 7 + 8 + 9 + 10 + 11 + 12 + 13 + 14 + 15 + 16 + 17 + 18 + 19 + 20 + 21 + 22
+ 23 + 24 + 25 + 26 + 27 + 28 + 29 + 30 + 31 + 32 + 33 + 34 + 35 + 36 + 37 + 38

<재귀호출을 이용한 누적합> 5부터 38까지의 합은 : 731
계속하려면 아무 키나 누르십시오 . . .
```

6. 정적 배열의 선언과 활용

※ 정적 배열

-선언된 배열의 주소가 정적 즉, 고정되어 있고 재 할당이 불가하며, 정적 배열의 이름은 포인터 상수.

※ 1차원 배열

```
int a[5];
```

a				
0	1	2	3	4

※ 2차원 배열

```
int a[3][2];
```

a					
0		1		2	
0	1	0	1	0	1

	0	1
0		
1		
2		

※ scanf_s(서식 문자열, [&]변수, 출력 버퍼크기)

-기존 scanf함수의 오버플로우 방지를 목적으로 %c 또는 %s에 한하여 버퍼크기를 지정하는 인수 추가.

※ 정적 배열의 이름

-정적 배열의 이름은 포인터 상수로서 배열 arr에 대하여 arr과 &arr은 동일. 즉, 정적 배열의 이름은 정적 포인터 상수
이므로 값 부가 대상이 되는 &연산이 불가해야하나, 예외적으로 정적 배열 명에 대한 포인터 상수는 &연산에 대한 값
부를 인정. 단, 배열 명에 단독 사용 시만 가능하며 배열 명에 대한 연산 후의 &연산은 제외.

정리하자면 정적 배열명은 단독 사용 시는 포인터 상수로 값 부가 인정되지 않으나, 정적 배열 명에 다른 포인터 연산
없이 &연산만 적용한 경우는 예외적으로 값 부를 인정.

-"scanf_s("%d", arr);" 과 "scanf_s("%d", &arr);"은 동일한 표현.

1) 1차원 배열

(1) 정적 배열의 선언과 초기화

```
1  #include <stdio.h>
2
3  void main() {
4      int ar[5];      // 1차원 정적 배열에 대한 선언. -
5                      // 형식 : 타입명 배열변수명[배열크기].
6
7      ar[0] = 1;      // 배열변수에 대한 선언 후 초기화.
8      ar[1] = 2;
9      ar[2] = 3;
10     ar[3] = 4;
11     ar[4] = 5;
12
13     printf("%d %d %d %d %d\n", ar[0], ar[1], ar[2], ar[3], ar[4]);
14
15     for (int i = 0; i < 5; i++)    // 루프를 활용한 초기화.
16     {
17         ar[i] = 5 - i;
18     }
19
20     for (int i = 0; i < 5; i++)
21     {
22         printf("%d ", ar[i]);
23     }
24     puts("");
25 }
```


(2) 배열 선언 시 초기화

```
1  #include <stdio.h>
2
3  void main() {
4      int ar1[5] = { 1,2,3,4,5 }; // 배열 선언 시 바로 초기화. 중괄호로 된 초기화 블록으로 묶어 요소를 일표로
5      int ar2[5];                // -구분. 단, 최초 선언 시에만 가능하며 13행과 같이 선언 후에는 불가.
6      int ar3[] = { 6,7,8,9,10 }; // 초기화 블록 지정 시 배열크기 생략 가능.
7      char stAr1[5] = { 'a','b','c','d' }; // 문자 배열에 대한 초기화의 경우 문자의 끝을 점검하기 위해 -
8                                          // 문자열의 맨 끝에 널(NULL) 종료문자('\0') 자동삽입. 단, 최초-
9                                          // 초기화 블록을 이용한 선언 시에만 널문자 자동삽입. 따라서 -
10                                         // 배열크기를 지정할 경우 널 종료문자를 포함하여 설정.
11     char stAr2[] = "Babo";          // 문자열의 경우 당행과 같이 초기화 블록을 생략하고 쌍따옴표로 묶어
12                                     // -초기화 가능.
13
14     //ar2[5] = { 6,7,8,9,10 };
15
16     for (int i = 0; i < 5; i++) printf("%d ", ar1[i]);
17     puts("");
18
19     for (int i = 0; i < 5; i++) printf("%d ", ar3[i]);
20     puts("");
21
22     for (int i = 0; i < 5; i++) printf("%c", stAr1[i]); // 배열 순회를 통한 문자 요소 출력.
23     puts("");
24     printf("%s\n", stAr1); // 문자열 형태의 배열의 경우 당행과 같이 문자열 서식과 배열 이름(포인터상수)만
25     printf("%s\n", stAr2); // -전달하면 널 종료문자로 끝 점검이 되어 21행과 같이 배열 순회없이 전체 요소값
26                             // -참조 가능. 숫자의 경우 2자리 이상의 연속된 데이터를 나열하면 구분이 모호하여
27                             // -개별 인덱스로 일일이 접근해야 하나, 문자의 경우 연속적으로 나열하여도 구분이
28                             // -모호하지 않으므로 끝 점검만 된다면 일일이 배열을 순회할 필요가 없음.
29     char stAr3[5];
30
31     stAr3[0] = 'A', stAr3[1] = 'B'; // 선언 시 초기화 블록을 사용하지 않고 이와 같이 선언 후 요소를 초기화
32     printf("%s\n", stAr3);          // -하는 경우 널 문자가 자동삽입되지 않아 끝 점검이 되지 않고 이에 따라
33                                     // -초기화되지 않은 영역의 쓰레기값까지 출력. 따라서 문자열 배열의 끝을
34                                     // -지정하기 위해 32행과 같이 널 종료문자를 직접 삽입해야 함.
35     stAr3[2] = '\0';
36     printf("%s\n", stAr3);
37
38     //ar2 = ar1; // 정적 배열명은 상기한 바와 같이 포인터 상수이며 이에 따라 직접 대입 불가하여 배열끼리
39                 // -복사 불가. 따라서 40행과 같이 각 요소를 직접 대입해야만 가능. 또한 정적배열의 경우
40                 // -배열 크기에 대한 재할당 불가.
41
42     for (int i = 0; i < 5; i++) ar2[i] = ar1[i];
43     for (int i = 0; i < 5; i++) printf("%d ", ar2[i]);
44     puts("");
45 }
```

(3) 배열의 할당과 크기

```
1  #include <stdio.h>
2
3  void main() {
4      int n = 5;
5      //int ar1[n];           // 정적 배열의 경우 배열할당 시 그 크기를 실행시에 결정이
6                              // -되는 변수로 설정 불가.
7      int ar[] = { 1,2,3,4,5 };
8      char stAr[] = "Babo PSY";
9
10     int arsize = sizeof(ar); // 일반 변수를 대상으로 sizeof()연산자를 이용하여 그 크기를 조사하면
11                              // -그 타입 크기가 조사되는 것으로 보아 배열명 ar가 포인터 상수이므로
12                              // -포인터의 타입 크기가 조사될 것 같지만 그렇지 아니하고 sizeof() -
13                              // 연산자의 대상이 정적배열명인 경우 배열 전체 크기가 byte단위로 리턴됨.
14     int stArSize = sizeof(stAr);
15
16     printf("%d\n", arsize);
17     printf("%d\n\n", stArSize);
18
19     for (int i = 0; i < arsize / sizeof(int); i++) printf("%d ", ar[i]);
20     puts("");
21
22     for (int i = 0; i < stArSize / sizeof(stAr[0]); i++) printf("%c", stAr[i]);
23     puts("");
24 }
```

2) 다차원 배열

(1) 다차원 배열의 선언과 초기화

```
1  #include <stdio.h>
2
3  void main() {
4      int ar1[3][2] = {          // 2차원 배열에 대한 초기화.
5          {1,2},
6          {3,4},
7          {5,6},
8      };
9      int ar2[2][3][2] = { { {1,2},{3,4},{5,6} }, { {7,8},{9,10},{11,12} } }; // 3차원 배열에 대한 초기화.
10
11     for (int i = 0; i < 3; i++)
12     {
13         for (int j = 0; j < 2; j++)
14         {
15             printf("%d ", ar1[i][j]);
16         }
17         puts("");
18     }
19     puts("\n");
20
21     for (int i = 0; i < 2; i++)
22     {
23         for (int j = 0; j < 3; j++)
24         {
25             for (int k = 0; k < 2; k++)
26             {
27                 printf("%2d ", ar2[i][j][k]);
28             }
29             puts("");
30         }
31         puts("");
32     }
33     puts("\n");
34 }
35
```


(2) 다차원 배열의 크기 조사

그림(6-2-2-1)

ar1[3][2]							
주소부	1차 첨자	0		1		2	
	2차 첨자	0	1	0	1	0	1
값 부							

그림(6-2-2-2)

ar2[2][3][2]													
주소부	1차 첨자	0						1					
	2차 첨자	0		1		2		0		1		2	
	3차 첨자	0	1	0	1	0	1	0	1	0	1	0	1
값 부													

```
1  #include <stdio.h>
2  /*
3   < 정적배열의 구성 >
4   정적배열은 그림(6-2-2-1)과 같이 상위 첨자부가 바로 밑 계층의 하위 첨자부로 구성. 따라서 각 계층의
5   크기를 조사하고자 한다면 해당 계층부터 맨 밑 계층까지의 전체 크기를 조사한 후 그 값을 바로 밑 계층
6   부터 바닥 계층까지의 전체 크기로 나누면 확인 가능.
7   예를 들어 그림(6-2-2-1) 배열 ar1의 1차 첨자 크기 3을 조사하고 싶다면 1차 첨자 계층부터 맨 바닥 계층
8   까지의 전체 크기가 6이고 바로 밑 계층인 2차 첨자 계층부터 바닥 계층까지의 전체 크기가 2이므로 이를
9   나누면 1차 첨자 계층의 크기인 3이 조사.
10
11  < 정적배열 크기 조사 >
12  sizeof()연산자의 대상이 배열일 경우 항상 대상 계층의 바로 밑 계층부터 바닥 계층까지의 전체 크기가
13  조사됨. 참고로 정적배열과 동적배열은 그 구조가 달라 동적배열의 크기 조사는 sizeof()연산자로 조사
14  불가하여 _msize()함수를 이용.
15
16  < 예시 > - int ar1[3][2]
17  sizeof(ar1)      : ar1배열명 바로 밑 계층인 1차 첨자부 부터의 전체 크기 조사. => 3 * 2 * 4(타입크기)
18  sizeof(ar1[0])   : ar1[0] 바로 밑 계층인 2차첨자부 부터의 전체 크기 조사.    => 2 * 4(타입크기)
19  sizeof(ar1[0][0]) : ar1[0][0] 바로 밑 계층인 일반변수의 값부 하나의 크기 조사. => 4(타입크기)
20
21  따라서 정적배열에 대한 임의 계층의 첨자 크기를 알고 싶다면 sizeof()연산자로 바로 위 상위 계층을 sizeof()
22  연산자의 대상으로 전달하여 바로 밑 계층인, 조사하고자 하는 계층부터의 전체 크기가 리턴될 것이고
23  조사하고자 하는 임의 계층을 sizeof()연산자의 대상으로 전달하면 바로 밑 계층부터의 전체 크기가 리턴되므로
24  이를 나누면 해당 계층의 첨자 크기 조사 가능.
25
26  ※ 각 계층의 첨자 크기 = sizeof(해당 계층 바로 상위 계층) / sizeof(해당 계층)
27  */
```

```

29 void main() {
30     int ar1[3][2] = {
31         {1,2},
32         {3,4},
33         {5,6},
34     };
35     int ar2[2][3][2] = { { {1,2},{3,4},{5,6} }, { {7,8},{9,10},{11,12} } };
36
37     printf("%d\n", sizeof(ar1));          // ar1배열 전체에 대한 크기 조사.
38     printf("%d\n", sizeof(ar1[0]));      // ar1배열의 1차 첨자 바로 밑 계층부터의 전체 크기 조사.
39     printf("%d\n\n", sizeof(ar1[0][0])); // ar1배열 2차 첨자 바로 밑 계층인 일반 변수 하나의 크기 조사.
40
41     printf("%d\n", sizeof(ar2));          // ar2배열 전체에 대한 크기 조사.
42     printf("%d\n", sizeof(ar2[0]));      // ar2배열의 1차 첨자 바로 밑 계층부터의 전체 크기 조사.
43     printf("%d\n", sizeof(ar2[0][0]));   // ar2배열의 2차 첨자 바로 밑 계층부터의 전체 크기 조사.
44     printf("%d\n\n", sizeof(ar2[0][0][0])); // ar2배열 3차 첨자 바로 밑 계층인 일반 변수 하나의 크기 조사.
45
46
47     for (int i = 0; i < sizeof(ar1) / sizeof(ar1[0]); i++)
48     {
49         for (int j = 0; j < sizeof(ar1[0]) / sizeof(ar1[0][0]); j++)
50         {
51             printf("%d ", ar1[i][j]);
52         }
53         puts("");
54     }
55     puts("\n");
56
57
58     for (int i = 0; i < sizeof(ar2) / sizeof(ar2[0]); i++)
59     {
60         for (int j = 0; j < sizeof(ar2[0]) / sizeof(ar2[0][0]); j++)
61         {
62             for (int k = 0; k < sizeof(ar2[0][0]) / sizeof(ar2[0][0][0]); k++)
63             {
64                 printf("%2d ", ar2[i][j][k]);
65             }
66             puts("");
67         }
68         puts("");
69     }
70     puts("\n");
71 }
72

```


(3) 행렬의 활용

```
1  #include <stdio.h>
2
3  void main() {
4      int ar[5][5];
5
6      for (int i = 0, v = 1; i < sizeof(ar[0]) / sizeof(ar[0][0]); i++)
7      {
8          for (int j = 0; j < sizeof(ar) / sizeof(ar[0]); j++)
9          {
10             ar[j][i] = v++;
11          }
12      }
13
14      for (int i = 0; i < sizeof(ar) / sizeof(ar[0]); i++)
15      {
16          for (int j = 0; j < sizeof(ar[0]) / sizeof(ar[0][0]); j++)
17          {
18             printf("%2d ", ar[i][j]);
19          }
20          puts("");
21      }
22      puts("");
23  }
```

(4) 문자열 배열

```
1  #include <stdio.h>
2
3  void main() {
4      char str[11]; // 문자열 배열 선언. 널 종로문자('\0')를 포함한 배열 선언에 유의.
5
6      printf("영문 10자 한글5자 이내의 공백을 제외한 문자를 입력하세요^^ : ");
7      scanf_s("%s", str, sizeof(str)); // scanf_s() 함수의 실인수에 문자변수(%c) 또는 문자열 배열(%s)이 오는 경우 이와 같이 출력버퍼(Buffer)
8      // 크기를 지정하는 인수가 추가되어야 함. 단, 출력버퍼의 크기는 널 문자를 포함해야 함에 유의.
9      printf("입력한 문자열 출력 : %s\n\n", str); // scanf_s() 함수의 인수는 주소(포인터)가 대상이 되어 일반 변수의 경우 '&'연산자를 이용한 주소를 전달
10     // 해야 하나 배열의 경우 배열명 자체가 포인터 상수이므로 배열명 자체를 직접 전달. 또한 함수의 종로는
11     // 개행문자로 인식하지만 문자열의 끝은 공백, 탭, 개행문자로 인식하므로 공백을 포함한 문자열의 전달이
12     // 불가.
13     // scanf_s() 함수는 입력 버퍼로 전달된 문자열들을 출력버퍼로 전달할 때 개행문자를 포함하지 않는 특성과
14     // 입력 버퍼로부터 전달받은 데이터를 출력버퍼에 순서대로 저장하고 출력 버퍼의 끝에 항상 널 종로문자를
15     // 삽입하여 배열에 전달하는 특성을 지님.
16     // 오버플로우(Overflow)는 출력 버퍼 크기보다 입력된 문자열이 많아 출력버퍼에 다 들어가지 못하고 입력
17     // 버퍼에 문자열이 남게되는 현상을 의미하는데 scanf_s() 함수는 오버플로우(Overflow) 여부를 입력버퍼의
18     // 값을 출력버퍼로 전달한 후 출력버퍼의 공간이 가득차 출력버퍼의 끝에 널 종로문자를 넣을 수 없는
19     // 경우를 오버플로우로 판단. 이러한 오버플로우가 발생했을 때에는 가득찬 출력버퍼의 값을 비우고 널 종로
20     // 문자만 배열에 전달되는 것을 확인 가능.
21     // 오버플로우(Overflow)가 발생하지 않는다는 전제하에 연속적으로 scanf_s() 함수를 호출하면 입력버퍼로
22     // 부터 개행문자를 제외한 문자열들을 출력버퍼로 전달후 출력버퍼에 끝에 널 종로문자를 삽입하여 항상
23     // 정상적으로 실행되는 것을 확인가능.
24     // 반면 연속적인 scanf_s() 함수 호출을 할때 입력버퍼의 문자열이 출력버퍼에 전달된 후 출력버퍼가 가득차
25     // 그 끝에 널 종로문자를 삽입할 수 없는 경우 오버플로우로 인식하여 출력버퍼에 전달된 값들을 비워 버리고
26     // 널 종로 문자 하나만 전달되는 것을 확인 가능하며, 이후 호출되는 scanf_s() 함수 호출시에 개행 문자를
27     // 제외한 문자열이 입력버퍼에 존재한다면 출력버퍼로 전달되어 널 종로문자와 함께 배열에 전달 됨과 동시에
28     // 입력 기회를 잃어버리는 현상을 관측 가능.
29     // 따라서 이와같은 오버플로우 발생에 따른 문제점을 해결하려면 입력 버퍼의 끝에 항상 개행문자가 남게
30     // 되는 특성을 이용 scanf_s() 함수를 사용하여 문자열을 연속적으로 입력받는 경우 33행과 같이 개행문자를
31     // 추출할때 까지 입력버퍼를 비움으로써 해결 가능.
32
33     while (getchar() != '\n'); // 입력 버퍼 비우기.
34     // getchar() : 버퍼를 이용해 한 문자를 입력받되 입력 버퍼의 개행문자를 포함하여 출력버퍼로 전달.
35
36
37     printf("영문 10자 한글5자 이내의 공백을 포함한 문자를 입력하세요^^ : ");
38
39     gets_s(str, sizeof(str)); // get_s(대상 배열, 버퍼크기) : scanf_s() 함수와 같이 버퍼를 이용하여 입력을 받는다는 점은 동일하지만
40     printf("입력한 문자열 출력 : %s\n\n", str); // scanf_s() 함수와 다른점은 입력의 대상이 문자열만 가능하여 입력 서식이 필요치 않으며 개행 문자만
41     // 문자열의 끝으로 인식하여 공백을 포함한 문자열 입력이 가능. 또한 scanf_s() 함수는 입력버퍼의
42     // 개행문자를 포함하지 않는 반면 gets_s() 함수는 getchar() 함수와 마찬가지로 입력버퍼의 개행문자도 포함
43     // 하여 출력버퍼로 전달하는 특성을 지님.
44     // 동작방식은 scanf_s() 함수와 동일하지만 가장 큰 차이점은 오버플로우 발생시 scanf_s() 함수와 달리 실행
45     // 오류로 처리하여 프로그램을 강제 종료하는 특성.
46 }
```

<프로젝트 실습>

<1단계>

```
int ar1[10] = { 1,2,3,4,5,6,7,8,9,10 };  
int ar2[10] = { 10,9,8,7,6,5,4,3,2,1 };
```

-위와 같은 배열 두 개를 선언하고 두 배열의 값을 맞교환하여 아래와 같이 출력하는 프로그램을 구현하라.

```
ar1 배열 :10 9 8 7 6 5 4 3 2 1  
ar2 배열 : 1 2 3 4 5 6 7 8 9 10  
계속하려면 아무 키나 누르십시오 . . .
```

<2단계>

```
int ar[20];
```

-위와 같은 배열을 선언하고 5부터 24까지의 자연수를 저장하여 아래와 같이 출력하는 프로그램을 구현하라.

-단, 선언한 배열변수 ar과 인덱스 변수 외에는 추가 변수 설정 불가.

```
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24  
계속하려면 아무 키나 누르십시오 . . .
```

<3단계>

-크기가 5인 1차원 배열을 선언하고 배열의 각 요소를 아래와 같이 입력받아 출력하는 프로그램을 구현하라.

```
배열의 1번째 요소 입력 : 85  
배열의 2번째 요소 입력 : 77  
배열의 3번째 요소 입력 : 93  
배열의 4번째 요소 입력 : 80  
배열의 5번째 요소 입력 : 65  
  
입력된 정수 중 최대값 : 93  
입력된 정수 중 최소값 : 65  
입력된 정수 총합 : 400  
계속하려면 아무 키나 누르십시오 . . .
```

<4단계>

-5행 5열 크기의 2차원 배열을 선언하여 아래와 같이 출력하는 프로그램을 구현하라.

-단, 배열에 값 저장 시에는 1부터 25까지 순서대로 저장.

```
25 24 23 22 21
20 19 18 17 16
15 14 13 12 11
10 9 8 7 6
5 4 3 2 1

계속하려면 아무 키나 누르십시오 . . .
```

<5단계>

-배열크기가 각 5인 2차원 배열을 선언하여 값을 저장하고 아래와 같이 출력하는 프로그램을 구현하라.

```

1
2 3
4 5 6
7 8 9 10
11 12 13 14 15

계속하려면 아무 키나 누르십시오 . . .
```

<6단계>

-5행 5열 크기의 2차원 배열을 선언하여 아래와 같이 출력하는 프로그램을 구현하라.

-단, 배열에 값 저장 시에는 1부터 25까지 순서대로 저장.

```
1 2 3 4 5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9

계속하려면 아무 키나 누르십시오 . . .
```