



C++ 프로그래밍

김 형 기

hk.kim@jbnu.ac.kr

C++ 프로그램의 구조

(Summary) C++ Program Structure

- 빌드 프로세스
- 오류의 종류
- 기본 구조, 기능과 용어
 - 키워드 / 지시자 / 연산자
 - 전처리 지시문
 - main() 함수
 - 주석
 - Namespace
 - 표준 입출력

(Summary) C++ Program Structure

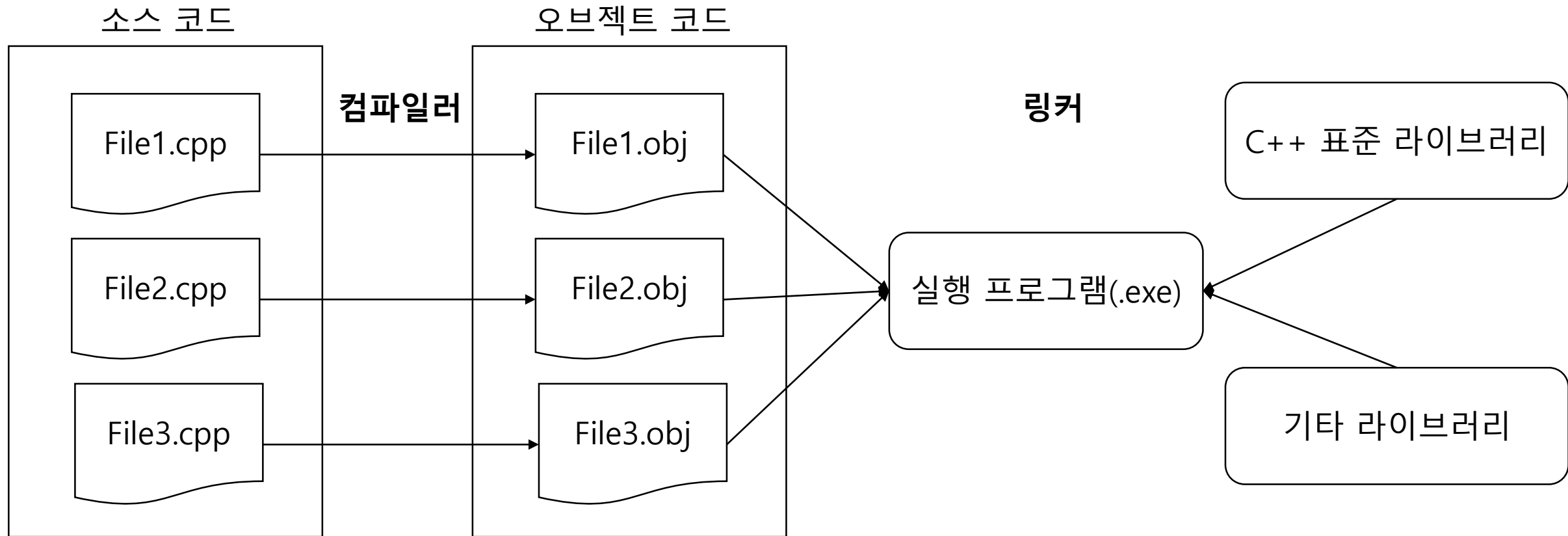
- 빌드 프로세스
- 오류의 종류
- 기본 구조, 기능과 용어
 - 키워드 / 지시자 / 연산자
 - 전처리 지시문
 - main() 함수
 - 주석
 - Namespace
 - 표준 입출력

Terminologies

- 프로그래밍 언어: 고수준의 소스 코드 작성에 사용, Human-readable
- 오브젝트 코드: Machine-readable, 컴퓨터가 실행할 수 있는 코드
- 컴파일러: 소스코드를 오브젝트 코드로 변환하는 도구
- 링커: 오브젝트 코드를 실행 파일(exe)로 변환하는 도구
- 테스트 & 디버깅: 프로그램에 존재하는 오류를 찾고, 수정하는 과정

- IDE (Integrated Development Environment)
 - 텍스트 에디터 + 컴파일러 + 링커 + (디버거)
 - 텍스트 에디터: .cpp 소스 코드 / .h 헤더 파일의 편집기

C++ Build Process



Writing code using VS 2022

1. 새 프로젝트 만들기 → 빈 프로젝트

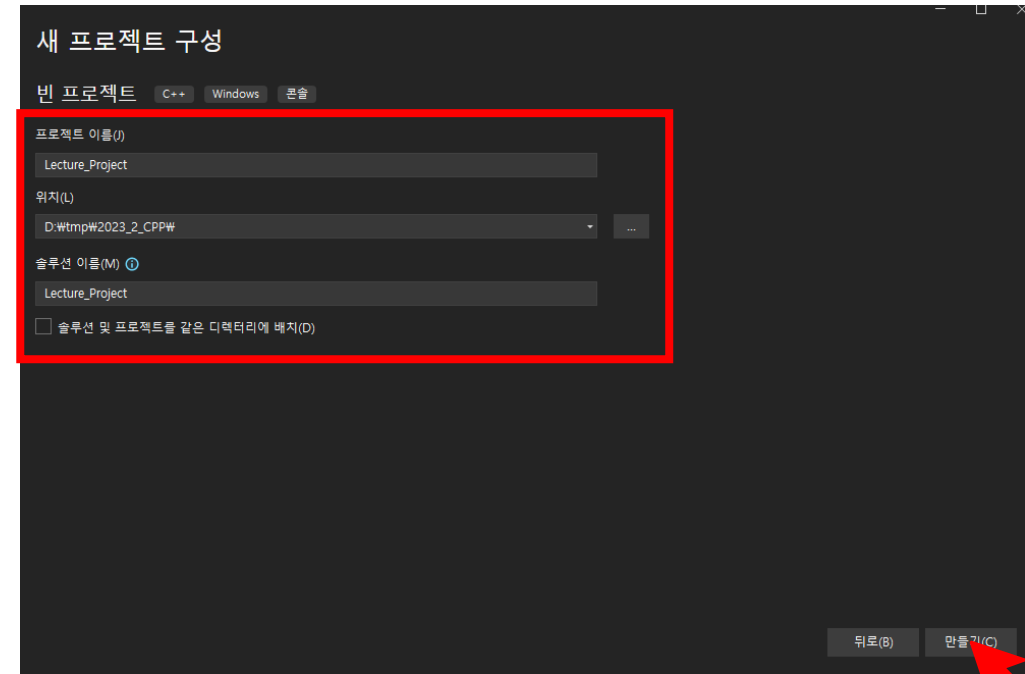
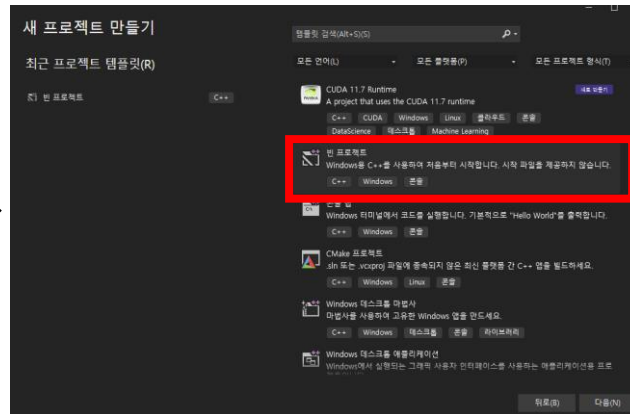
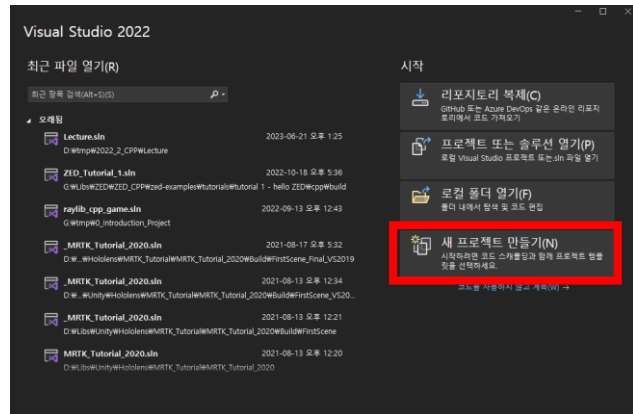
🔗 : 여러분이 스스로 의문을 가지면 좋겠을 사항에 대해 붙입니다.
(ex, 체크하고 만들면 어떻게 될까? 검색 또는 직접 해보기)

2. 프로젝트/솔루션 생성할 위치와 프로젝트 이름을 작성하고 확인 클릭

🔗 (솔루션 및 프로젝트를 같은 디렉터리에 배치하는 체크 해제 상태로 두기 권장)

● 주의사항

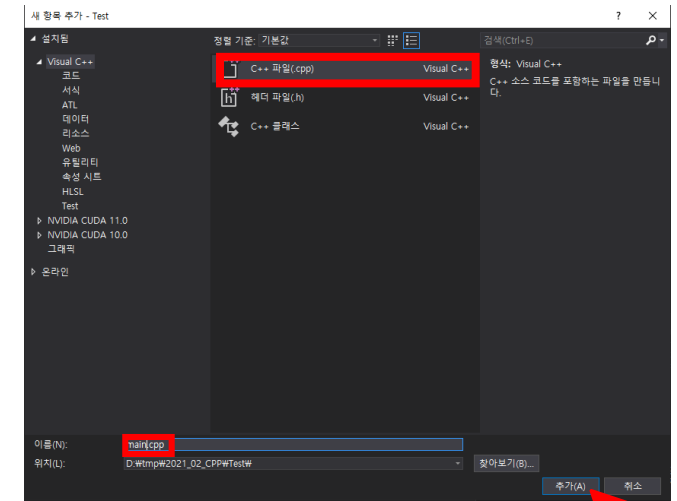
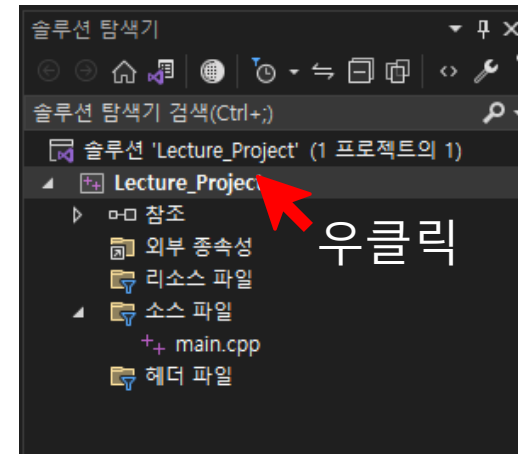
- (꼭!) 프로젝트 생성 위치는 디폴트로 선택되어 있는 곳 말고, 본인이 코드를 관리할 수 있는 곳으로 선택하세요.
- (권고) 프로젝트 이름, 파일 이름 등에 한글 사용하지 마세요.



2번의 새 프로젝트 창

Writing code using VS 2022, cont'd

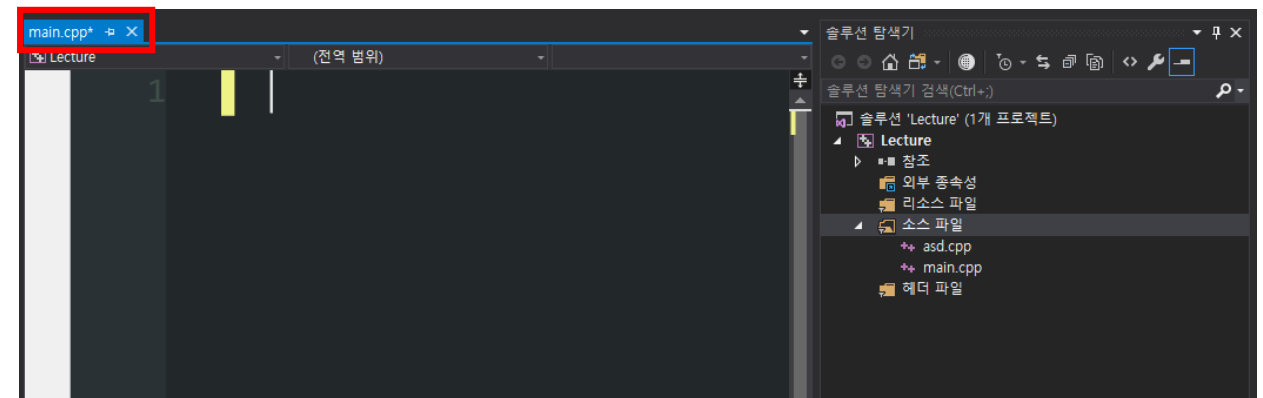
- 오른쪽 솔루션 탐색기의 프로젝트 이름 부분을 우클릭하여 추가 → 새 항목 선택
- 새 항목 추가 창에서 C++파일(.cpp) 선택
- 아래쪽 파일 이름을 main.cpp로 바꾸고 추가 클릭
- 왼쪽 편집기에 main.cpp가 열려있는지 확인 후 코드 작성



5번의 새 항목 추가 창

● 주의사항

- 파일 이름에 한글 사용하지 마세요.



7번의 편집기에 main.cpp가 열려있는지 확인 방법

- 입출력

```
1 #include <iostream>
2
3 int main()
4 {
5     int favoriteNumber;
6
7     std::cout << "Enter the number: ";
8     std::cin >> favoriteNumber;
9     std::cout << "You entered " << favoriteNumber << std::endl;
10
11     return 0; //optional
12 }
```

*따라서 직접 타이핑 하세요.

*첫 단계부터 따라하지 않으면 평생 안 합니다...

First program

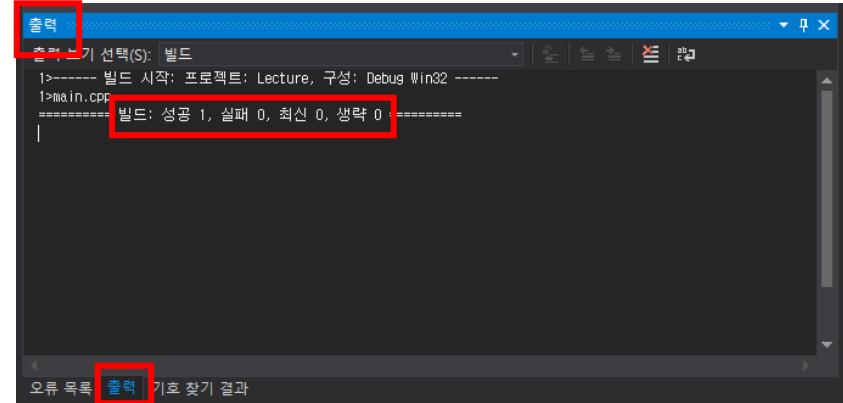
- 입출력 in C style

```
1 #include <stdio.h>
2
3 void main()
4 {
5     int favoriteNumber;
6     printf("Enter the number: ");
7     scanf("%d", &favoriteNumber);
8     printf("You entered %d", favoriteNumber);
9 }
```

Build code using VS 2017

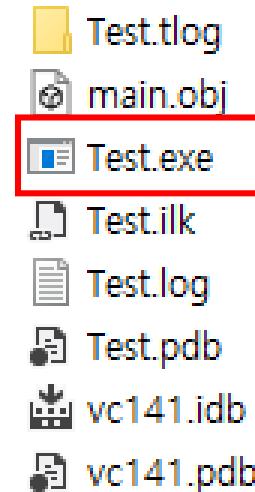
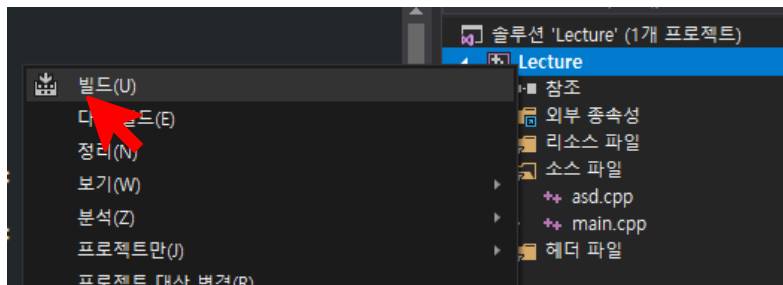
- (컴파일) Ctrl+F7

- 각 소스(cpp)에 대해 obj 파일을 생성하는 과정
- 아래쪽 출력 창에서 성공했는지 확인
 - 오류가 있다면 실패하고, 오류 메시지를 보여줌



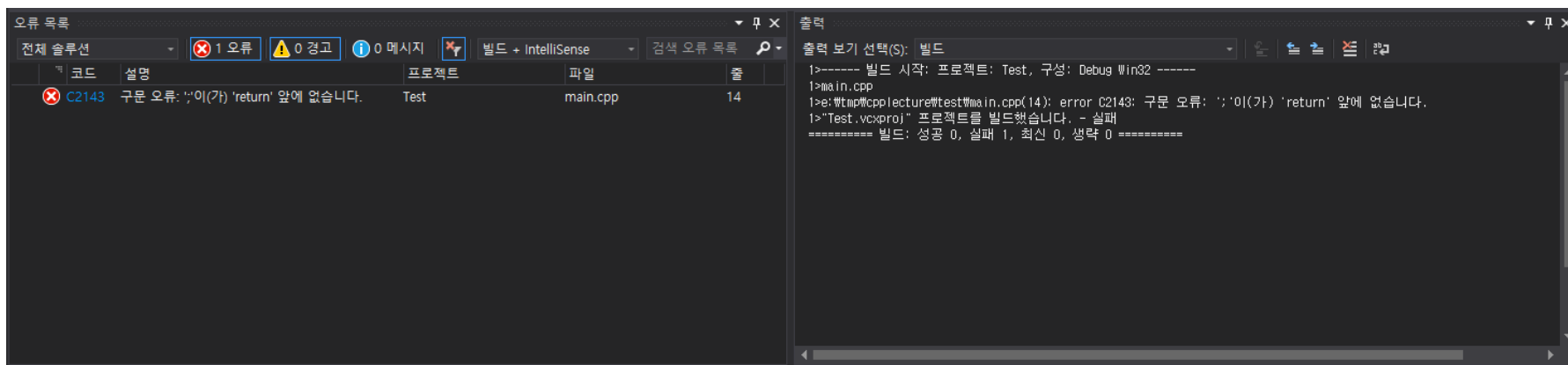
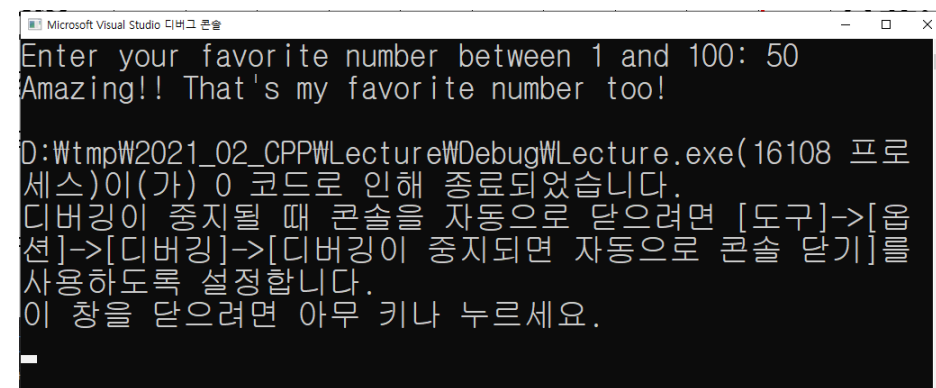
- (빌드) 프로젝트 우클릭 후 빌드 클릭 : Build (Compile + Linking)

- Obj 파일들을 링크하여 실행 가능한 exe 생성



● (실행, 디버깅) F5

- 코드를 빌드(=컴파일+링크)하고 exe를 실행
 - 즉, F5만 누르면 컴파일+링크 과정이 통합적으로 실행됨
- 출력을 보여주는 콘솔창이 나타남
 - 프로그램이 종료된 후, 아무 키나 누르면 콘솔 창이 닫힘
 - 창을 닫지않고 다시 빌드하려고 하면 문제 발생하는 경우 있으므로 주의!
- Ctrl+F5 기능은 정확히 뭘 하는건지 알기 전에는 그냥 사용하지 마세요!



(Summary) C++ Program Structure

- 빌드 프로세스
- 오류의 종류
- 기본 구조, 기능과 용어
 - 키워드 / 지시자 / 연산자
 - 전처리 지시문
 - main() 함수
 - 주석
 - Namespace
 - 표준 입출력

Compiler Errors

- 프로그래밍 언어의 규칙을 위반하는 경우
- 문법적 오류 – 코드 자체 오류

```
1 std::cout << "Errors << std::endl;  
2 // or  
3 return 0
```

- 의미 오류

```
1 int a = 5;  
2 string b = "Hello World";  
3  
4 a + b;
```

Compiler Warnings

- 코드에 잠재적인 문제가 있을 것으로 예상될 때
- 빌드는 가능하지만, 무시하면 안됨!

```
1 int distanceDriven;  
2 std::cout << distanceDriven;
```

**위 코드의 경우 VS 2022(MSVC 컴파일러 기본 설정)에서는 오류(Error)로 처리
컴파일러마다 동작이 약간씩 다를 수 있음

Linker Errors / Runtime Errors

- 링크 에러

- obj 파일의 링크 과정에서 오류가 있을 경우
- 주로 라이브러리 또는 obj 파일을 (어떤 이유에서) 찾을 수 없는 경우

- 런타임 에러

- 프로그램의 실행 도중 발생하는 오류
- Divided by zero, file not found, out of memory, etc...
- 프로그램의 crash
- 예외 처리를 통해 문제 발생에 따르는 처리를 할 수 있음!

Logical Errors

- 프로그램의 동작에 관한 논리적 오류
- 프로그램 작성자의 실수가 원인
- 테스트 과정을 통해 찾아내고, 수정해야 함!

```
1 if (age ≥ 19)
2 {
3     std::cout << "You can drink!";
4 }
```

(Summary) C++ Program Structure

- 빌드 프로세스
- 오류의 종류
- 기본 구조, 기능과 용어
 - 키워드 / 지시자 / 연산자
 - 전처리 지시문
 - main() 함수
 - 주석
 - Namespace
 - 표준 입출력

C++ Programming Structure

- 구성요소 – keyword, identifier, operator
- 전처리기 지시문 - #include,...
- main() 함수
- 네임스페이스
- 주석 - //, /* */
- 입출력 – cin/cout

C++ Keyword, Identifier, Operator

● 키워드

- 약 90개의 키워드 (변수 타입, if, for 등)
- 언어 자체에서 예약된 단어들

● 식별자

- 변수, 함수, 타입 등 개발자가 지정하는 부분
- 대소문자 구분!

● 연산자

- +, -, *, /, >>, <<, ::, ...

● 이러한 요소들이 모여 "문법"을 이룸

cppreference.com

Create account

Search

Page

Discussion

View

Edit

History

C++

C++ language

C++ keywords

C++ keywords

This is a list of reserved keywords in C++. Since they are used by the language, these keywords are not available for re-definition or overloading.

alignas (since C++11)	double	register(2)
alignof (since C++11)	dynamic_cast	reinterpret_cast
and	else	requires (since C++20)
and_eq	enum	return
asm	explicit	short
atomic_cancel (TMTS)	export(1)	signed
atomic_commit (TMTS)	extern(1)	sizeof(1)
atomic_noexcept (TMTS)	false	static
auto(1)	float	static_assert (since C++11)
bitand	for	static_cast
bitor	friend	struct(1)
bool	goto	switch
break	if	synchronized (TMTS)
case	import (modules TS)	template
catch	inline(1)	this
char	int	thread_local (since C++11)
char16_t (since C++11)	long	throw
char32_t (since C++11)	module (modules TS)	true
class(1)	mutable(1)	try
compl	namespace	typedef
concept (since C++20)	new	typeid
const	noexcept (since C++11)	typename
constexpr (since C++11)	not	union
const_cast	not_eq	unsigned
continue	nullptr (since C++11)	using(1)
co_await (coroutines TS)	operator	virtual
co_return (coroutines TS)	or	void
co_yield (coroutines TS)	or_eq	volatile
decltype (since C++11)	private	wchar_t
default(1)	protected	while
delete(1)	public	xor
do	constexpr (reflection TS)	xor_eq

C++ preprocessor

- 전처리기

- 컴파일 이전에 처리됨
- "#" 으로 시작

```
1 // include
2 #include <iostream>
3 #include "myFile.h"
4
5 // define
6 #ifdef
7 #ifndef
8 #define
9 #undef
10
11 #pragma
```

● #include ?

- 단순한 복사 붙여넣기!
- 프로젝트 속성 → C/C++ → 전처리기 → 파일로 전처리를 통해 확인

```
1 #include <iostream>
2
3 int main()
4 {
5     int a = 0;
6
7     return 0;
8 }
```

이 위쪽으로 약
54000줄의 코드는
iostream에서 복사
붙여넣기 한 것

```
54839 .#pragma warning(pop)
54840 .#pragma pack(pop)
54841 #line 53 "c:\\program.files.(x86)\\microsoft.visu
54842 #line 54 "c:\\program.files.(x86)\\microsoft.visu
54843
54844
54845
54846
54847
54848 #line 2 "e:\\tmp\\cpplecture\\test\\main.cpp"
54849
54850 int main()
54851 {
54852     → int a = 0;
54853
54854     → return 0;
54855 }
54856
```

● #define ?

- 단순한 복사 붙여넣기!
- Platform independency 구현, 코드 단축, debug 목적 등으로 다양하게 활용

```
1 #include <iostream>
2
3 #define DEBUG
4
5 #ifdef DEBUG
6 #define LOG(x) std::cout << x << std::endl;
7 #else
8 #define LOG(x)
9 #endif
10
11 int main()
12 {
13     LOG("Hello");
14 }
```

**3번 라인의 #define ... 이
있는 경우와 없는 경우
동작을 비교*

C++ Comments

● 주석

- 프로그래머가 읽을 수 있는 정보를 제공하기 위함 (협업, 유지보수 목적)
- 전처리 단계에서 무시(삭제)되기 때문에 프로그램의 동작과는 무관
- `///
/* */`

```
1 #include <iostream>
2
3 int main()
4 {
5     int favoriteNumber; // 좋아하는 숫자
6
7     /*std::cout << "Enter number: ";
8     std::cin >> favoriteNumber;
9     std::cout << "Entered " << favoriteNumber << std::endl;*/
10 }
```


- 모든 C++ 프로그램은 **하나**의 main()함수를 **가져야 함**
 - main() 함수는 프로그램의 진입점(=프로그램이 실행되면 가장 먼저 실행되는 함수)
 - 리턴값 0이 올바른 프로그램 실행을 의미함(지금은 생략해도 무방)
- 두 가지 버전
 - 이번 학기 강의에서는 간단한 왼쪽 형태만 사용할 것

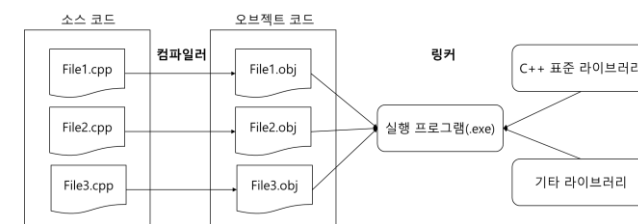
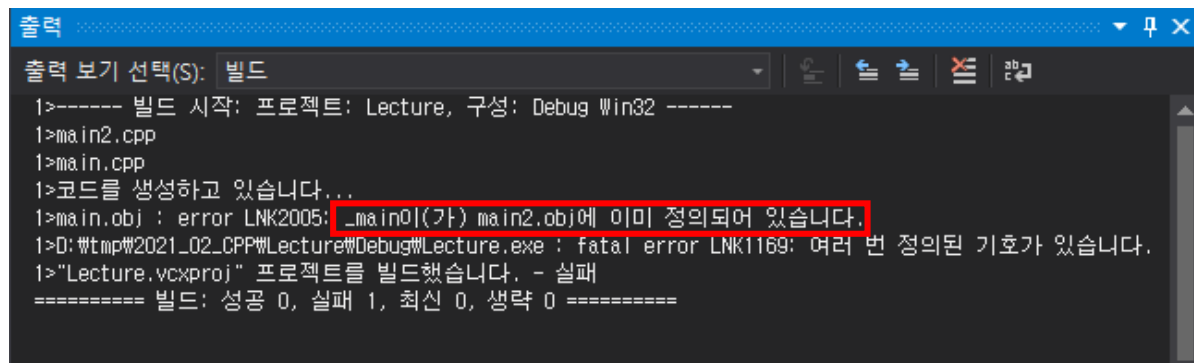
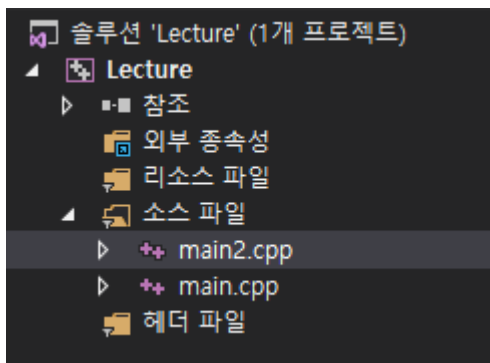
```
1 int main()  
2 {  
3     // code  
4 }
```

>> program.exe

```
1 int main(int argc, int *argv[])  
2 {  
3     // code  
4 }
```

>> program.exe argument1 argument2

- 모든 C++ 프로그램은 하나의 main()함수를 가져야 함
 - 동일 프로젝트에 cpp파일을 계속 추가하여 작성하면 오류 발생 → 하나의 프로그램에 2개 이상의 main함수가 존재하기 때문
 - 해결법 1) 솔루션을 우클릭하여 추가→새 프로젝트로 별도의 프로젝트를 만들어 사용
 - 이때, 현재 빌드하고자 하는 프로젝트를 우클릭하여 "시작 프로젝트로 설정" 필요
 - 해결법 2) 현재 빌드하고자 하는 cpp이외의 코드는 전체 주석처리하고 빌드
 - 해결법 3) ?



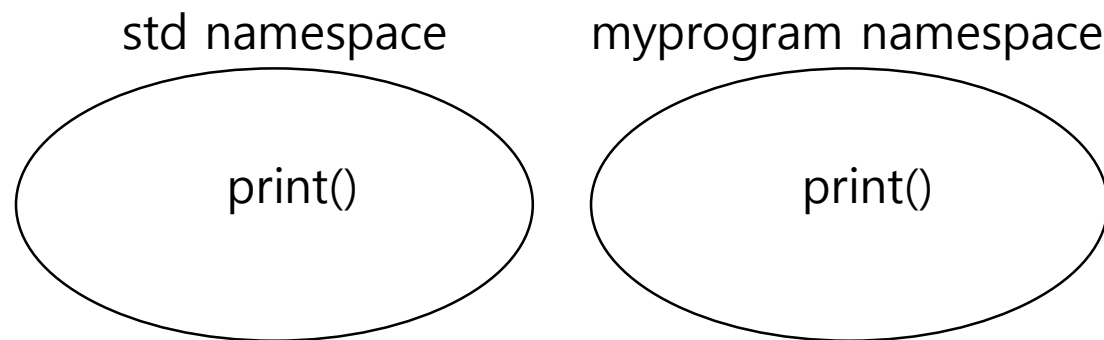
- 프로젝트 안의 "모든" cpp 파일이 하나의 "프로그램"을 구성
- 즉 cpp 파일 여러 개에 main이 존재하면 하나의 "프로그램"에 여러 main이 존재하여 오류가 발생!

각 cpp파일에 모두 main함수가 존재하면,

이와 같은 오류가 발생하고 빌드 실패

C++ namespace

- `std::cout`
- 충돌 방지를 위함
 - 외부 라이브러리와 구현한 소스 코드간의 이름 충돌 가능성
 - 또는 코드가 커지면, 내가 만든 함수들 사이에서도 실수로 충돌 가능
- 코드의 “그룹화” 로 이해
 - 서로 다른 namespace로 그룹화하여 충돌을 방지할 수 있음
- “`::`” : scope resolution operator



- using namespace

- 특정 namespace 내의 함수들을 사용하겠다는 선언
- 남용은 금물!
 - Using namespace를 모든 코드에 넣는다면 namespace의 기능을 상실함...
 - 강의 자료에서는 지면의 한계로 인해, 단축 표현을 위해 사용

```
1 #include <iostream>
2
3 int main()
4 {
5     int favoriteNumber;
6
7     std::cout << "Enter ...";
8     std::cin >> favoriteNumber;
9     std::cout << "Amazing!! ..." << std::endl;
10 }
```

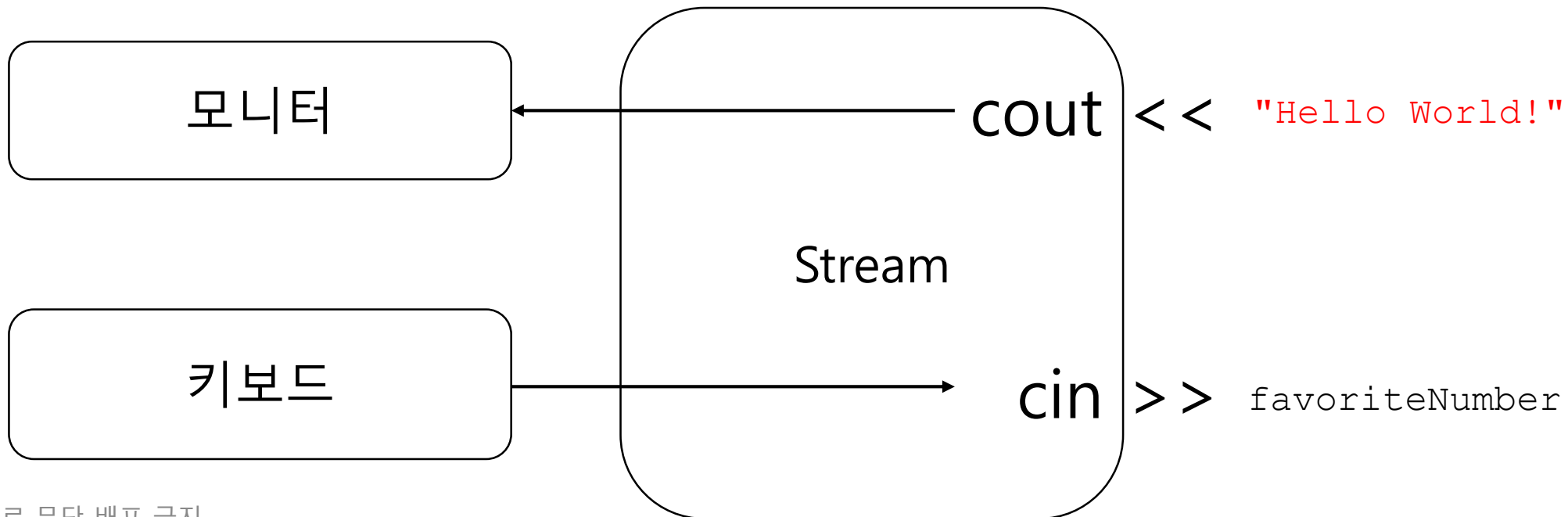
`Std::`로 인해 길어지기 때문에

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     int favoriteNumber;
6
7     cout << "Enter ...";
8     cin >> favoriteNumber;
9     cout << "Amazing!! ..." << endl;
10 }
```

강의 자료에서만 이와 같이 씁니다.

Basic I/O

- cout과 <<
 - C++의 표준 출력 스트림, 삽입 연산자
- cin과 >>
 - C++의 표준 입력 스트림, 추출 연산자

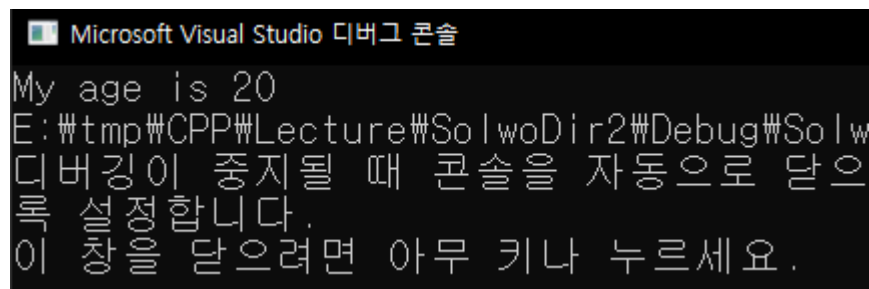


Basic I/O, Output

- cout과 <<

- C++의 표준 출력 스트림, 삽입 연산자
- 순차적인 출력이 가능

```
1 int age = 20;  
2 cout << "My age is" << age;
```



Microsoft Visual Studio 디버그 콘솔

My age is 20
E:\tmp\CPP\Lecture\SolwoDir2\Debug\Solw
디버깅이 중지될 때 콘솔을 자동으로 닫으
록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요.

- 줄바꿈은 필요한 경우 명시해 주어야 함

```
1 cout << "My age is" << age << endl;  
2 cout << "My age is" << age << "\n";
```

Basic I/O, Input

- cin과 >>

- C++의 표준 입력 스트림, 추출 연산자
- 순차적인 입력이 가능

```
1 cin >> myAge >> myHeight;
```

- 변수의 타입에 맞게 해석이 불가능한 입력이 주어질 땐 undetermined behavior

변수와 상수

C++ Program Structure

- 변수와 메모리
 - Bit, byte와 Hex 표현
 - 메모리
 - 변수
 - 변수의 정의, 초기화, 사용
- 변수의 타입
- 상수

C++ Program Structure

- 변수와 메모리
 - Bit, byte와 Hex 표현
 - 메모리
 - 변수
 - 변수의 정의, 초기화, 사용
- 변수의 타입
- 상수

Bit, Byte, Hex

- 0.5 byte = 4 bit = $2^4 = 16$ 개의 숫자 표현 가능 (양수 범위 0~15)
- 1 byte = 8 bit = $2^8 = 256$ 개의 숫자 표현 가능 (양수 범위 0~255)
- Hexadecimal(Hex, 16진수)
 - 16진수 2개로 1 byte를 보기 쉽게 표현 ← Hex 사용 이유
 - Hex 표현임을 알리기 위해 앞에 0x를 붙임
 - 0x 부분은 값으로써의 의미는 없음

Bit 표현 (1 byte)	16진수 (Hex)	10진수
0000 0000	0x 00	0
0000 0001	0x 01	1
0000 1010	0x 0A	10
1010 0000	0x A0	160
1010 1011	0x AB	171
1111 0000	0x F0	240
1111 1111	0x FF	255

몇몇 수만 예시로
표시함. 0~255 사이값
모두 표현 가능

Bit 표현 (0.5 byte)	10진수	16진수 (Hex)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Memory

- 프로그래밍을 할 때 사용하는 가장 중요한 자원!

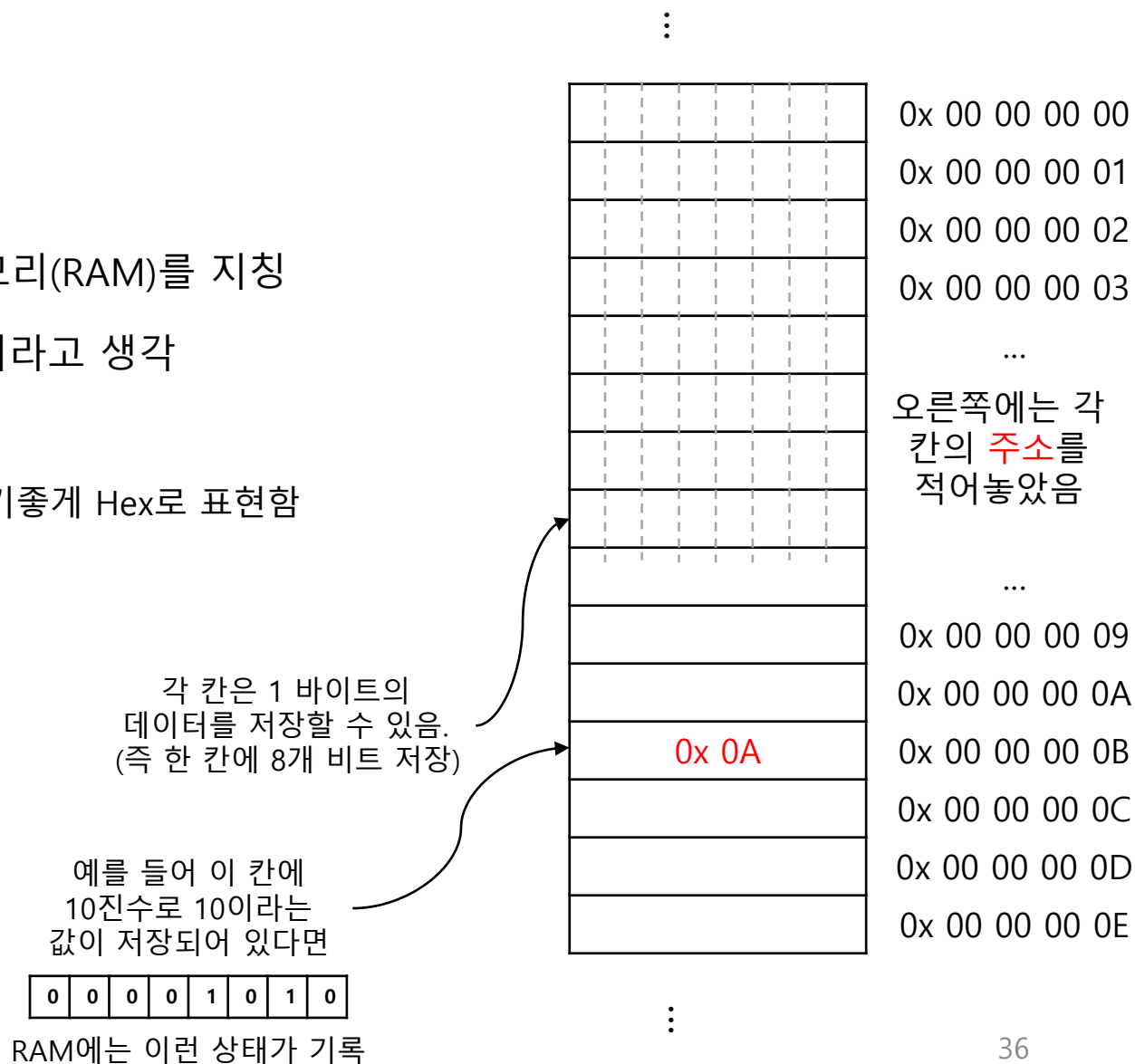
- CPU가 가장 중요한 게 아닙니다!

- 메모리는 읽고 쓸 수 있는 **바이트** 공간의 집합

- 메모리에는 다양한 종류가 있으나 여기서는 메인 메모리(RAM)를 지칭
- 층 하나(byte)마다 8개의 방(bit)이 있는 커다란 호텔이라고 생각
 - 각 방은 0 또는 1인 상태
 - 우리는 방(bit)단위로는 생각할 필요가 없음, 따라서 보기 좋게 Hex로 표현함

- 각 층에는 번호가 붙어있고, 이를 "주소"라고 함

- 주소 또한 Hex로 표현함



Memory and Variables

- 우리 개발 환경에서, 정수를 저장할 때 타입에 따라 아래와 같은 바이트 용량을 사용하도록 되어 있음

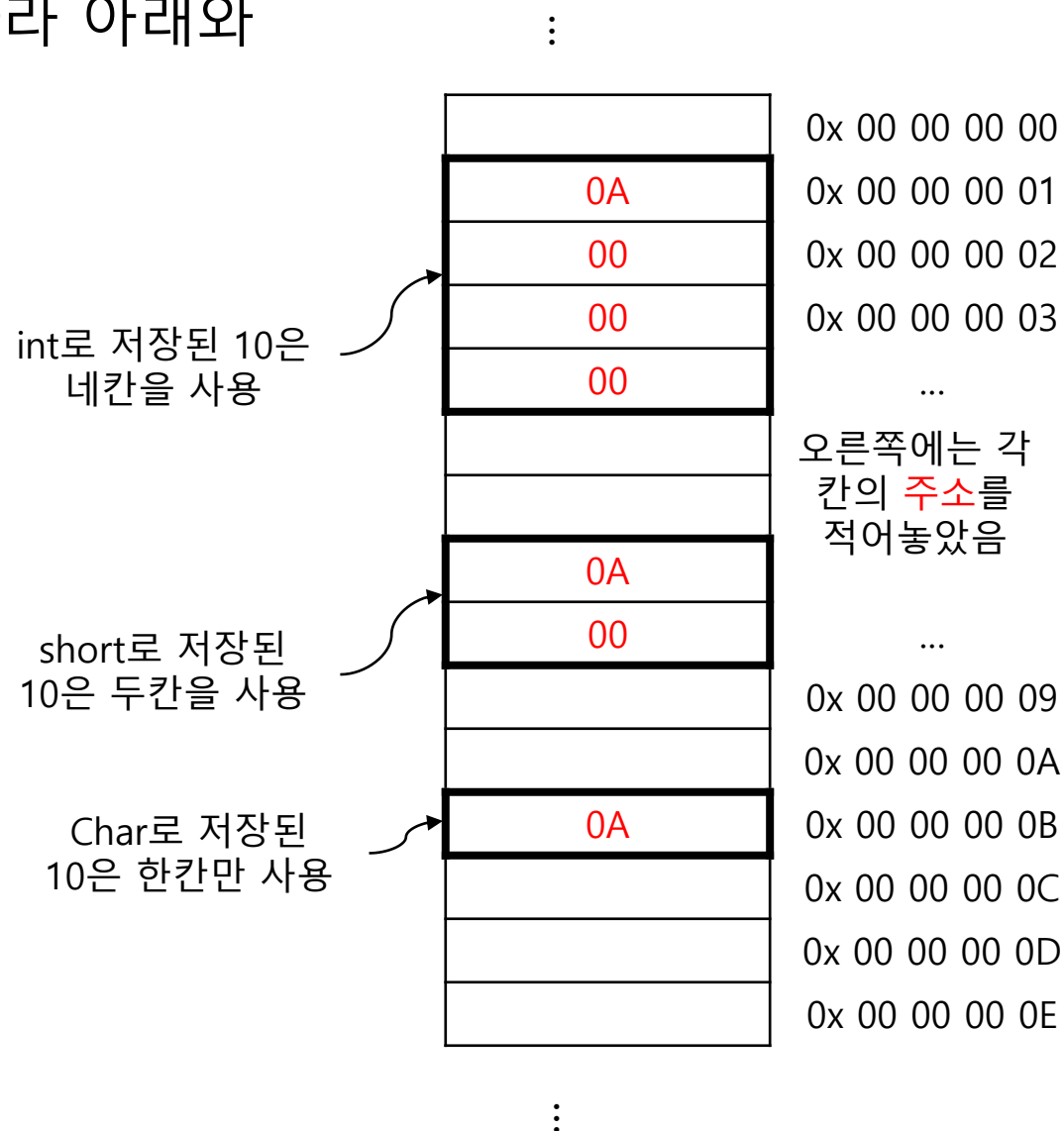
- Char: 1 바이트
- Short: 2 바이트
- Int: 4 바이트

- 따라서 **10이라는 값**을 저장하는 경우

- Char: 0A 로 저장
- Short: 00 0A 로 저장
- Int: 00 00 00 0A 로 저장

(*당연히 Hex지만, 0x는 생략했습니다.)

(*왜 그림에서는 순서가 반대인지 궁금하신 분은 추가 슬라이드 참고)

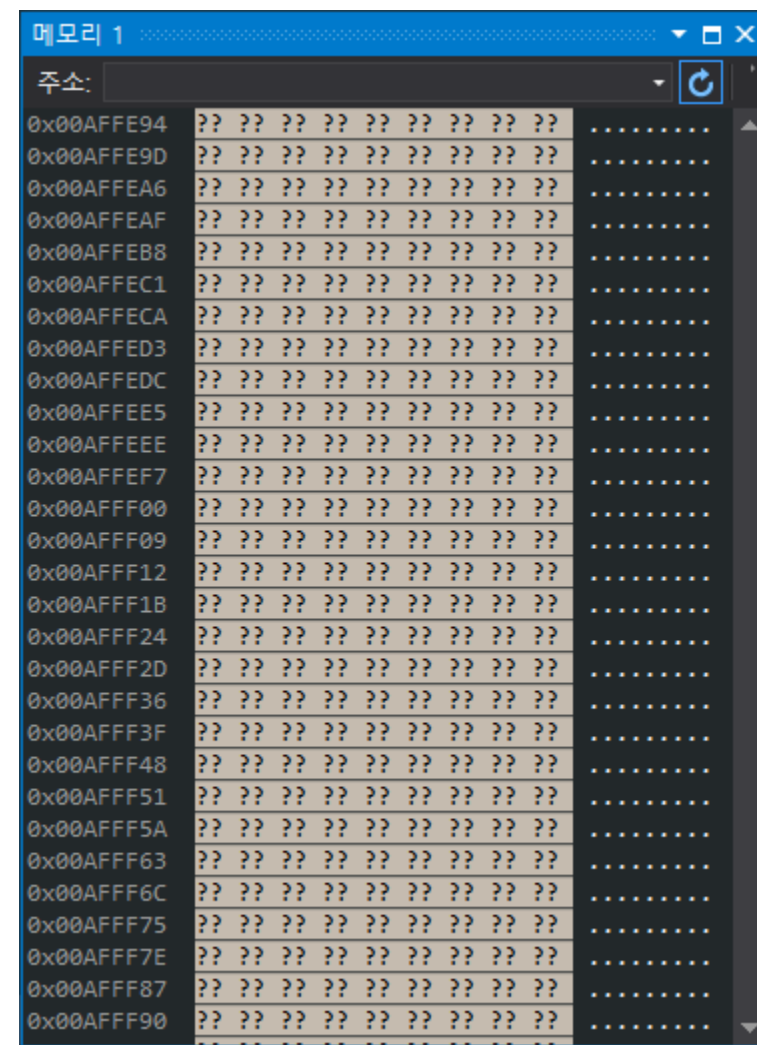
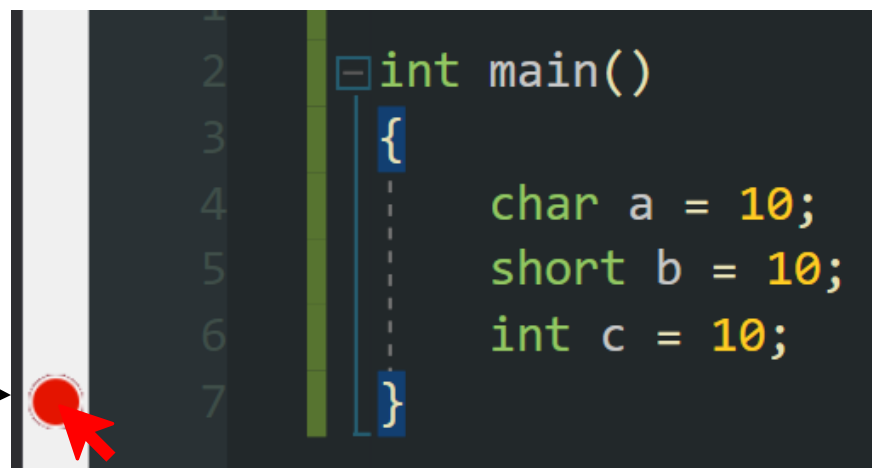


Memory Window in VS2022

- 직접 눈으로 확인할 수 있음!

1. 아래와 같이 코드를 작성한 뒤, main함수 끝부분 왼쪽을 클릭하여 중단점을 걸어놓음
 - (중단점이 무엇인지는 다음 시간에 설명)
2. F5를 눌러 디버깅을 시작
3. 디버그→창→메모리→메모리 1을 눌러 오른쪽과 같은 창이 나오도록 함

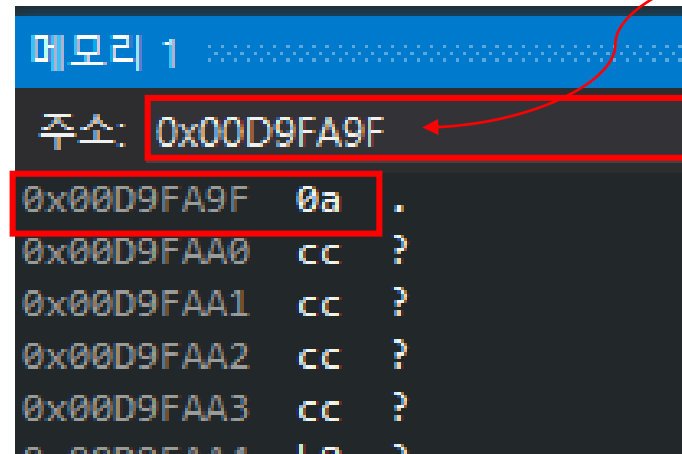
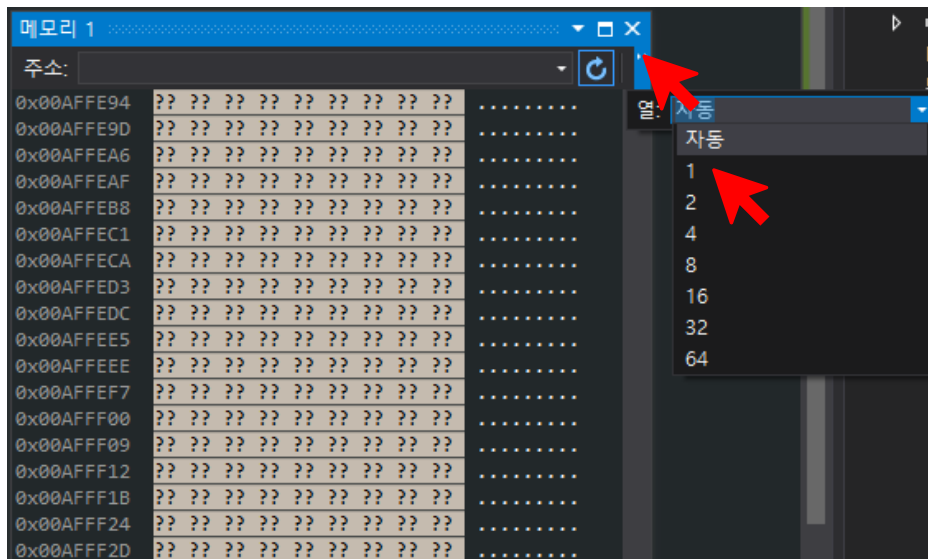
끝부분을 클릭하면
빨간 동그라미로
중단점이 표시



무섭게 생긴 창이 등장!

Memory Window in VS2022

- 오른쪽 위 버튼을 눌러 메모리가 1열로 표시되게 수정 (1줄에 1 byte만 표시)
- 위쪽 "주소:" 오른쪽에 "&a"를 입력해 a가 저장된 위치의 메모리 관찰
 - &를 변수앞에 붙이면 주소를 의미
 - 실제 주소는 여러분 PC에서는 다를 수 있고, 프로그램을 실행할 때마다도 다릅니다.

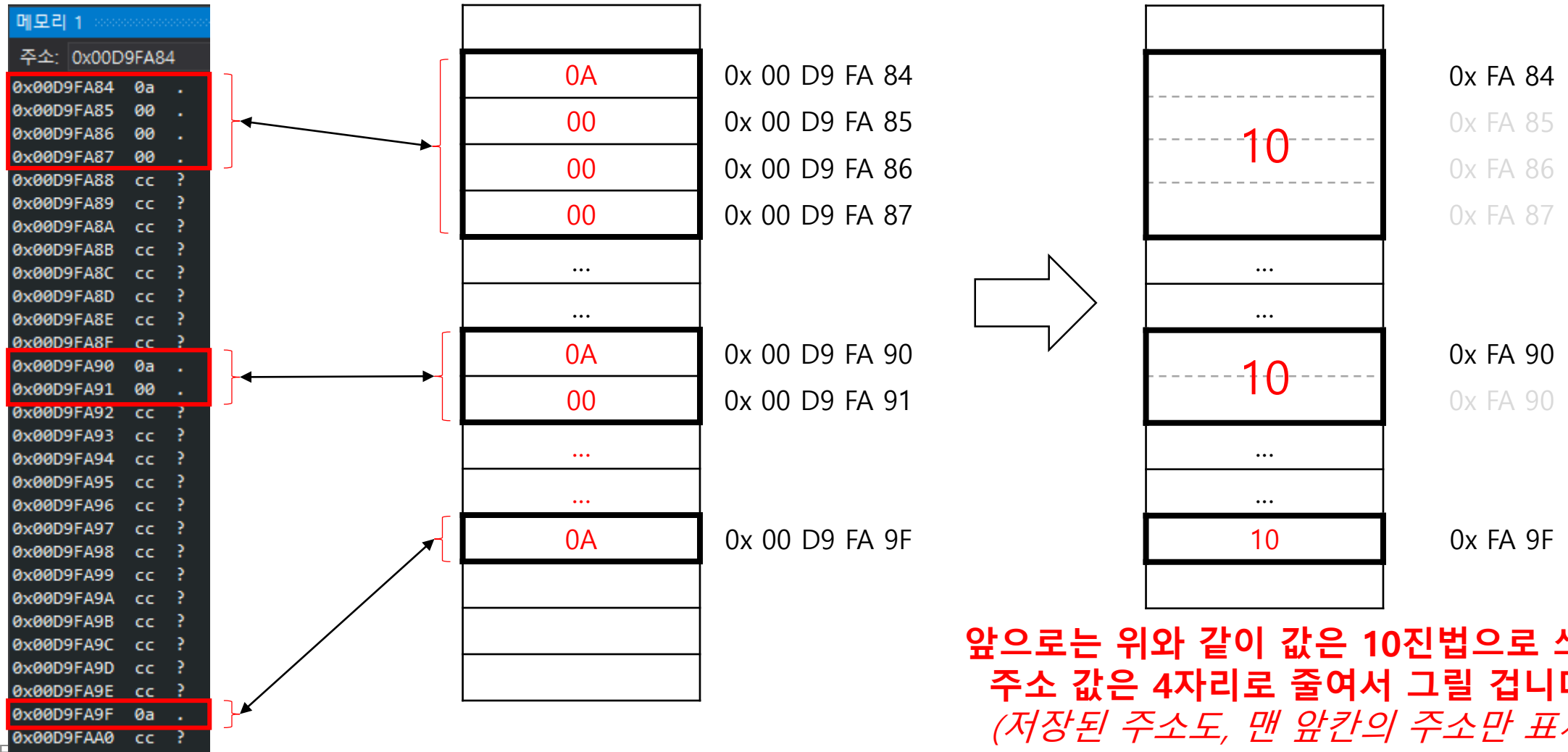


여기에 &a를 입력하면
변수 a가 저장되어 있는
주소 값으로 자동 변경됨

변수 a가 저장된 주소는
0x00D9FA9F이며 거기에 "0a"가
저장된 것을 확인 가능!

Memory Window in VS2022

6. 위쪽 주소에 다시 &c를 입력하고 창 길이를 확장해 보면, 37페이지 그림과 동일



앞으로는 위와 같이 값은 10진법으로 쓰고,
주소 값은 4자리로 줄여서 그릴 겁니다!
(저장된 주소도, 맨 앞칸의 주소만 표시)

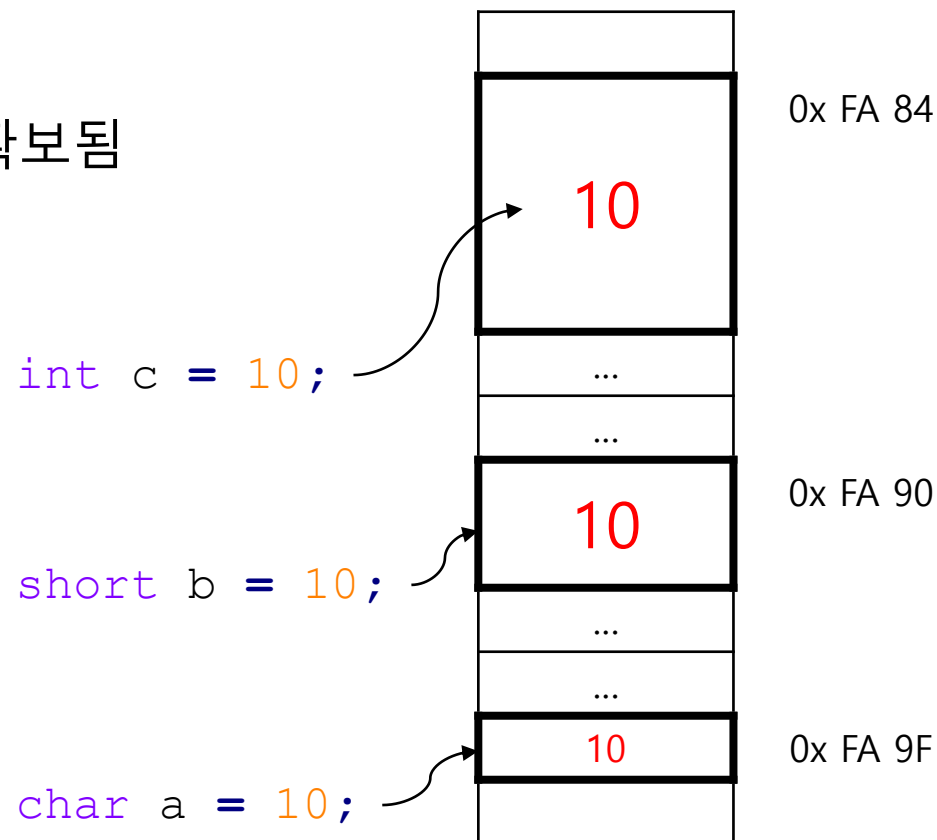
● 따라서 변수란...

- 메모리의 주소에 붙이는 이름!
- 변수를 만들면, 메모리에 변수를 위한 공간(바이트)이 확보됨
 - 공간의 크기(몇 칸)는 변수의 타입에 따라 결정
- 변수에 값을 대입하면, 그 메모리 주소에 값이 기록됨
 - 물리적으로는 0101010...로
- 변수를 만든다 → **변수를 정의**한다고 표현

● 변수라는 개념이 없다면

- 아래와 같이 불편하게 코드를 작성해야 했을 것

```
1 int 0xFA84 = 10;  
2 0xFA84 = 0xFA90 + 6;
```



Definition and Initialization

● 변수의 정의

- (C++에서) 변수를 정의할 때는 반드시 타입을 명시해야 함
 - 바이트를 몇 칸이나 확보해야 하는지 미리 알아야 하기 때문

```
1 char a;  
2 int age;  
3 double rate;  
4 std::string name;
```

● 변수의 초기화

- 변수를 정의하면서 초기값을 설정하는 것을 초기화라고 함
 - 변수를 생성하는 시점부터, 메모리에 값이 저장되어 있음

```
1 char a = 10;  
2 int age = 21;  
3 double rate = 0.85;  
4 std::string name = "Hyunki Kim";
```

```
print(sys.getsizeof(int()))
```

```
24
```

```
Process finished with exit code 0
```

**파이썬과 같이 미리 타입을 명시하지 않아도 되는 언어들은 얼마나 큰 데이터가 들어올지 모르기 때문에 일단 크게 확보해 놓습니다.
(효율성 저하. 대신 편의성 증가)*

Using a Variable

- 변수의 사용

- 변수 이름은 변수가 확보한 그 메모리에 접근하기 위해 사용됨
- 메모리에 값을 **읽고 쓰는** 것이 변수의 사용임

```
1 a = 20; //메모리 내 a변수 위치에 20을 쓰기  
2 printf("%d", a); //메모리 내 a변수 위치의 값을 읽어서 출력
```

- 간단한 규칙

- 이름 앞에 타입(ex, int, float)이 붙어있다? → 변수의 정의 (& 초기화)
- 이름 앞에 아무것도 없다? → 변수의 사용

C++ Program Structure

- 변수와 메모리
 - Bit, byte와 Hex 표현
 - 메모리
 - 변수
 - 변수의 정의, 초기화, 사용
- 변수의 타입
- 상수

Primitive Data Types

● Integer

■ 정수를 표현

형식 이름	바이트	기타 이름	값의 범위
int	4	signed	-2,147,483,648 ~ 2,147,483,647
unsigned int	4	unsigned	0 ~ 4,294,967,295
__int8	1	char	-128 ~ 127
unsigned __int8	1	unsigned char	0 ~ 255
__int16	2	short, short int 및 signed short int	-32,768 ~ 32,767
unsigned __int16	2	unsigned short, unsigned short int	0 ~ 65,535
__int32	4	signed, signed int 및 int	-2,147,483,648 ~ 2,147,483,647
unsigned __int32	4	unsigned, unsigned int	0 ~ 4,294,967,295
__int64	8	long long, signed long long	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
unsigned __int64	8	unsigned long long	0 ~ 18,446,744,073,709,551,615
short	2	short int, signed short int	-32,768 ~ 32,767
unsigned short	2	unsigned short int	0 ~ 65,535
long	4	long int, signed long int	-2,147,483,648 ~ 2,147,483,647
unsigned long	4	unsigned long int	0 ~ 4,294,967,295
long long	8	없음(그러나 __int64와 동일)	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
unsigned long long	8	없음(그러나 unsigned __int64와 동일)	0 ~ 18,446,744,073,709,551,615

Primitive Data Types

● Floating point

- 실수를 표현

형식 이름	바이트	기타 이름	값의 범위
float	4	없음	3.4E+/-38(7개의 자릿수)
double	8	없음	1.7E+/-308(15개의 자릿수)
long double	double과 동일	없음	double과 동일

- 실수는 메모리에 어떻게 저장되나요 → 2학년 "컴퓨터 구조" 강의에서 배움

● Boolean

- true / false

형식 이름	바이트	기타 이름	값의 범위
bool	1	없음	false 또는 true

Using the sizeof operator

- sizeof 연산자

- 타입 또는 변수의 바이트 단위 크기를 리턴

```
1 sizeof(int) // 타입의 크기
2 sizeof(double) // 타입의 크기
3 sizeof(favoriteNumber) // 변수의 크기
```

- 변수의 최대/최소값

```
1 INT_MAX
2 INT_MIN
3 FLT_MAX
4 FLT_MIN
```

C++ Program Structure

- 변수와 메모리
 - Bit, byte와 Hex 표현
 - 메모리
 - 변수
 - 변수의 정의, 초기화, 사용
- 변수의 타입
- 상수

Constant

- 상수

- 변수와 유사하지만, 초기화 이후 변할 수 없는 값
- 사용 목적 → **실수 방지, 프로그램의 견고함**
 - 실제 현장에서, 그리고 다른 사람과 협업을 할 때 매우 중요한 매너!

- 상수의 종류

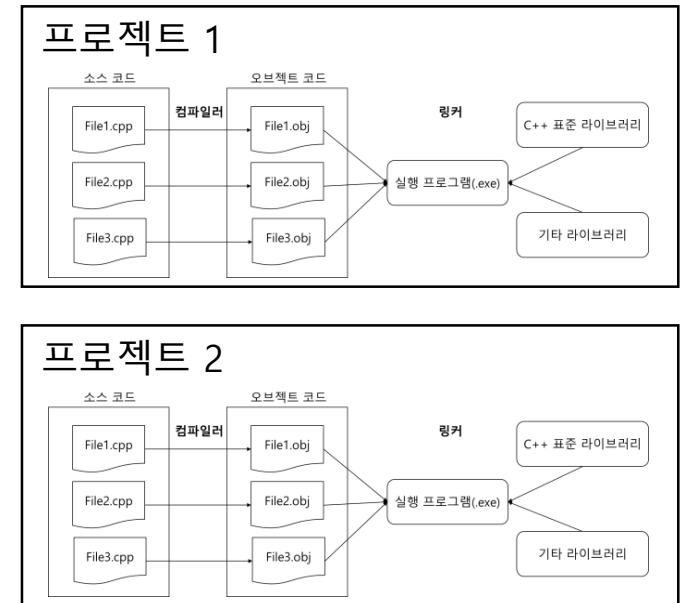
- 리터럴 상수 : 12, 3.14, "Hyunki" (r-values)
- 선언 상수 : `const int favoriteNumber = 50;`
- 상수 표현 : `constexpr int age = 20;`
- 열거형
- Defined : `#define pi 3.1415926`

추가 슬라이드

Solution and Project

- 비주얼 스튜디오의 솔루션은 단순한 폴더 개념
 - 관련 있는 여러 프로젝트를 하나로 묶어서 관리하기 위함
- 프로젝트는 개별적인 빌드 결과(exe, lib, dll)를 만드는 단위
 - 프로젝트 내 소스파일들로 개별적인 exe, lib, dll을 빌드하여 생성함
- “시작 프로젝트” 설정은 F5를 눌렀을때 실행할 결과물을 해당 프로젝트의 결과물(exe)로 설정함

솔루션



Endianness

- 메모리에 바이트를 저장하는 순서는 프로세서에 따라 다름
- 높은 자리수를 낮은 메모리 주소에 저장하는 것을 Big endian
- 낮은 자리수를 낮은 메모리 주소에 저장하는 것을 Little endian이라 함
- Big endian은 보기엔 헷갈리지만 효율성이 높아 주로 사용됨
 - <https://ko.wikipedia.org/wiki/%EC%97%94%EB%94%94%EC%96%B8>

00 00 00 0A

높은 자리수 낮은 자리수

0A	0x 00 00 00 08
00	0x 00 00 00 09
00	0x 00 00 00 A0
00	0x 00 00 00 A1

Little endian의 경우

00	0x 00 00 00 08
00	0x 00 00 00 09
00	0x 00 00 00 A0
0A	0x 00 00 00 A1

Big endian의 경우

Casting

- Implicit vs Explicit type cast

```
1 int main()
2 {
3     int a = 5;
4     double b = a; //implicit cast
5
6     double c = 1.23;
7     int d = c; //implicit cast
8
9     double e = 1.23;
10    int f = (int)c; //c-style explicit cast
11
12    double g = 1.23;
13    int h = static_cast<int>(g); //c++-style explicit cast (compile-time check + clear syntax)
14
15    double i = static_cast<Player>(g); //Error!
16    double j = static_cast<Player*>(&g); //Error!
17    double k = reinterpret_cast<Player*>(&g); //Okay! Type punning, reinterpretation
18
19    //dynamic_cast for run-time check
20
21 }
```