

C언어의 기본구조

2018년 10월 20일 토요일 오후 3:13

1. 헤더 파일

C언어 문장은 `#include<stdio.h>`(:헤더 파일의 한 종류 : 기본 입출력함수 및 각종 수많은 함수들을 가짐) 문장으로 시작 : C언어는 모든 수행하는 명령들이 함수로 구성되어있음 -> C언어 명령은 함수로 정의되어있기에 그걸 쓰면 됨

: ex) `printf()` 를 수행하는건 `stdio.h`파일이 있기에 가능

== `#include<stdio.h>` ->이걸 안쓰면 C언어 실행이 안됨 !!!!! 무조건 써줘야함 !!!

Printf() 포맷

2018년 10월 20일 토요일 오후 5:09

변환 문자열	의미
%o	8진 정수 형식으로 출력
%d	10진 정수 형식으로 출력
%ld	long형 10진 정수 형식으로 출력
%x	16진 정수 형식으로 출력
%u	부호 없는 10진 정수 형식으로 출력
%f	소수점 형식으로 출력
%e %E	지수 형식으로 출력
%g %G	%e와 %f 중 짧은 쪽, 소수점에 이어지는 0은 생략
%c	문자 형식으로 출력
%s	문자열 형식으로 출력

자료형	서식	내용	사용 예	결과
문자형	%c	문자 1개	printf("%c",'B');	B
문자열형	%s	문자열	printf("%s","Good");	Good
정수형	%d	십진수	printf("%d",110);	110
	%o	8진수	printf("%o",110);	156
	%x	16진수	printf("%x",110);	6e
실수형	%f	float형	printf("%f",12.789);	12.789000
	%e	지수형	printf("%e",12.789);	1.278900e+01
unsigned	%u	양수값만 표현	printf("%u",20);	20

1. C언어 자료형

2018년 10월 20일 토요일 오후 3:35

자료형 : 기본적으로 프로그램은 자료를 입력받아 처리 후 결과를 만들어 냄

-> 컴퓨터는 0,1밖에 모름 -> 현실세계 데이터를 컴퓨터에 인식시키려면 디지털화(0,1로 바꿔)해서
컴퓨터 내부에 저장시켜야함 -> 자료마다 다르게 규칙을 만들어 내부에 표현하는 규칙

1. 자료형(data와 type이 있음) : 문자, 정수, 실수, 숫자 등 매우 다양한 자료형

각각의 자료형은 메모리를 차지하는 크기(Byte)가 다름 -> 표현할수있는 수의 범위가 다름

기본형

- 문자형 :
 - Char :
 - unsigned char : 문자형을 수로 표현, 음수표현 X
- 정수형 :
 - int
 - unsigned int
 - short
 - unsigned short
 - long
 - unsigned long
- 실수형 :
 - float
 - double
- void형 : void

기타

- 배열
- 구조체(struct)
- 공용체(union)
- 열거형(enum)
- 포인터형

각 자료형의 크기 및 표현가능한 수의 범위

자료형	자료형예약어	기억공간의 크기	표현가능한 수의 범위
문자형	char	1 Byte	-128 ~ +127
	unsigned char	1 Byte	0 ~ +255
정수형	int	2 또는 4 Byte	-32768 ~ +32767(2B의 경우)
	unsigned int	2 또는 4 Byte	0 ~ +65535(2B의 경우)
	short	2 Byte	-32768 ~ +32767
	unsigned short	2 Byte	0 ~ +65535
	long	4 Byte	-2147483648 ~ +2147483647
	unsigned long	4 Byte	0 ~ +4294967295
실수형	float	4 Byte	$3.4 \times 10^{-38} \sim 3.4 \times 10^{+38}$
	double	8 Byte	$1.7 \times 10^{-308} \sim 1.7 \times 10^{+308}$

2. 수의 표현

컴퓨터 내부 수 저장 방법 : 정수형, 실수형 다름

정수형 - 고정소수점방식(Fixed Point)

실수형 - 부동소수점방식(Floating Point) : 매우 큰 수, 매우 작은 수를 표현하기 편리

-> 정규화를 시켜야함 (모든 유효 숫자를 소수 첫째자리로 옮기는 것) -> 나중에 100을 곱해줘야 처음 데이터 값과 같아짐

3. 문자 표현

문자->숫자 : 아스키코드(1byte)로 인식 (자바는 유니코드(2byte))

-아스키 : 대문자 A : 65, 소문자 a : 97

2. 문자 상수, 문자열 상수

2018년 10월 20일 토요일 오후 4:08

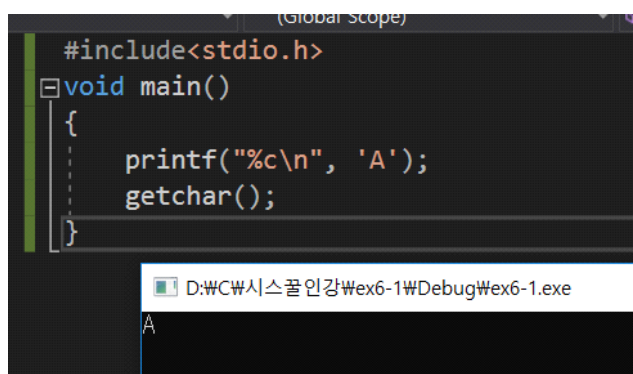
변수 : 메모리 위치의 이름

상수 : 변수 메모리 위치에 저장되는 값

-> 변수 값은 프로그램에서 얼마든지 변할 수 있으나, 상수는 프로그램에 의해 절대 변하지 않음

1. 문자 상수 - visual studio2017 ex6-1 project

: 문자 1개의 의미 홀따옴표 ' ' 안에 표시



```
#include<stdio.h>
void main()
{
    printf("%c\n", 'A');
    getchar();
}
```

The screenshot shows the Visual Studio 2017 IDE. The top part displays the source code of a C program. The code includes the standard input/output header, defines the main function, and uses printf to print the character 'A' followed by a newline. The bottom part shows the output window where the character 'A' has been printed.

<제어문자>

: Back slash(\)와 함께 사용 되어 하드웨어를 제어하는 역할을 하는 문자

- 종류 :

- \n : new line
- \t : tab
- \a : Beep 소리
- \r : Carriage Return
- \f : Form Feed

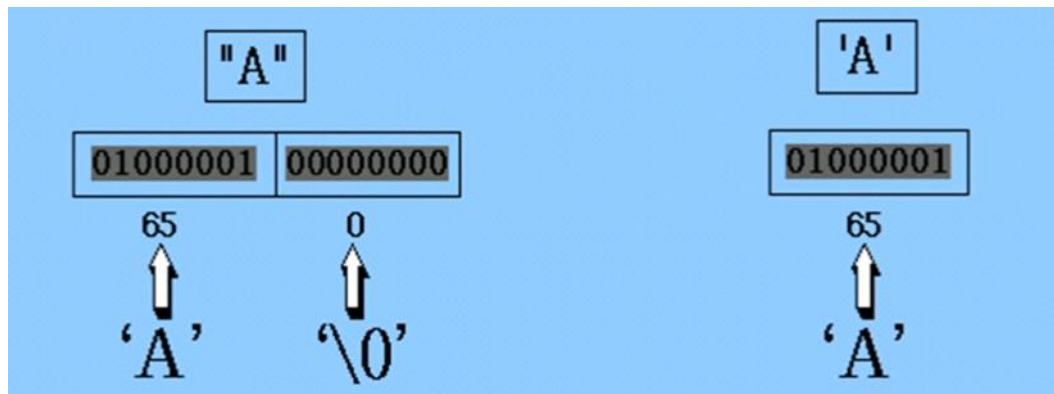
3. 문자열 상수 (String Constant)

- 의미 : 문자 여러 개, 쌍 따옴표 " " 안에 표시

<문자와, 문자열은 확실히 구분됨>

'A' : 1byte

"A" : 2byte



Ex6-2

결과

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

```
void main() {
```

```
    printf("%dWn", 12);
```

```
    printf("%dWn", 014);
```

```
    printf("%dWn", 0xC);
```

```
    printf("%dWn", 'W014');
```

```
    printf("%dWn", 'WxC');
```

```
    getchar();
```

```
}
```

12

12

12

12

12

3. 정수형, 실수형, 열거형 상수

2018년 10월 20일 토요일 오후 4:32

〈정수형 상수〉: Integer Constant
: 소수점이 없는 양수, 음수를 의미

C언어에서 표현법

진법	수학적 표현	C언어에서의 표현1 (정수 상수임)	C언어에서의 표현2 (문자 상수임)
8진수	(14) ₈	014	'\014'
10진수	12	12	12
16진수	(C) ₁₆	0xC	'\xC'

정수 출력: %s

```
void main()
{
    printf("%s\n", "Hello world!");
    getchar();
}
```

D:\C#\시스템관리\Wex6-1\Debug\Wex6-1.exe
Hello world!

〈실수형 상수〉: Float Constant
: 부동소수점이라 불리는 실수형 상수 : 소수점이 있는 수를 의미

C언어에서 표현법

일반적인 표현	지수형식의 표현
12.78	1.278000e+001
456.3789	4.563789e+002
0.45678	4.567800e-001

실수 출력: %e

```
void main()
{
    printf("%e\n", 12.78);
    printf("%e\n", 456.3789);
    printf("%e\n", 0.45678);
    getchar();
}
```

D:\C#\시스템관리\Wex6-1\Debug\Wex6-1.exe
1.278000e+01
4.563789e+02
4.567800e-01

〈열거형 상수〉: Enumerated Constant

: 사용자가 정의해 사용하는 상수

: 내부적으로 정수와 동일하게 취급

: 열거형 상수는 이름을 열거하여 0부터 값이 붙어지며,

이름으로 정수 대신 사용됨

- 소스 코드에서 다음과 같이 쓰였다고 한다면, 프로그램에서는 HANA를 0으로 DUL 을 1로 SET을 2로 NET을 3으로 인식한다.
- 보기처럼 HANA, DUL, SET, NET 과 같은 상수를 열거형 상수라고 한다.

```
enum DEF_num{ HANA, DUL, SET, NET}
대응값-> 0 1 2 3
```

4. 변수 개념 / 문자형 변수

2018년 10월 20일 토요일 오후 4:58

<변수>

- : 주 메모리의 위치를 의미하는 프로그램상의 명칭
- : 프로그램 상에서는 일정한 이름, 메모리 상에서는 주소로

★<변수명(identifiers)을 만드는 규칙>

1. 변수명에 올 수 있는 문자는 영문자와, 숫자, 밑줄(`under score: _`)만 올 수 있다.
2. 변수명은 영문자나 밑줄(`under score: _`)로 시작할 수 있고, 숫자로 시작할 수 없다.
3. 변수명은 대소문자를 구분하므로, Tot와 tot는 서로 다른 변수이다.
4. 변수명은 공백을 허용하지 않는다.
5. `?`, `!`, `#`과 같은 특수 문자는 변수명에 사용할 수 없다.
6. 예약어는 변수명으로 사용할 수 없다.

int, short, ...

잘못된 변수명의 예	올바른 변수명의 예
① Wabc	① abc
② 3sum	② sum3
③ 2hap	③ hap
④ Fall78	④ Fall78
⑤ int	⑤ IN_T

<문자형 변수> : Character Variable

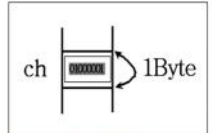
- : 문자형 상수 값이 저장되는 변수 - 1Byte 기억공간 차지
- : 문자형 변수로 선언 되어도 C언어에서는 1Byte 정수로 표현 가능
- 1Byte 표현 가능 범위: $-128 \sim +127$ ($0 \sim 255$)

<특징>

`char ch ;` 라고

선언했다면 메모리 할당은 그림과 같다.

- 1Byte 기억공간으로 표현할 수 있는 수의 범위는 $-128 \sim +127$ 이지만 unsigned로 선언 되면 음수부분을 저장할 기억공간을 양수를 표현하는데 할애하므로 127의 약 2배인 $0 \sim 255$ 까지다.



```
void main()
{
    char ch;
    ch = 'A';
    printf("문자 %c의 아스키코드 값은 %d입니다.\n", ch, ch);
    getchar();
}
```

D:\C#\시스템인강\ex6-1\Debug\ex6-1.exe

문자 A의 아스키 코드 값은 65입니다.

5. 정수형 변수 / 실수형 변수

2018년 10월 20일 토요일 오후 5:10

<정수형 변수(Integer Variable) >

: int, short, long

정수형 변수는 앞에서 배운 정수형 상수의 값이 기억되는 메모리 위치의 이름이다. C언어의 정수형은 크게 int, short, long 3가지와 각각의 unsigned형까지 포함해 6개로 구분된다. 컴퓨터의 특성을 생각하지 않고 정수를 선언하고자 한다면 short를 써주면 된다. C언어 정수형의 자료형에 왜 이렇게 많은 자료형이 존재할까? 예를 들어, 작성한 프로그램에 정수형이 필요한데 2Byte 기억공간의 최대치인 +32767을 넘어가는 부분이 있다고 가정해보자. 그렇다면 4Byte인 long형으로 선언할까? 아니다! 이때 음수를 표현하지 않는다는 가정 하에 우리는 unsigned를 사용하면 된다. unsigned로 선언 하게 되면, short나 int와 같이 2Byte만을 차지하면서 좀 더 큰 수를 표현할 수 있기 때문이다.

① 선언방법

```
int 변수명 ;
unsigned int 변수명 ;

또는

short 변수명 ;
unsigned short 변수명 ;

또는

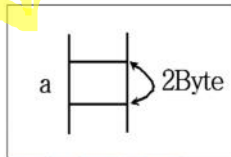
long 변수명 ;
unsigned long 변수명 ;
```

② 특징

- int a; 라고 선언
했다면 메모리 할당은 그림과 같다. (작은 머신에서)

- 2Byte 기억공간으로 표현할 수 있는 수의 범위는 -32768 ~ +32767이지만 unsigned로 선언 되면 음수부분을 저장할 기억공간을 양수를 표현하는데 할애하므로 32767의 약 2배인 0 ~ 65535 까지도.

- long형의 경우는 4Byte 기억공간의 크기를 갖는다.



<실수형 변수(Integer Variable) >

: 소수점이 있는 수

: 소수점표현 & 지수형식 표현

① 선언 방법

```
float 변수명 ;
double 변수명 ;
```

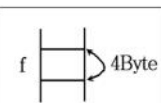
② 특징

- float f; 라고 선언

했다면 메모리 할당은 그림과 같다.

- 4Byte 기억공간인 float로 표현할 수 있는 실수의 범위는 대략 $-10^{-38} \sim +10^{+38}$ 이다.

- long형의 경우는 8Byte 기억공간의 크기를 가지며 표현할 수 있는 실수의 범위는 대략 $-10^{-308} \sim +10^{+308}$ 이다



<실행>

```
void main()
{
    float f;
    double d;
    f = 56;
    d = 56.1234;

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

    getchar();
}
```

%f서식은 기본적으로 소수점 아래 6자리 까지 나온다. 예제에서 변수 f에 소수점이 없는수 56을 저장했어도 %f서식으로 출력했기 때문에 소수점 아래 6자리까지 0으로 표시된 것이다.

<실행>

```
void main()
{
    short a;
    unsigned int b;
    long c;
    unsigned long d;

    a = 32768;
    b = 32768;
    c = 2147483648;
    d = 2147483648;

    printf("%d %u %ld %lu\n", a, b, c, d);

    getchar();
}
```

D:\WC\시스코 인강\Wex6-1\Debug\Wex6-1.exe

-32768 32768 -2147483648 2147483648

(해설)

~~-32768~~ 32768 ~~-2147483648~~ 2147483648

(해설)

short형의 저장가능한 최대값은 +32767이고 long
형의 저장가능한 최대값은 +2147483648이다. 변수
b와 변수 d의 경우 unsigned형으로 선언되었기 때문
에 short나 long형의 최대값을 넘어가는 수가 저장
가능 하지만 변수 a나 변수 c는 올바른지 못한값
(overflow된 값)이 저장 되게 된다.

※ 출력서식 long형의 경우 l(영문자 L의 소문자)을
붙여주고, unsigned형의 경우 u를 붙여주어야 한다.

6. 열거형 변수 (Enumerated Variable)

2018년 10월 20일 토요일 오후 5:34

1. 열거형 변수

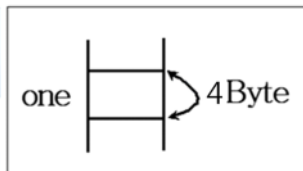
: 열거형 상수를 저장하는 기억 공간의 이름
: 정수형과 동일하게 취급되고 4Byte공간

① 선언방법

```
enum 열거형이름 {열거형 상수들};
enum 열거형이름 변수명 ;
```

② 특징

- 예제처럼 enum DEF_num one; 라고 선언 했다면 메모리 할당은 그림과 같다.



2. 변환 지시자(Conversion Specification)

C언어에서 기본적인 입,출력을 담당하고 있는 `printf()` 함수 및 `scanf()` 함수는 변환지시자(Conversion Specification)라고 불리는 서식을 갖는다. 이 서식들은 입, 출력시에 변수에 저장된 값 또는 입,출력되는 값들이 어떻게 변환되는지를 결정한다

변환지시자

자료형	서식	내용	사용 예	결과
문자형	%c	문자 1개	<code>printf("%c", 'B');</code>	B
문자열형	%s	문자열	<code>printf("%s", "Good");</code>	Good
정수형	%d	십진수	<code>printf("%d", 110);</code>	110
	%o	8진수	<code>printf("%o", 110);</code>	156
	%x	16진수	<code>printf("%x", 110);</code>	6e
실수형	%f	float형	<code>printf("%f", 12.789);</code>	12.789000
	%e	지수형	<code>printf("%e", 12.789);</code>	1.278900e+01
unsigned	%u	양수값만 표현	<code>printf("%u", 20);</code>	20

%l : long형

<실행>

```
void main()
{
    enum DEF_num {HANA, DUL, SET, NET};
    enum DEF_num one;
    enum DEF_num two;
    one = HANA; → 0
    two = DUL; → 1

    printf("%d\n", one);
    printf("%d\n", two);
    printf("%d\n", HANA + DUL + SET + NET);
    getchar();
}
```

0 1 2 3
0+1+2+3 = 6

<예제2>

```
void main()
{
    enum DEF_num {HANA, DUL, SET, NET};
    enum DEF_num temp1;
    int jungsu;
    jungsu = DUL;
    temp1 = NET;

    printf("%d\n", jungsu);
    printf("%d\n", temp1);
    getchar();
}
```

0 1 2 3
1 3

DUL은 지금 1이니까 Int형 변수 jungsu에 넣을 수 있음

7. 자료형 변환

2018년 10월 20일 토요일 오후 5:52

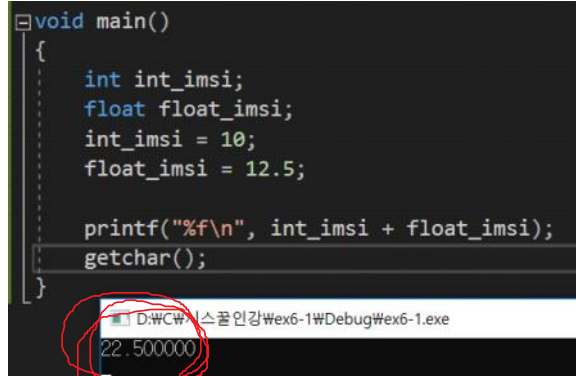
1. 자동 형변환(Automatic Type Conversion) = 묵시적 형변환

- C언어는 서로 다른 자료형의 연산식에서 둘 중 한 자료형으로 나머지 한쪽의 자료형을 바꾸어 갈게한 후 결과를 만들어 낸다.
- 이 때 대부분은 작은 기억장소의 자료형이 큰 기억장소의 자료형으로 변환된 후 연산하게 되는데 이것을 자동 형변환 또는 묵시적 형변환이라고 부른다.
- (큰 쪽으로의 변환은 데이터의 손실이 발생하지 않으나 작은 쪽으로의 변환은 데이터 값이 잘려 나가므로 손실이 발생하기 때문이다.)

<실행>

```
void main()
{
    int int_imsi;
    float float_imsi;
    int_imsi = 10;
    float_imsi = 12.5;

    printf("%f\n", int_imsi + float_imsi);
    getchar();
}
```



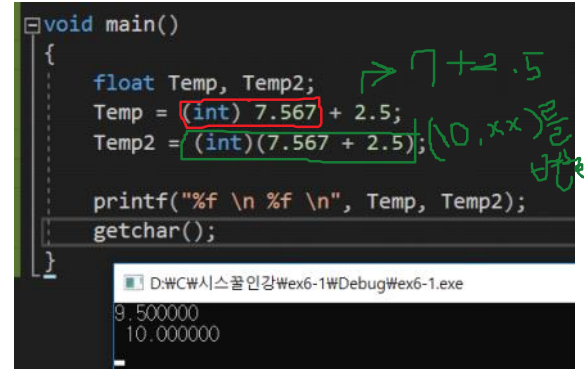
2. 명시적 형변환(Explicit Type Conversion)

- 프로그램에서 프로그래머가 강제적으로 자료형을 변환시키는 것을 의미하며 Cast 연산자를 사용해 변환 한다. 사용 방법은 (자료형) 부분에 변화시키고자하는 자료형을 써주며 다음 수식을 써주면 된다.

(자료형) 수식 :

<실행>

```
void main()
{
    float Temp, Temp2;
    Temp = (int) 7.567 + 2.5;
    Temp2 = (int)(7.567 + 2.5);
    printf("%f \n %f \n", Temp, Temp2);
    getchar();
}
```



Temp = (int) 7.567 + 2.5 ; 에서 7.567 만 int형으로 바꾸므로 소수점이 잘려 7이 되고 7 + 2.5의 연산식을 수행하며, Temp2 = (int) (7.567 + 2.5) ; 에서는 전체식에 대해 cast연산자가 적용되므로 먼저 연산 후 정수로 만들어 최종 결과에 소수점이 잘린다.

8. 진법

2018년 10월 20일 토요일 오후 6:04

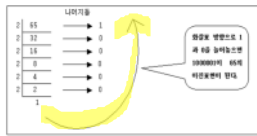
<2진법>

* 사용digit
0, 1

*진법변환

① 10진수를 2진수로

: 2로 나눈 나머지만 나열해준다.

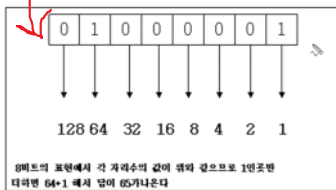


② 2진수를 10진수로

10진수를 2진수로 변환하고자 할 때 2로 나누었으므로 이제는 거꾸로 2를 곱해야한다. 예를 들어, 1000001을 10진수로 하고자 할 때는, 다음 수식처럼 구해되는데 그것보다는 그림처럼 자리수를 외워서 더하면 편리하다

$$1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 65$$

2진수 자리값



<8진법>

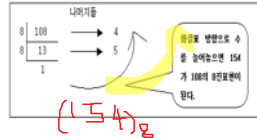
- 봉어빵에는 봉어가 없듯이 8진수에는 8이라는 수가 존재하지 않는다.
- 즉, 8이 되면 이것은 1과0으로 10이라는 새로운 단위가 만들어져야한다.
- 따라서, 사용 가능한 digit는 0에서 7까지의 수이고, 변환하는 방법은 2진수의 변환과 동일하다.

* 사용digit

0, 1, 2, 3, 4, 5, 6, 7

*진법변환

① 10진수를 8진수로



② 8진수를 10진수로

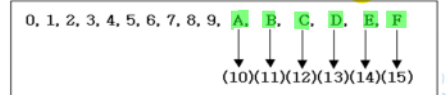
• (154)₈의 10진수 변환은 다음과 같다.

$$1 \times 8^2 + 5 \times 8^1 + 4 \times 8^0 = 64 + 40 + 4 = 108$$

<16진법>

- 앞에서 말했듯이 봉어빵에는 봉어가 없고, 8진수에는 8이 없다, 마찬가지로 16진수에는 16이 없다. 16이 되면 새로운 10으로 단위가 바뀌게 된다.
- 그리고 10진수의 10에 해당하는 개념과 16진수의 10이 겹치게 되므로 16진수에서는 10진수의 10에 해당하는 값의 표기를 알파벳 A를 빌어서 표현하고, 차례로 B, C, D, E, F는 각각 11, 12, 13, 14, 15의 값을 의미한다.

* 사용digit



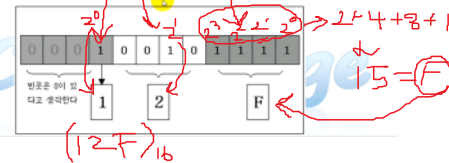
*진법변환

① 10진수를 16진수로

10진수를 16진수로 변환할 때, 앞의 두 방법처럼 해도 되겠지만, 약간 다른 방법으로 시도해보자. 만약 십진수 303을 16진수로 변환한다고 가정 한다면, 먼저 303을 2진수로 만들어보라.

• (303)₁₀ -----> 1 0010 1111 이 된다.

• 그런 다음 이진수로 변환된 수자를 4개씩 묶어보라. 그리고 각 4비트에 해당하는 16진수의 digit를 써주면 쉽게 16진수로 전환할 수 있다.



② 16진수를 10진수로

16진수를 10진수로 변환하는 방법은 앞의 두 방법처럼 각 자리수의 크기만큼 곱해주면 된다.

$$1 \times 16^2 + 2 \times 16^1 + F \times 16^0 = 256 + 32 + 15 = 303$$

9. 표준 입출력 함수

2018년 10월 20일 토요일 오후 6:22

<표준 입출력 함수>

- 데이터를 컴퓨터 내부로 읽어 들이는 것을 **입력(input)**이라 하고, 입력된 데이터를 프로그램에서 실행한 결과를 외부로 내보내는 것을 **출력(output)**이라고 한다.
- 이 때, 입력을 위해 여러 가지 장비를 생각해 볼 수 있다. 예를 들면, 입력 장치로는 키보드, 마우스, 조이스틱 등등이 있겠고, 출력 장비에 대해서도 모니터의 화면, 프린터, 플라터 등이 바로 그것이다.
- 이러한 많은 입.출력 장비 중 표준으로 정해 놓은 장비가 있는데, 표준 입력 장치로는 키보드이고, 표준 출력 장치로는 모니터이다.
- 이를 합쳐서 **표준 입출력장치(Standard Input Output device)**라고 한다.
- C언어에서 반드시 **include하는 헤더 파일 stdio.h의** 의미도 바로 이 표준 입출력 함수들의 원형에 대해서 정의해 놓았기 때문에 이런 이름을 가지게 된 것이다.
- 또한 이 장에서 설명하는 입출력 함수들을 사용하려면, 반드시 **#include<stdio.h>**를 프로그램 머리부에 써 주어야 한다.
- C언어에서는 이러한 표준 입출력 장비를 통해 입출력을 담당하는 함수가 있는데.
- 이를 표준 입출력 함수라 하며, 뒤부분에서 배울 파일 입출력 함수와 비교가 되는 내용이다.
- C언어의 대표적인 입출력 함수는 printf(), scanf()등이 있으며, 그 외의 몇 가지 표준 입출력 함수에 대해서도 그 사용법을 다음절에서 알아보자.

<표준 입출력 함수 종류>

표준 입력 함수	표준 출력 함수
<div>1 getchar(), scanf(), gets(), getch(), 2 3 4 getche(), cgets(), 5 6</div>	<div>1 putchar(), printf(), puts(), 2 3 putch(), cputs(), cprintf() 4 5 6</div>

<입력함수>

1:
2:
3:
4:
5:
6:

<출력함수>

1:
2:
3:
4:
5:
6:

10. 단일 문자 입출력 함수

2018년 10월 21일 일요일 오후 2:38

<단일 문자 출력 함수 putchar(>

단순히 문자 1개를 화면으로 출력하는 함수로 stdio.h 파일에 원형(prototype)이 정의되어 있으므로 프로그램 머리에 프리프로세서 지시자 #include<stdio.h>를 써 주어야 한다. 실제로 putchar()함수는 stdio.h 파일에서 함수 putc()로부터 얻는 매크로로 정의하고 있다.

[형식]

```
putchar ( 문자 ) ;
```

여기서 문자 부분에 올 수 있는 것으로는, 문자 상수, 문자 변수, 제어 문자 등이다.

<실행>

```
void main()
{
    char ch = 'a';
    putchar(ch);
    putchar('\n');
    putchar(97);
    putchar('\n');
    putchar('b');
    putchar('\n');
    putchar(98);
    getchar();
}
```

97 → a의 아스키코드
98 → b의 아스키코드



<단일 문자 입력 함수 getchar(>

- 함수 getchar()는 표준 입력 장치인 키보드로부터 문자 1개를 입력받는 함수이다. 함수 getchar()의 원형은 stdio.h에 정의되어 있으므로, #include<stdio.h>를 반드시 프로그램 머리에 써 주어야 한다.
- 함수 getchar()의 리턴 값은 입력받은 문자의 아스키코드이며, 파일의 끝을 나타내는 EOF는 stdio.h의 헤더 파일에 값이 -1로 정의되었으며, ctrl+z를 입력하면 파일의 끝을 나타내는데 프로그램에서는 파일의 끝을 판단하는데 유용하게 쓰인다.
- 실제로 getchar()함수는 stdio.h 파일에서 함수 getc()로부터 얻는 매크로로 정의하고 있다. 사용형식은 다음과 같다.


사용형식

```
변수 = getchar( ) ;
```

<실행>

```
void main()
{
    int ch;
    ch = getchar(); //값받으려는
    putchar(ch);

    getchar(); //정지목적
    getchar(); //정지목적
}
```



11. 문자열 입출력 함수

2018년 10월 21일 일요일 오후 2:49

<문자열 출력 함수 puts(>

- 함수 puts()은 화면으로 문자열을 출력시킬 때 사용하는 함수이다.
- puts()함수는 그 함수 안에 자체적으로 new line(' \n')을 가지기 때문에 굳이 new line을 해주지 않아도 된다.
- 함수 puts()의 원형은 stdio.h에 정의되어 있으므로, #include<stdio.h>를 반드시 프로그램 머리부에 써 주어야 한다.
- 사용형식은 다음과 같다.

```
puts("문자열");
```

<실행>

```
#include<stdio.h>
void main() {
    char * ptr = "Language";
    puts(ptr);
    getchar();
}
```

-> 포인터 변수가 가리키고 있는 문자열이 찍힘

<실행2>

```
void main()
{
    puts("Hello");
    getchar();
}
```

D:\WC\시스폴인강\wex6-1\Debug\wex6-1.exe

Hello

<실행3>

```
void main()
{
    putchar('Y');
    putchar('e');
    putchar('s');
    putchar('\n');
    puts("YES");

    getchar();
}
```

D:\WC\시스폴인강\wex6-1\Debug\wex6-1.exe

Yes
YES

<문자열 입력 함수 gets(>

- 함수 gets()은 키보드로부터 문자열을 입력받는 입력 함수이다. gets()의 끝 문자인 s를 string 이라 생각하면 쉽게 이해가 될 것이다.
- 함수 gets()의 원형은 stdio.h에 정의되어 있으므로, #include<stdio.h>를 반드시 프로그램 머리부에 써 주어야 한다. 사용형식은 다음과 같다.

```
gets(문자열변수);
```

```
gets_s(문자열변수);
```

C언어고유(표준)문법

visual studio 문법

<실행>

```
void main()
{
    char name[10];
    char address[30];
    puts("이름을 입력하세요.");
    gets_s(name);
    puts("주소를 입력하세요.");
    gets_s(address);
    printf("이름은 %s이고, 주소는 %s입니다.\n", name, address);

    getchar();
}
```

D:\WC\시스폴인강\wex6-1\Debug\wex6-1.exe

이름을 입력하세요.
주소를 입력하세요.
정은이고, 주소는 상현동입니다.

12. C언어 출력 서식 함수 printf()

2018년 10월 21일 일요일 오후 3:15

<출력 서식을 가지는 출력 함수 printf()>

- C언어에서 가장 많이 쓰이는 출력함수로 출력 대상을 화면으로 출력하되 각 자료형에 맞게 출력하는 서식을 가지고 있다.
- 앞의 자료형의 설명에서는 **변환지시자**(Conversion Specification)라는 용어로 설명했다.
- 이 번장에서는 조금 더 자세히 살펴보고자 하자.
- 함수 printf()의 원형은 stdio.h에 정의 되어 있으므로, 반드시 프로그램 머리부에 #include<stdio.h>를 써 주어야 한다.
- 사용형식은 다음과 같다.

사용형식

```
printf("출력 서식", 출력 대상)
```

출력대상은 변수나 상수가 될 수 있고, 출력 대상 없이 출력 서식부분(이때 " " 안에는 문자열만 온다)만도 올 수 있으며 출력 서식만을 다시 보면 다음과 같다. 예제를 통해서 printf()의 출력 서식에 대해 익혀보자

자료형	서식
문자형	%c
정수형	%d, %ld, %o, %lo, %x, %lx, %i, %li
실수형	%e, %f, %g
문자열형	%s

<printf() 함수의 출력 서식 주의 문자>

- printf()의 " "의 출력 서식 부분에 사용 할 때 주의해야 할 문자들이 있다. 다음 예제는 이런 주의해야 할 문자들을 실행해보는 예제로 첫째, 기호 " 와 '는 컴파일러가 출력서식의 "과 혼동 할 수 있으므로, \w" 나 \w' 처럼 **역슬래쉬**를 하나 해주고 " 나 '를 써주면 출력서식의 "과 혼동하지 않고 화면에 출력할 대상으로 이해하고 실행한다.
- 둘째, % 기호는 printf()의 " "의 출력 서식 부분에 사용 할 때, %%처럼 %를 하나 더 써 주어야 한다.
- 셋째, 역슬래쉬(\w)를 printf()의 " "의 출력 서식 부분에 사용 할 때, \w\w처럼 \w를 하나 더 써 주어야 한다.

<실행>

```
void main()
{
    printf("\'\' \n");
    printf("\\" \n");
    printf(" 50%% \n");
    printf(" 제어문자 \\n \n");

    getch();
}
```

D:\WC\시스폴인강\wex6-1\Debug\wex6-1.exe

```
'\''
"\"
50%
제어문자 \n
```

12-2 printf()- 문자형, 정수형

2018년 10월 21일 일요일 오후 3:24

<printf() - 문자형>

- printf()의 " "의 출력 서식 부분에 사용하는 문자형 서식은 **%c**이며, 예제를 통해 자세히 살펴보자.

<예제>

```
void main()
{
    printf("12345\n");
    printf("%c \n", 'Y');
    printf("%5c \n", 'Y');
    printf("%-5c \n", 'Y');

    getchar();
}
```

1	2	3	4	5	①문자열 12345를 출력한다
Y					②문자 1개 'Y'를 출력한다.
				Y	③5칸 잡아 문자 1개 'Y'를 출력한다.(오른쪽 정렬)
Y					④5칸 잡아 문자 1개 'Y'를 출력하되 왼쪽정렬한다(- 때문에)

문장 printf("%-5c %n",'Y');에서 출력은 기본적으로 오른쪽 정렬이나 -를 해주면 반대로 정렬해서 왼쪽 정렬로 바뀐다.

<printf() -정수형>

- 정수형 서식은 가장 많은 서식을 가지는 자료형으로 대표적인 **%d** 서식으로 출력모양을 알아보자.

<예제>

```
void main()
{
    int x = 13579;
    printf("12345 \n");
    printf("%5d \n", x);
    printf("%3d \n", x);
    printf("%-7d \n", x);
    printf("%7d \n", x);
    printf("%07d \n", x);

    getchar();
}
```

1	2	3	4	5	①		
1	3	5	7	9	②		
1	3	5	7	9	③		
1	3	5	7	9			④
0	0	1	3	5	7	9	⑤

①번은 문자열 "12345"을 5칸 잡아 출력하고, ②번은 변수 x값 13579를 %5d 때문에 5칸 잡아 출력하고, ③번은 %3d 때문에 출력할 값보다 적은 자리수를 적어 놓았지만, 무시하고 출력할 값의 자리수 만큼 자리수를 잡아 출력한다. ④번은 %-7d 때문에 7칸 잡기는 하지만 -로 오른쪽정렬이 왼쪽정렬로 바뀌며, ⑤번은 %07d 때문에 7칸 잡아 오른쪽 정렬 후 남은 공백은 숫자 0으로 채운다.

12-3 printf()- 실수형

2018년 10월 21일 일요일 오후 3:39

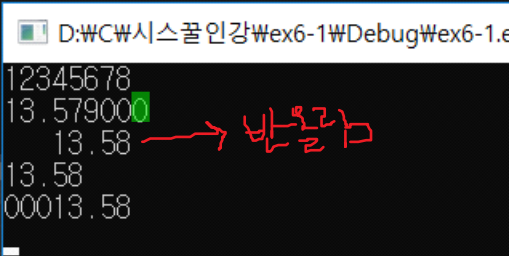
<printf() - 실수형>

- 실수형 서식은 %e, %f, %g 등이 있으나, 가장 많이 쓰이는 %f를 가지고 여러 가지 출력 형태를 살펴보자.
- %f 서식에서 점을 기준으로 왼쪽 숫자는 전체 칸의 개수이고, 점을 기준으로 오른쪽 숫자는 소수점 이하 표시 할 수의 개수이다.

<실행>

```
void main()
{
    float x = 13.579;
    printf("12345678 \n");
    printf("%f \n", x);
    printf("%8.2f \n", x);
    printf("%-8.2f \n", x);
    printf("%08.2f \n", x);

    getchar();
}
```



1	2	3	4	5	6	7	8	①
1	3	.	5	7	9	0	0	②
			1	3	.	5	8	③
1	3	.	5	8				④
0	0	0	1	3	.	5	8	⑤

①번은 8칸 잡아 문자열 "12345678"을 출력하고, ②번은 변수 x값 13.579을 출력하되 %f 서식은 소수점 6자리까지 출력하므로 전부 9칸이 잡힌다. ③번은 %8.2f 때문에 8칸 잡히고 소수점 이하 2자리까지 표시하는데 유효숫자가 소수점 이하 2자리 이상이므로 소수점 이하 3째 자리에서 반올림 한다. ④번은 ③번과 같으나 - 때문에 왼쪽으로 정렬한다. ⑤번은 ③번과 같으나 왼쪽 빈칸을 숫자 0으로 채운다.

13. C언어 입력 서식 함수 scanf()

2018년 10월 21일 일요일 오후 3:50

<scanf()>

- 표준 입력 장치인 키보드로부터 여러 가지 자료형의 데이터를 입력받을 때, 이용하는 함수로, 서식을 가지는 입력함수이며, scanf()의 함수의 원형은 stdio.h에 정의되어 있으므로, #include<stdio.h>를 반드시 프로그램 머리에 써 주어야 한다.

사용형식

scanf("입력서식", &변수리스트)

- scanf() 함수의 사용 형식에서 변수가 여러 개 올 수 있는데, 중요한 것은 기호 &가 온다는 것이다.
 - 기호 &은 ampersand라고 읽으며, 그 쓰임새에 대해서는 10장 포인터 부분에서 자세히 알아보도록 하고, 이번 장에서는 단순히 그 변수의 메모리 시작 주소로만 이해하고 넘어가자.
 - 즉, scanf() 함수의 두 번째 인자 부분에는 주소 개념이 와야 한다고 이해하고 가자. 그리고 이 때문에 scanf() 함수를 사용 할 때는 변수앞에 반드시 기호 &을 붙인다고 생각하면 된다.
- (단, 입력서식이 %s이거나 변수자체가 포인터변수처럼 주소를 의미하면 &을 붙이지 않는다.)

<scanf()>

키보드로 문자열 데이터를 읽어 들일 때의 문장으로 서식 %s을 사용하며 매우 중요한 사실은 변수 앞에 기호 &가 붙지 않는다는 것이다.

<실행>

```
void main()
{
    char name[10];
    char address[30];
    puts("이름을 입력하세요.");
    scanf("%s", name); // gets(name); 가능
    puts("주소를 입력하세요.");
    scanf("%s", address); // gets(address); 가능
    printf("이름은 %s이고, 주소는 %s입니다.\n", name, address);
}
```

Microsoft Visual Studio Debug Console

이름을 입력하세요.
정
주소를 입력하세요.
경기
이름은 정은이고, 주소는 경기도입니다.

예제에서 두 개의 입력 문장을 보면, 변수 개념 앞에 &가 오지 않았다. 함수 scanf()의 ()안의 변수부분에는 엄밀히 말해서 변수의 메모리 시작 주소가 온다고 했다. 그런데 name이나 address는 배열의 이름이므로 배열의 이름은 그 자체가 그 배열이 할당된 메모리 시작 주소를 의미하기 때문에, &을 따로 안 써 줘도 된다. 다르게 표현하면, 출력 서식 부분이 %s이면 변수가 오는 부분에 &을 해주지 않아도 된다.

13-2 scanf()- 문자형, 정수형

2018년 10월 21일 일요일 오후 3:53

Scanf() - 문자형

- 키보드로 문자 1개를 읽어 들일 때의 문장으로 서식 **%c**을 사용하며, getchar() 함수로 다시 표현 할 수 있다. 변수 앞에 기호 &을 붙인다.

--scanf함수 사용시 #pragma warning(disable:4996) 필수 입력

<실행>

```
void main()
{
    char ch;
    puts("문자 1개를 입력하십시오");
    scanf("%c", &ch);
    printf("입력한 문자는 %c입니다. \n", ch);

    getchar();
    getchar();
}
```

D:\WC\시스폴인강\Wex6-1\Debug\Wex6-1.exe

문자 1개를 입력하십시오
f
입력한 문자는 f입니다.

Scanf() -정수형

- 키보드로 정수를 읽어 들일 때의 문장으로 서식 **%d**을 사용하며, 변수 앞에 기호 &을 붙인다.
- %d%d**이므로 두 개의 정수를 읽어 들여 printf()문장을 이용해 두 개의 정수를 출력하고, 두 개의 정수를 더한 값과, 두 개의 정수를 곱한 값, 두 개의 정수의 차이를 출력하는 프로그램이다. 이 때 두 개의 입력 값은 enter키나 공백(spacebar)으로 구분된다.

<실행>

```
void main()
{
    int x, y;
    puts("숫자 2개 입력");
    scanf("%d%d", &x, &y);
    printf("입력한 숫자는 %d와 %d입니다. \n", x, y);
    printf("입력한 숫자 합은 %d입니다. \n", x + y);
    printf("입력한 숫자 곱은 %d입니다. \n", x * y);
    printf("입력한 숫자 차는 %d입니다. \n", x - y);

    getchar();
    getchar();
}
```

D:\WC\시스폴인강\Wex6-1\Debug\Wex6-1.exe

숫자 2개 입력
2 4
입력한 숫자는 2와 4입니다.
입력한 숫자 합은 6입니다.
입력한 숫자 곱은 8입니다.
입력한 숫자 차는 -2입니다.

13-3 scanf()- 실수형

2018년 10월 21일 일요일 오후 4:01

scanf() - 실수형

- 키보드로 실수형 데이터를 읽어 들일 때의 문장으로
서식 **%f**을 사용하며(%e나 %g를 사용 할 수도 있다.
그러나 %f가 주로 사용된다.) 변수 앞에 기호 **&**을
붙인다.

<실행>

```
void main()
{
    float kor, eng, mat, ave;
    puts("숫자 3개를 입력하세요");
    scanf("%f%f%f", &kor, &eng, &mat);
    printf("입력한 점수는 %f, %f, %f입니다. \n", kor, eng, mat);
    printf("입력한 점수는 총점은 %5.1f입니다. \n", kor + eng + mat);
    ave = (kor + eng + mat) / 3.;
    printf("입력한 점수의 평균은 %5.1f입니다. \n", ave);
}
```

Microsoft Visual Studio Debug Console

```
숫자 3개를 입력하세요
99
77
80
입력한 점수는 99.000000, 77.000000, 80.000000입니다.
입력한 점수는 총점은 256.0입니다.
입력한 점수의 평균은 85.3입니다.
```

%f 서식은 소수점 6자리까지 출력

90, 77, 80을 입력 했을 때, 입력 값은 정수처럼 보이지만 입력 받는 서식이 **%f**이므로 실수로 받아들인다. 90, 77, 80을 입력 하고 곧바로 출력 했을 때, 단순한 %f의 출력은 소수점 아래 6 자리까지 출력되므로, 결과의 밑줄 그은 부분처럼 90.000000, 77.000000, 80.000000로 출력된다. 이 때 90.000000이 출력 되는 부분을 **%d**로 출력하게 되면 이미 float형 데이터로 저장 된 것을 정수형 데이터로 출력하는 것이 되므로, 0으로 찍히게 된다. 출력 결과에서 총점과 평균의 출력형식은 %5.1f로 확실하게 표현 해 주었기에 5자리 잡아 소수점 아래 1자리까지 출력된 것이다.

14. 산술연산자

2018년 10월 21일 일요일 오후 4:58

<산술 연산자> Arithmetic Operator

- 산술연산자는 산술연산을 수행하는 연산자이다. 우리가 흔히 말하는 더하기, 빼기, 곱하기, 나누기 등이 이에 속한다.

의미	더하기	빼기	곱하기	나누기	나머지 구함	단항+	단항-
기호	+	-	*	/	%	+	-
예	10+2	10-2	10*2	10/2	10%2	+2	-2

<더하기 연산자> +

더하기 연산자는 + 기호를 사용하며 양쪽에 피연산자가 오는 이항 연산자(binary operator)이다. 예를 들어 `printf("%d\n", 10+20);`은 "10+20"을 출력하라는 것이 아니라, 10+20의 결과인 30을 출력하라는 의미이다. 피연산자로는 값, 수식, 상수 모두 올 수 있다.

<빼기 연산자> -

빼기 연산자는 - 기호를 사용하며 양쪽에 피연산자가 오는 이항 연산자(binary operator)로서 왼쪽의 피연산자의 값에서 오른쪽의 피연산자의 값을 뺀다. 피연산자로는 값, 수식, 상수 모두 올 수 있다. `printf("%d\n", 30-20);`의 출력결과는 10이다.

<곱하기 연산자> *

곱하기 연산자는 * (asterisk)를 사용하며, 역시 양쪽에 피연산자가 오는 이항 연산자(binary operator)이다. 피연산자로는 값, 수식, 상수 모두 올 수 있다. `printf("%d\n", 10*2);`의 결과로 20이 출력된다.

<나누기 연산자> /

나누기 연산자는 / (forward slash)를 사용하며, 역시 양쪽에 피연산자가 오는 이항 연산자(binary operator)이다. 피연산자로는 값, 수식, 상수 모두 올 수 있다. `printf("%d\n", 5/3);`의 출력결과는 1과 1.7이다.

<나머지 연산자> %

나머지 연산자는 %를 사용하며, 역시 양쪽에 피연산자가 오는 이항 연산자(binary operator)로서 왼쪽의 피연산자를 오른쪽의 피연산자로 나눈 나머지를 구하는 연산자이다. 이때 양쪽의 피연산자는 반드시 정수형이어야 한다.

<부호 연산자> -, +

지금까지의 산술연산자들은 모두 양쪽에 피연산자가 오는 이항연산자이나, 부호를 의미하는 +나 -는 연산자가 하나인 단항 연산자이다. 예를 들어 `printf("%d\n", 50-20);`의 결과는 30이지만 `printf("%d\n", 20-50);`의 결과는 -30이며 이때의 -는 부호를 의미하는 단항연산자이다.

<실행>

```
void main()
{
    printf("%.2f\n", 10.0 / 5);
    printf("%.2f\n", 10 / 5.0);
    printf("%.2f\n", 10.0 / 2.0);
    printf("%.2f\n", 10.0 * 2);
    printf("%.2f\n", 10.0 - 5);
    printf("%.2f\n", 10.0 - 5.0);
}
```

Microsoft Visual Studio Debug Console

```
2.00
2.00
5.00
20.00
5.00
5.00
```

15. 관계연산자 1

2018년 10월 21일 일요일 오후 5:26

<관계연산자>

조건을 비교할 때 많이 쓰이는 연산자로서 결과는 참(True : 1), 거짓(False : 0)으로 나타난다. C언어에서는 0이 아닌 모든 것은 참(True : 1)으로 간주 한다.

조건을 참, 거짓 여부에 따라 결과가 참이면 1로 거짓이면 0으로 나타난다. 이때 결과 0값이 정수 값 0, 1과 동일하게 취급되어 예제처럼 정수 변수 c에 결과 값을 저장한 후 출력해도 된다.

연산자	예	의미	결과
==	a == b	a와 b가 같다	같으면 1 다르면 0
!=	a != b	a와 b가 같지 않다.	같지않으면 1 같으면 0
>	a > b	a가 b보다 크다	a가 더크면 1 아니면 0
>=	a >= b	a가 b보다 크거나 같다.	a가 크거나 같으면 1 아니면 0
<	a < b	a가 b보다 작다	a가 b보다 작으면 1 아니면 0
<=	a <= b	a가 b보다 작거나 같다.	a가 b와 같거나 작으면 1 아니면 0

<실행>

```
#include<stdio.h>
void main() {
    int a, b, c ;
    a = 10;
    b = 5;
    c = a > 5;
    printf("%d\n", c);
    c = a < 5;
    printf("%d\n", c);
    c = a == b;
    printf("%d\n", c);
    c = a != b;
    printf("%d\n", c);
    getchar();
}
```

<실행2>

```
#include<stdio.h>
void main(){
    char a,b,c;
    a = 'A';
    b = 'B';
    c = a > b;①
    printf("%d Wn", c);
    c = a < b;②
    printf("%d Wn", c);
    c = a == b;③
    printf("%d Wn", c);
    c = a != b;④
    printf("%d Wn", c);
}
```

char형 변수의 '크다', '작다'를 비교한다는 의미는 그 문자의 **아스키 코드 값을 비교**하는 것이다. 예제에서 'A'와 'B'를 비교했으므로, 'A'는 아스키 코드 값 65를 'B'는 아스키 코드 값 66을 의미하여 비교하게 되는 것으로 ①번은 65>66을 ②번은 65<66을 ③번은 65 == 66을 ④번은 65 != 66을 각각 의미하게 된다.

16. 논리연산자

2018년 10월 21일 일요일 오후 5:48

NOT > AND > OR

관계연산자 > 논리연산자

<논리 연산자> Logical Operator

C언어의 논리연산자 종류는 **&&, ||, !**으로 각각 논리의 **AND, OR, NOT**을 의미하며, 논리의 이 AND, OR, NOT 이 의미하는 뜻을 먼저 알아야 한다, 다음은 이들이 의미하는 표이다. 연산자의 우선순위는 **NOT > AND > OR**의 순이다.

AND연산자 &&의 진리표

x	y	x && y	의미
0	0	0	거짓
0	1	0	거짓
1	0	0	거짓
1	1	1	참

AND연산자 &&은 네모 친 부분 즉, x와 y 모두 참 일 때만, 참이고 나머지는 거짓.

OR연산자 ||의 진리표

x	y	x y	의미
0	0	0	거짓
0	1	1	참
1	0	1	참
1	1	1	참

OR연산자 || 는 네모 친 부분 즉, x와 y 모두 거짓 일 때만 거짓이고 나머지는 참의 결과를 나타낸다. 기호 || 는 |을 2번 누르는 것으로 한번 누른 |의 기호이름은 pipe라고 부른다.

NOT연산자 !의 진리표

x	!x	의미
0	1	참
1	0	거짓

NOT연산자 !는 진리 값을 반대로 만들어 주기만 하면 된다.

<실행>

```
#include<stdio.h>
void main(){
char a, b, c;
a = 'A';
b = 'B';
c = !2 || 3 && !0;①
printf("%d \n", c);
c = a < b && a == b ;②
printf("%d \n", c);
c = a < b || a == b ;③
printf("%d \n", c);
}
```

1
0
1

1. $0 || 3 \&\& 1$

$0 || 1 \rightarrow 1$

2. $65 < 66 \&\& a == b$

$1 \&\& 0 \rightarrow 0$

3. $65 < 66 || 65 == 66$

$1 || 0 \rightarrow 1$

(1) $c = !2 || 3 \&\& !0;$ ①에 대해

!2는 참(1)의 반대이므로 0이 되고, 3은 그 자체가 참이고, !0은 거짓(0)의 반대 이므로, $0 || 1 \&\& 1$ 의 식으로 바뀌 쓸 수 있다. 이때 ||의 순위보다 &&가 먼저이므로 명암있는 부분 부터 먼저 하면, 그 결과가 참이 되어, $0 || 1$ 의 문장으로 바뀌고, 결과는 1이다.

(2) $c = a < b \&\& a == b ;$ ②에 대해

$a < b$ 는 'A' < 'B' 로 $65 < 66$ 의 결과 참인 1이 되고, $a == b$ 는 'A' == 'B' 로 $65 == 66$ 의 결과 거짓 0이 되어 $1 \&\& 0$ 만 남게 된다. 결과는 [표 AND연산자 &&의 진리표]의 보기에 따라 0이 된다.

(3) $c = a < b || a == b ;$ ③에 대해

$a < b$ 는 'A' < 'B' 로 $65 < 66$ 의 결과 참인 1이 되고, $a == b$ 는 'A' == 'B' 로 $65 == 66$ 의 결과 거짓 0이 되어 $1 || 0$ 만 남게 된다. 결과는 [표 OR연산자 ||의 진리표]의 보기에 따라 1이 된다.

17. 음수의 표현

2018년 10월 21일 일요일 오후 6:49

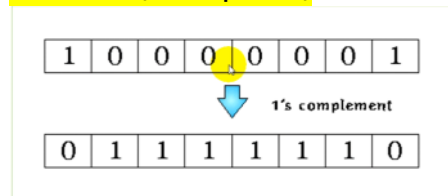
- 컴퓨터에서 음수를 표현하는 방법은 이론적으로 3가지가 있다.
- 1. 부호화 절대치 방법(Signed-magnitude representation)
 - 최상위 비트는 부호표현이고, 나머지는 자리 값을 갖는 크기를 나타낸다. **원**
- 2. 1의 보수 방법(1's complement)
 - 음수를 1의 보수로 표현하는 방법은 비트를 0은 1로 1은 0으로 바꿔준다.
 - 0이 2개(+0과 -0) 존재하는 모순성이 있다.

정수는 16비트(2바이트)로 표현하지만 편의상 하위 1바이트만 표현해보자

수	2의 보수 표현
-1	<div>0 0 0 0 0 0 0 1</div> <div>↓ 1을 0으로 0을 1로</div> <div>1 1 1 1 1 1 1 0</div> <div>↓ 1을 한다</div> <div>1 1 1 1 1 1 1 1</div>
-2	<div>0 0 0 0 0 0 1 0</div> <div>↓ 1을 0으로 0을 1로</div> <div>1 1 1 1 1 1 0 1</div> <div>↓ 1을 한다</div> <div>1 1 1 1 1 1 1 0</div>

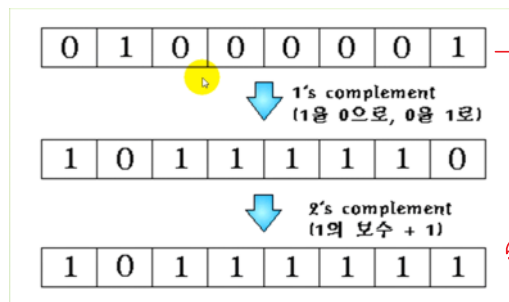
올림수

1의 보수 방법(1's complement)



2의 보수 방법(2's complement)

1의 보수에다가 +1을 해준다. +0, -0 과 같이 0이 2개 존재하는 1의 보수의 모순이 없어졌다. 따라서 컴퓨터의 음수의 표현은 2의 보수를 사용한다.



2의 보수로

18. 비트연산자 - 쉬프트(shift)

2018년 10월 21일 일요일 오후 7:10

<쉬프트 연산자 shift operator>

- 비트를 이동시키는 연산자로 오른쪽으로의 이동과 왼쪽으로의 이동하는 두 가지 종류의 방식이 있다.

연산자	뜻	예	의미
<code>>></code>	오른쪽으로 이동	<code>5 >> 2</code>	5의 이진수표현에서 오른쪽으로 2칸 쉬프트
<code><<</code>	왼쪽으로 이동	<code>5 << 2</code>	5의 이진수표현에서 왼쪽으로 2칸 쉬프트

쉬프트 연산자 `<<`나 `>>`를 만났을 때 제일 먼저 해야 할 일은 왼쪽의 피 연산자를 **이진표현**으로 만드는 것이다. 그런 다음 오른쪽의 피연산자의 수만큼 왼쪽이나 오른쪽으로 **이동(shift)**하면 쉽게 답을 구할 수 있다.

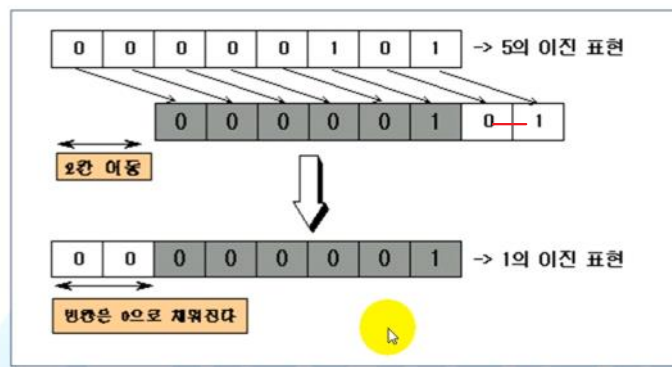
<실행> >>

```
void main()
{
    int a, b, c;
    a = 5;
    b = 2;
    c = a >> b;
    printf("%d >> %d = %d\n", a, b, c);
}
```

Microsoft Visual Studio 디버그 콘솔

5 >> 2 = 1

연산과정

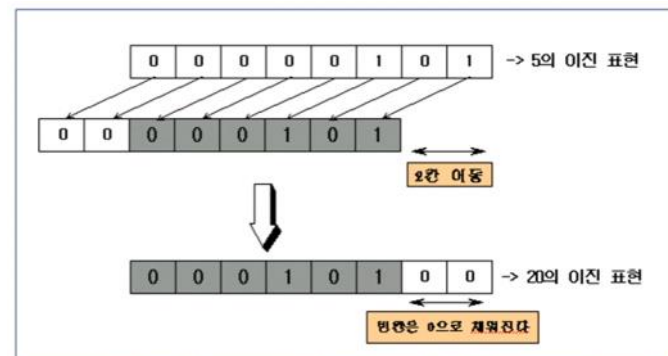


<실행2> <<

```
void main()
{
    int a, b, c;
    a = 5;
    b = 2;
    c = a << b;
    printf("%d << %d = %d\n", a, b, c);
}
```

Microsoft Visual Studio 디버그 콘솔

5 << 2 = 20



18-2 비트연산자 &(AND), |(OR)

2018년 10월 21일 일요일 오후 7:25

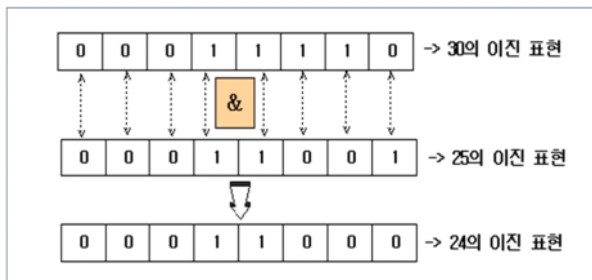
<AND> → 까다로움!!

x	y	x & y	의미
0	0	0	거짓
0	1	0	거짓
1	0	0	거짓
1	1	1	참

<실행>

```
void main()
{
    int a, b, c;
    a = 30;
    b = 25;
    c = a & b;
    printf("%d & %d = %d\n", a, b, c);
}
```

Microsoft Visual Studio 디버그 콘솔
30 & 25 = 24



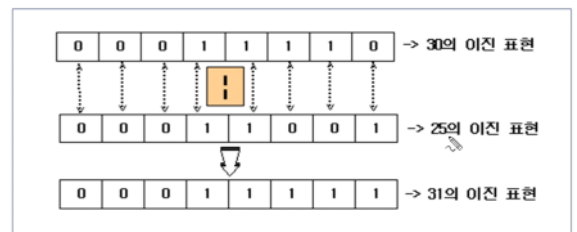
<OR>

x	y	x y	의미
0	0	0	거짓
0	1	1	참
1	0	1	참
1	1	1	참

<실행>

```
void main()
{
    int a, b, c;
    a = 30;
    b = 25;
    c = a | b;
    printf("%d | %d = %d\n", a, b, c);
}
```

Microsoft Visual Studio 디버그 콘솔
30 | 25 = 31



화살표의 각 비트에 대응되는 부분을 논리 OR로 연결해 각각 논리 OR연산을 하면 결과가 00011111과 같이 나온다. 00011111은 십진수로 31이 된다. 30과 25의 논리 OR연산은 31이 된다.

18-3 비트연산자 ^ (XOR), ~(NOT)

2018년 10월 21일 일요일 오후 7:34

일반적인 논리 연산자인 AND, OR, NOT에 대해서는 앞에서 설명하였다. 이번 절에서는 비트의 논리 연산자인 ^ (비트 XOR), ~(비트 NOT)에 대해서 살펴 보도록 하자. AND, OR, NOT에 대해서는 각각의 진리표에 대해 알아보았지만, 연산자 XOR에 대해서는 이번 절에서 처음 설명되는 내용이므로 이것의 진리표를 알아보자. 기호 ^는 캐럿(carrot)이라 부르고, 기호 ~는 틸더(tilder)라 부른다.

<XOR> 청개구리연산~

비트XOR 연산자 ^의 진리표

x	y	x ^ y	의미
0	0	0	거짓
0	1	1	참
1	0	1	참
1	1	0	거짓

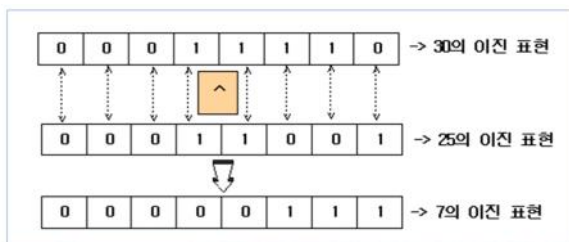
논리연산 XOR은 일명 청개구리 연산으로 네모 친 부분처럼 두 입력 값이 서로 다를 때만 참인 연산이다.

<실행>

```
void main()
{
    int a, b, c;
    a = 30;
    b = 25;
    c = a ^ b;
    printf("%d ^ %d = %d \n", a, b, c);
}
```

Microsoft Visual Studio 디버그 콘솔

30 ^ 25 = 7



논리 XOR은 서로 반대되는 입력 값이 들어왔을 때만 참이고, 같은 입력이 들어오면 거짓 연산하면 00000111이 된다.

<NOT> 비트 뒤집기

<실행>

```
void main()
{
    int a, b, c;
    a = -1;
    b = -4;

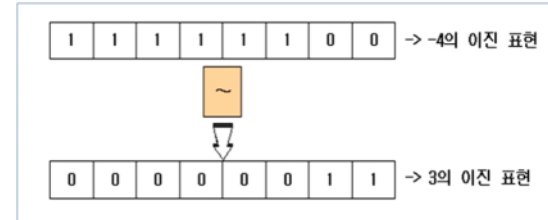
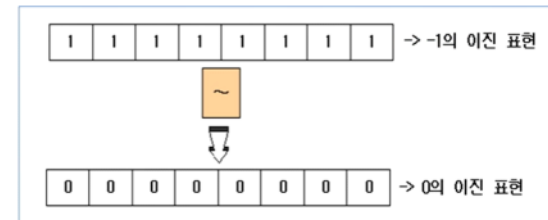
    c = ~a;
    printf("~%d = %d \n", a, c);

    c = ~b;
    printf("~%d = %d \n", b, c);
}
```

Microsoft Visual Studio 디버그 콘솔

~-1 = 0

~-4 = 3



19. 증감연산자

2018년 10월 22일 월요일 오전 11:02

<증감연산자>

어떤 변수에 대해 1증가하고 1감소하는 표현으로 다음의 2종류가 있다.

종류	예	의 미
전위형	<code>++a</code>	<code>a = a+1</code> 또는 <code>a += 1</code>
	<code>--a</code>	<code>a = a-1</code> 또는 <code>a -= 1</code>
후위형	<code>a++</code>	<code>a = a+1</code> 또는 <code>a += 1</code>
	<code>a--</code>	<code>a = a-1</code> 또는 <code>a -= 1</code>