1. Hello, World!

프로그래밍 기초 교육

일단 이 것부터 쳐보자

- 왼쪽 코드를 빌드해서 실행시키면...

  코드를 프로그램으로 만드는 것을 빌드라고 한다.

Visual Studio (Windows)에서

1. New Project에서 'Win32 Console Application'을 선택해 프로젝트를 만든다. (Location을 기억해두자)
2. Source Files 폴더를 오른쪽 클릭하고 Add 〉 New Item (또는 Ctrl+Shift+A)를 눌러 .cpp 파일을 만든다.
3. 코드를 작성한다.
4. Ctrl+F5를 누르면 만든 코드가 빌드되고 실행된다. 빌드가 제대로 되지 않을 경우 F7을 눌러 빌드 시킨다.

```
#include <iostream>
using namespace std;

int main(){

    cout << "Hello, World" << endl;

    return 0;
}
```

```
Hello, World
```

만들어진 .exe 파일은 빌드 모드에 따라 위에서 기억한 Location 폴더의 Debug 또는 Release에 위치해 있다.
(프로젝트를 오른쪽 클릭한 다음 Open Folder in Windows Explorer를 누르면 프로젝트 폴더로 이동할 수 있다.)

일단 이 것부터 쳐보자

- 왼쪽 코드를 빌드해서 실행시키면...

  코드를 프로그램으로 만드는 것을 빌드라고 한다.

Xcode(macOS)에서

1. Create a new Xcode project를 선택하고 macOS 탭에서 'Command Line Tool'을 고르고 [Next]를 누른다.
2. 빈 칸을 채우고 Language는 'C++'로 하고 [Finish]를 누르면 프로젝트를 저장한 위치를 선택한다.
3. 오른쪽 사이드바에서 'main.cpp'를 선택한다.
4. 코드를 작성한다.
5. cmd+R을 누르면 만든 코드가 빌드되고 실행된다. 빌드가 제대로 되지 않을 경우 cmd+B를 눌러 빌드 시킨다.

```
#include <iostream>
using namespace std;

int main(){

    cout << "Hello, World" << endl;

    return 0;
}
```

```
Hello, World
```

Windows와 달리 실행 파일은 어딘가에 꼭꼭 숨겨져 있다.
참고로, Windows와 maxOS의 C++ 실행 파일은 서로의 운영 체제에서 실행시킬 수 없다.

코드 읽기

```
#include <iostream>
using namespace std;

int main(){

    cout << "Hello, World" << endl;
    return 0;
}
```

콘솔에 출력한다.

글자 "Hello, World"

<<를 여러 개 사용해 많은 개수를 출력할 수 있다.

줄 바꿈 (end line)

0으로 끝낸다.
(컴퓨터에게 정상적으로 끝냈다는 걸 알려준다.)

- C++은 기본적으로 main()에서 시작해서 한 줄 씩 한 줄 씩 아래로 진행된다.

손으로 짚어가면서 읽는 것이 좋다.

```
Hello, World
```

코드의 구조

우선 어느 정도의 개념만 알고 가자.

전처리 지시
Preprocessor Directives

전처리 지시는 코드가 컴파일 되기 전에 해야 할 것을 알려준다.

전역 선언
Global Declarations

어떤 자료를 사용할 수 있게 준비하는 것을 '선언, 정의'라고 한다.
전역 선언된 것은 코드 전체에서 사용할 수 있다.

```
int main()
{
    지역 정의   Local Definitions

    각종 구문   Statements
}
```

main 함수

프로그램이 켜지면 int main()의 중괄호 안쪽에 있는 코드부터 실행된다.

```
int func()
{
    지역 정의   Local Definitions

    각종 구문   Statements
}
```

(사용자 정의) 함수

함수는 어떤 동작을 하는 일종의 코드 묶음이다.
함수 만드는 법은 다음 챕터에서 배운다

중괄호 사이의 구역 → 블록 block

블록 안 쪽에서 선언(지역정의)된 것은 그 블록에서만 유효하다.

전처리 지시자

- #으로 시작하는 전처리 지시자는 **컴파일 하기 전에 처리해야 할 일**들을 알려준다.

#include ⟨iostream⟩

- include 구문은 라이브러리(미리 만들어진 기능 묶음)을 첨부한다.

#define KEY VALUE

- 코드 상의 모든 'KEY'를 'VALUE'로 치환한다.

```
#define loop while(true)
int main(){
    loop{
        cout << "Hello" << endl;
    }
    return 0;
}
```

치환되어 컴파일된다

```
int main(){
    while(true){
        cout << "Hello" << endl;
    }
    return 0;
}
```

이름 공간

- 이름 공간(name space)는 이름이 같은 변수/함수를 구분해 **서로 충돌하는 것을 방지**한다.

  ex) '철수'라는 동명이인이 있으면 이름만 가지고는 구분이 안 된다.

  성(이름 공간)를 붙여 '김::철수'와 '안::철수'로 구분한다.

- `using namespace std;`를 적으면 기본적으로 `std`를 사용한다.

  ex) cout과 endl은 각각 콘솔 출력, 줄 바꿈을 나타내며, std라는 이름 공간에 있다.

```
#include <iostream>


int main(){
    std::cout << "Hello"<< std::endl;
}
```

```
#include <iostream>
using namespace std;

int main(){
    cout << "Hello"<< endl;
}
```

`using std::cin;`과 같이 특정 개체(cin)에 대해 지정된 이름 공간(std)를 기본으로 사용하도록 할 수 있다.

주석

- 기능에는 영향을 주지 않고 코드에 코멘트를 달 때 사용한다.
  - 개발자만 볼 수 있기 때문에 주로 코드에 대한 설명을 적는다.

한 줄 주석

```
int main(){
    // 아무 일도 일어나지 않았다...
    return 0;
}
```

여러 줄 주석

```
int main(){
    /*
        이렇게 길어졌지만
        아무 일도 일어나지 않았다...
    */
    return 0;
}
```

주석 안에 또 주석을 달면 안 된다!

bad ex)

```
/*
    ==========
    /*
       이렇게 쓰면 안 된다!
    */
    ==========
*/
```

변수

- 특정한 종류(자료형, type)의 자료(값, value)을 담고 있는
  이름이 붙여진(named) 메모리 공간
  … 넣을 수 있는 종류가 한정된 자료(data)를 담을 수 있는 **상자**라고 생각하자

선언 declaration

- 변수를 만든다. 만들어야 사용할 수 있다.

```
int val; // 자료형 식별자(identifier)
```

int는 정수, float는 실수를 나타낸다.

```
int n;
float a, b;
```

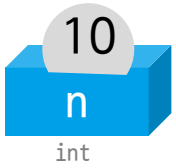
,(쉼표)로 구분해서 여러 개의 변수를 한 번에 만들 수 있다.

변수

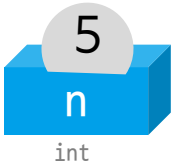
초기화 initialization

- 변수에 값을 넣는다(대입한다).

```
int n; // 먼저 만들어야 한다.
n = 10; // n에 10을 넣는다.
```




```
int n = 10;
//선언과 초기화는 한 번에 할 수 있다.
n = 5; // 값은 바꿀 수 있다.
cout << n; // 5 출력
```



변수를 그대로 출력하면
그 변수의 값이 출력된다.

상수 constant

```
const int n = 10;
```

변수를 선언할 때 앞에 const를 붙여주면
앞으로 값을 바꿀 수 없다.

식별자

- 각 변수나 이런 것들을 **부르는(식별하는) 고유한(unique) 이름**이다.
- A-Z, a-z, 0-9, _로 구성되며, 숫자로 시작할 수 없다.
- 대소문자를 구분한다. (KHLUG와 khlug는 다르다!)
- 예약어(문법에 사용되는 용어 등)은 식별자로 사용할 수 없다.

  ex) if, case, while, for, …

```
int n = 10;
n = 5;
```

좋은 식별자는 변수의 이름을 충분히 설명하면서 길지 않은 것!

myWeight
carNumber
studentID

자료형

- 자료의 종류에는 기본적으로 5 종류가 있다.
- 어떤 자료형을 가진 변수에 다른 종류의 자료형을 가진 데이터를 억지로 집어넣으려 하면 오류가 발생하거나 데이터가 손실된다.
- 자료형에 따라 변수의 크기(상자가 얼마나 큰가)가 달라지는데, 이에 따라 가능한 값의 범위가 달라진다.

부정형

- void 공허... 아무 것도 들어갈 수 없다.

자료형

정수형

- short int 2 bytes $-32768(-2^{15}) \sim 32767(2^{15}-1)$
- int 4 bytes $-2^{31} \sim 2^{31}-1$
- long int 4 bytes $-2^{31} \sim 2^{31}-1$
- long long int 8 bytes $-2^{63} \sim 2^{63}-1$

메모리에서

맨 앞자리를 부호로 사용한다.

`short a=6;` short int a=6;과 같음

0 (부호)	0	0	0	0	1	1	0

`short a=-6;` short int a=-6;과 같음

1 (부호)	1	1	1	0	0	1	0

음수에서 부호 이하는 2의 보수(0을 1로, 1을 0으로 바꾼 다음 1을 더하는 것)로 표현된다.

부호 안 붙이기

`unsigned`를 붙이면 맨 앞자리를 부호로 사용하지 않기 때문에 범위가 달라진다.

ex) `unsigned short int`는 0 ~ 2^8의 범위를 가짐

자료형

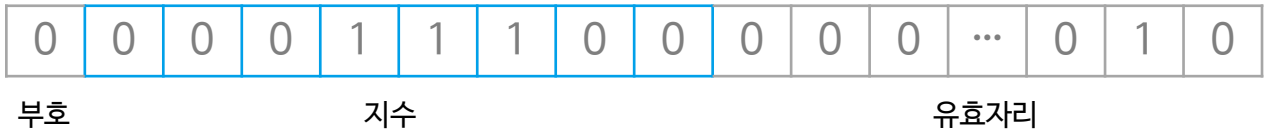
실수형

- float 4 bytes 소수점 표시 가능
- double 8 bytes 더 정밀하게 표현 가능
- long double 8 bytes 더 정밀하게 표현 가능

메모리에서

```
float f=4.7331654e-30;
```

$(-1)^{부호} \times (1+유효자리) \times 2^{(지수-127)}$



논리형

- bool 1 byte true(=1, 또는 0이 아닌 수) / false(=0)

자료형

문자형

- char 1 bytes 작은따옴표 안에 문자 하나를 적을 수 있다.

메모리에서

```
char c='A';
```

0	1	0	0	0	0	0	1

정수 65와 똑같이 저장된다.. 그렇다면?

형 변환 type casting

- 일부 자료형은 서로 변환이 가능하다.

```
char c = 'A';
cout << (int)c;
// char를 int로 변환해서 "65" 출력
int i = 97;
cout << (int)i;
// int를 char로 변환해서 "a" 출력
float f = 2.5;
cout << (int)f;
// float를 int로 변환해서 "2" 출력
(소수점 이하 버림)
```

실수에서 정수로 변환할 때 소수점 이하가 버려지듯이
일부 형 변환 시에는 데이터가 유실될 수 있다.

아스키 코드

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

오버플로우

- 정수형 변수에서 범위를 초과하면 어떻게 될까?

메모리에서

int x = 2147483647

0	1	1	1	…	1	1	1
부호							

int x = 2147483647 + 1

1	0	0	0	…	0	0	0
부호							

signed이고 맨 앞이 1이므로 음수이며,
이 수는 -2147483648를 나타낸다.
이럴 땐 자료형으로 long을사용해야 한다.

- 표현할 수 있는 범위를 넘어섰기 때문에 넘친다는 뜻으로 '오버플로우'라고 한다.

```
#include <iostream>
using namespace std;

int main(){
    int x = 2147483647;
    cout << x << endl;

    int y = x + 1;
    cout << y << endl;

    return 0;
}
```

```
2147483647
-2147483648
```

콘솔 입력

- 왼쪽 코드를 빌드해서 실행시키면
  글자와 숫자를 입력 받고,
  이를 다시 출력해준다.

영어는 1글자에 1바이트,
한글은 1글자에 2바이트

```
#include <iostream>
using namespace std;

int main(){

    char input1[20];
    int input2;

    cout << "글자를 입력하세요: ";
    cin >> input1;

    cout << "숫자를 입력하세요: ";
    cin >> input2;

    cout << endl << "입력한 글자: " << input1;
    cout << endl << "입력한 숫자: " << input2;

    cout << endl << endl;

    return 0;
}
```

char로 만든 변수에 [20]을 하면
20바이트까지 입력할 수 있다.
char* input;으로 하면 제한이 없다.

문자 하나는 '(작은따옴표)로 감싸지만
문자 여러 개 (문자열)은 "(큰 따옴표)로
감싼다.

cin

- 콘솔 입력, >>를 통해
  입력 값을 넘겨줄 변수를 지정한다.

콘솔 출력 +

setw(int)

- 바로 뒤의 출력물에 대해 칸을 만들어준다. 기본적으로 오른쪽 정렬이다.
- left 또는 right를 통해 앞으로의 정렬을 정할 수 있다.

setfill(char)

- 만들어진 칸이 비어있을 때 채울 문자를 정한다.

setprecision(int)

- 실수를 출력할 때 소수점 이하 몇 째 자리까지 출력할지 정한다.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main(){
    cout << setw(10) << "Hello"<< endl;
    cout << setw(10) << left
        << "Hello" << "\n";
    cout << setfill('0') << right;
    cout << setw(10) << 12345
        << "\n";
    cout << setprecision(10);
    cout << (1.0/7.0) << "\n\n";
    return 0;
}
```

〈iomanip〉 라이브러리에 있다.

```
     Hello
Hello
0000012345
0.1428571429
```

setw(10)
setw(10), left
setfill('0'), right, setw(10)
setprecision(10)

산술 연산자

- `a = b` 대입 — a의 값이 b의 값이 된다. **b의 값은 변하지 않는다.**

사칙연산 같은 자료형 끼리 연산을 하면 결과값도 그 자료형이다.

- `a + b` 덧셈 — a와 b를 더한 값이 된다.
- `a - b` 뺄셈 — a와 b를 뺀 값이 된다.
- `a / b` 나눗셈 — a와 b를 나눈 값이 된다.
- `a * b` 곱셈 — a와 b를 곱한 값이 된다.

정수를 정수로 나누면 정수가 되기 때문에 소수점이 날아갈 수 있다.

나머지 연산

나머지 연산

a를 b로 나눈 나머지가 된다.
정수만 된다.

```
int a = 3;
int b = 4;
cout << a + b ; // 7이 출력된다.
```

산술 연산자

할당연산

- a += b 덧셈 a = a+b 와 같은 결과가 된다.
- a -= b 뺄셈 a = a-b 와 같은 결과가 된다.
- a /= b 나눗셈 a = a/b 와 같은 결과가 된다.
- a *= b 곱셈 a = a*b 와 같은 결과가 된다.
- a %= b 나머지 연산 a = a%b 와 같은 결과가 된다.

```
int a = 3;
int b = 4;

a += b;

cout << a << endl;
// a(3)에 b(4)를 더해서 7이 출력된다.
```

산술 연산자

증감연산

- ++a 전위연산 a에 1을 더하거나 빼고 변수 값을 준다.
  --a

```
int a = 0;
cout << ++a << endl;
// a에 1을 더하고 주었기에 1 출력
cout << a << endl; // 1 출력
```

- a++ 후위연산 변수 값을 주고 a에 1을 더하거나 뺀다.
  a--

```
int a = 0;
cout << a++ << endl; // 0 출력
cout << a << endl;
// a를 쓰고 1을 더했기에 1 출력
```

불 연산

- 참(1), 거짓(0) 두 가지 원소만 존재하는 집합에서의 연산이다.
- 컴퓨터는 이진법을 사용하므로 불 연산을 토대로 모든 연산이 이루어진다.

논리부정 (NOT) 참 거짓 뒤집기

x	~x
0	1
1	0

논리곱 (AND) 모두 참이면 참

x	y	x · y
0	0	0
0	1	0
1	0	0
1	1	1

논리합 (OR) 하나라도 참이면 참

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

배타적 논리합 (XOR) 서로 다르면 참

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

비트 연산자

- 이진수 각 자리에 대해 논리연산을 한다.

비트연산

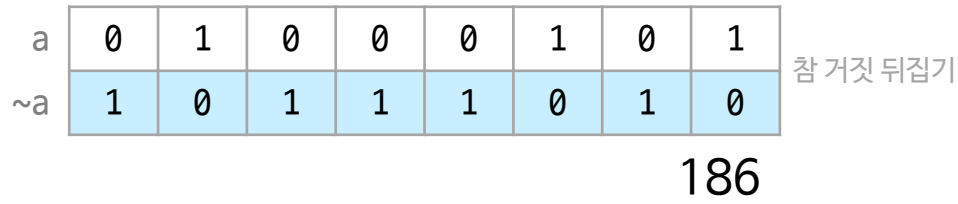
• ~a	비트 NOT	참/거짓 뒤집기
• a & b	비트 AND	(비트 별로) 둘 다 이 1이면 1
• a ¦ b	비트 OR	(비트 별로) 둘 중 하나가 1이면 1
• a ^ b	비트 XOR	(비트 별로) 서로 다르면 1
• a << b	비트 왼쪽 시프트	각 비트를 b번 왼쪽으로 민다.
• a >> b	비트 오른쪽 시프트	각 비트를 b번 오른쪽으로 민다.

단항 연산자인 ~을 제외하고 할당 연산이 가능하다.

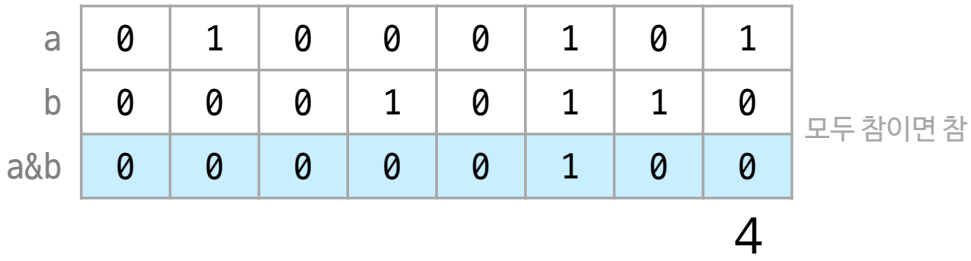
비트 연산자

```
unsigned char a = 69;    // 0b01000101, 1byte짜리 정수이다.
unsigned char b = 22;    // 0b00010110
```

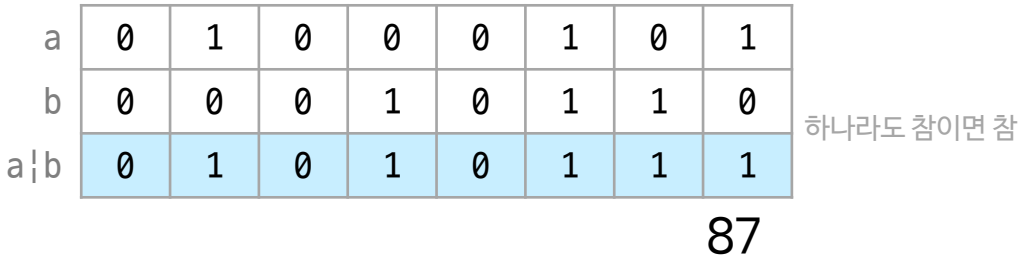
~a



a & b



a ¦ b



비트 연산자

```
unsigned char a = 69;    // 0b01000101
unsigned char b = 22;    // 0b00010110
```

a	0	1	0	0	0	1	0	1	
b	0	0	0	1	0	1	1	0	서로 다르면 참
a^b	0	1	0	1	0	0	1	1	

83

a	0	1	0	0	0	1	0	1	
a<<1	1	0	0	0	1	0	1	0	왼쪽으로 한 칸 이동, 오른쪽 끝은 0으로 채움

138

a	0	1	0	0	0	1	0	1	
a>>1	0	0	1	0	0	0	1	0	오른쪽으로 한 칸 이동, 왼쪽 끝은 부호 자리와 같은 것으로 채움

34

연산 우선 순위

- 우선순위가 높을수록 먼저 계산된다.
- 연산 방향은 피연산자를 어디부터 읽을 것인가를 말한다.

연산자 종류		연산자	연산 방향	우선순위
Primary		( ) [ ] -> .	→	▲ 높음
단항		! ~ ++ -- - (캐스팅) * &	←	
이항	승 제	* / %	→	
	가 감	+ -	→	
	비트	<< >>	→	
	대소	< <= => >	→	
	등가	== !=	→	
	비트 AND	&	→	
	비트 XOR	^	→	
	비트 OR	|	→	
	논리 AND	&&	→	
	논리 OR	||	←	
삼항	조건	? :	←	낮음 ▼
대입		= += -= *= /= %=	←	
나열		,	→	

a = (3 + 4) - 5 * 6

괄호부터 계산

- 보다 * 먼저 계산

- 계산

연산 방향이 오른쪽부터이므로 오른쪽부터 계산

연산 순위: ( ) → + → * → - → =

2017.03.21. 프로그래밍 기초 (2017-1)
with D.com