

[C++ Programming]

1. Intro & Variable [개인 정리]

Contents

- I. 빌드 프로세스
- II. 오류의 종류
- III. 기본 구조, 기능과 용어
 - 키워드 / 지시자 / 연산자
 - 전처리 지시문
 - main() 함수
 - 주석
 - namespace
 - 표준 입출력
- IV. 변수와 메모리
 - Bit, byte와 Hex 표현
 - 메모리
 - 변수
 - 변수의 정의, 초기화, 사용
- V. 변수의 타입
- VI. 상수

■ 목표

- ✓ (핵심) 객체 지향 프로그래밍 기법의 이해
- ✓ 메모리 레이아웃과 명령어에 따른 메모리의 변화 이해
- ✓ 자료구조/JAVA/윈도우즈프로그래밍/알고리즘/컴퓨터그래픽스 의 기반이 되는 파트

■ 학습 초반 부분 중요

: 메모리 레이아웃과 명령어에 의한 메모리 변화 이해 + 디버거를 사용한 관찰 방법 학습

■ 기억할 3가지

- ✓ 강의를 구경하지 말 것 -> 본인이 직접 하나하나 타이핑 해 볼 것!
- ✓ 스스로를 너무 믿지 말기
- ✓ 질문하기

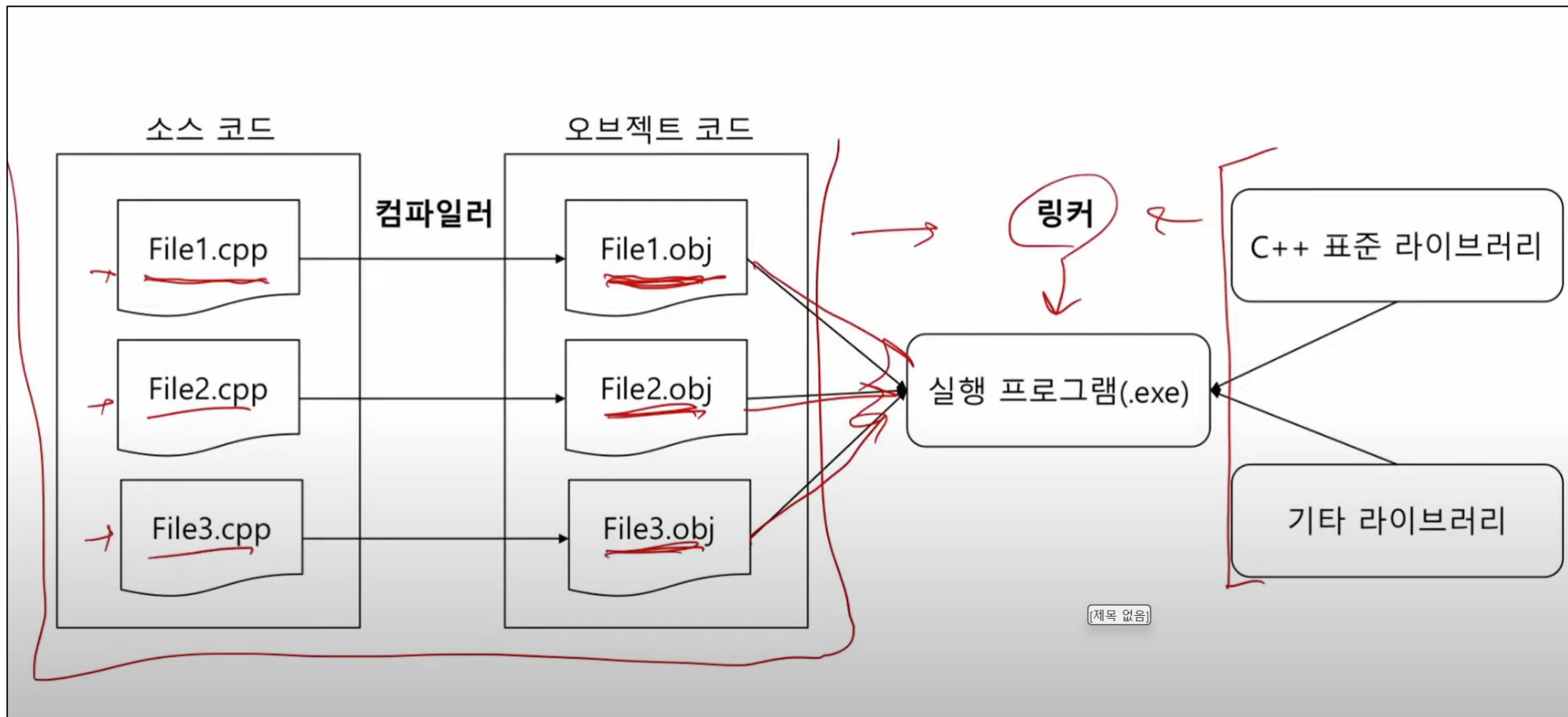
I. 빌드 프로세스

- 프로그래밍 언어 : 고수준의 소스 코드 작성에 사용, Human-readable
 - 텍스트 에디터: .cpp 소스코드 / .h 헤더 파일의 편집기
- 오브젝트 코드: 컴퓨터가 실행할 수 있는 코드, Machine-reable

- 컴파일러 : 소스코드를 오브젝트 코드로 변환하는 도구
- 링커: 오브젝트 코드를 실행 파일(exe)로 변환하는 도구
- 테스트&디버깅: 프로그램에 존재하는 오류를 찾고 수정함

- IDE (Integrated Development Environment) : 통합 개발 환경
 - 텍스트에디터 + 컴파일러 + 링커 + (디버거) → Visual Studio가 디버거가 잘되어있다.

C++ Build Process



C++ Build Process

II. 오류의 종류

▪ Compiler Errors

- 프로그래밍 언어 규칙 위반
- 문법 오류 : 코드자체 오류
- 의미 오류

▪ Compiler Warnings

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

▪ Linker Errors

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

▪ Runtime Errors

- 프로그램 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 (대부분의 경우) → 테스트를 통해서 수정 해야함!

III. 기본 구조, 기능과 용어

■ 키워드

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

■ 식별자

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

■ 연산자

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

■ 위와 같은 요소들이 모여 "문법" 을 이룸



Check out the latest remote job listings from Authentic Jobs.

ADS VIA CARBON

Page Discussion

Planned Maintenance
The site is in a temporary read-only mode to facilitate some long-overdue software updates. We apologize for any inconvenience this may cause!

C++ reference
C++11, C++14, C++17, C++20, C++23, C++26 | Compiler support C++11, C++14, C++17, C++20, C++23, C++26

Language Preprocessor – Comments ASCII chart Basic concepts Keywords Names (lookup) Types (fundamental types) The main function Modules (C++20) Contracts (C++26) Expressions Value categories Evaluation order Operators (precedence) Conversions – Literals Constant expressions Statements if – switch for – range-for (C++11) while – do-while Declarations – Initialization Functions – Overloading Coroutines (C++20) Classes (unions) Templates – Exceptions Freestanding implementations Standard library (headers) Named requirements Language support library Program utilities Signals – Non-local jumps Basic memory management Variadic functions source_location (C++20) Comparison utilities (C++20) Type support – type_info numeric_limits – exception initializer_list (C++11) Coroutine support (C++20) Contract support (C++26)	Diagnostics library Assertions – System error (C++11) Exception types – Error numbers basic_stacktrace (C++28) Debugging support (C++26) Memory management library Allocators – Smart pointers Memory resources (C++17) Metaprogramming library (C++11) Type traits – ratio integer_sequence (C++14) General utilities library Function objects – hash (C++11) Swap – Type operations (C++11) Integer comparison (C++20) pair – tuple (C++11) optional (C++17) expected (C++23) variant (C++17) – any (C++17) bitset – Bit manipulation (C++20) Containers library vector – deque – array (C++11) list – forward_list (C++11) inplace_vector (C++26) hive (C++26) map – multimap – set – multiset unordered_map (C++11) unordered_multimap (C++11) unordered_set (C++11) unordered_multiset (C++11) Container adaptors span (C++20) – mdspan (C++23) Iterators library Ranges library (C++20) Range factories – Range adaptors generator (C++23) Algorithms library Numeric algorithms	Strings library basic_string – char_traits basic_string_view (C++17) Text processing library Primitive numeric conversions (C++17) Formatting (C++20) – Localization text_encoding (C++26) Regular expressions (C++11) basic_regex – Algorithms Default regular expression grammar Null-terminated sequence utilities: byte – multibyte – wide Numerics library Common math functions Mathematical special functions (C++17) Mathematical constants (C++20) Basic linear algebra algorithms (C++26) Data-parallel types (SIMD) (C++26) Pseudo-random number generation Floating-point environment (C++11) complex – valarray Date and time library Calendar (C++20) – Time zone (C++20) Input/output library Print functions (C++23) Stream-based I/O – I/O manipulators basic_istream – basic_ostream Synchronized output (C++20) File systems (C++17) Concurrency support library (C++11) thread – jthread (C++20) atomic – atomic_flag atomic_ref (C++20) – memory_order Mutual exclusion – Condition variables Futures – Semaphores (C++20) latch (C++20) – barrier (C++20) Safe Reclamation (C++26) Execution support library (C++26) Feature test macros (C++20)
--	---	---

■ 전처리기

- 컴파일 이전에 처리됨
- “#”으로 시작

[예시]

```
#include <iostream>
```

```
#include "myFile.h"
```

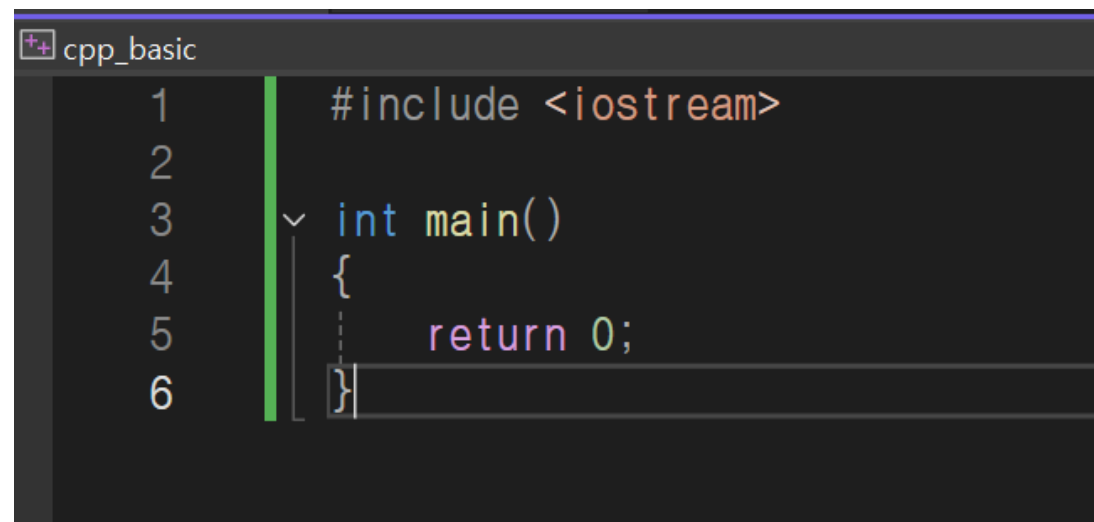
```
#ifdef
```

```
#ifndef
```

```
#define
```

```
#undef
```

```
#pragma
```



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

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

```
#include <iostream>
```

```
int main()
```

```
{
```

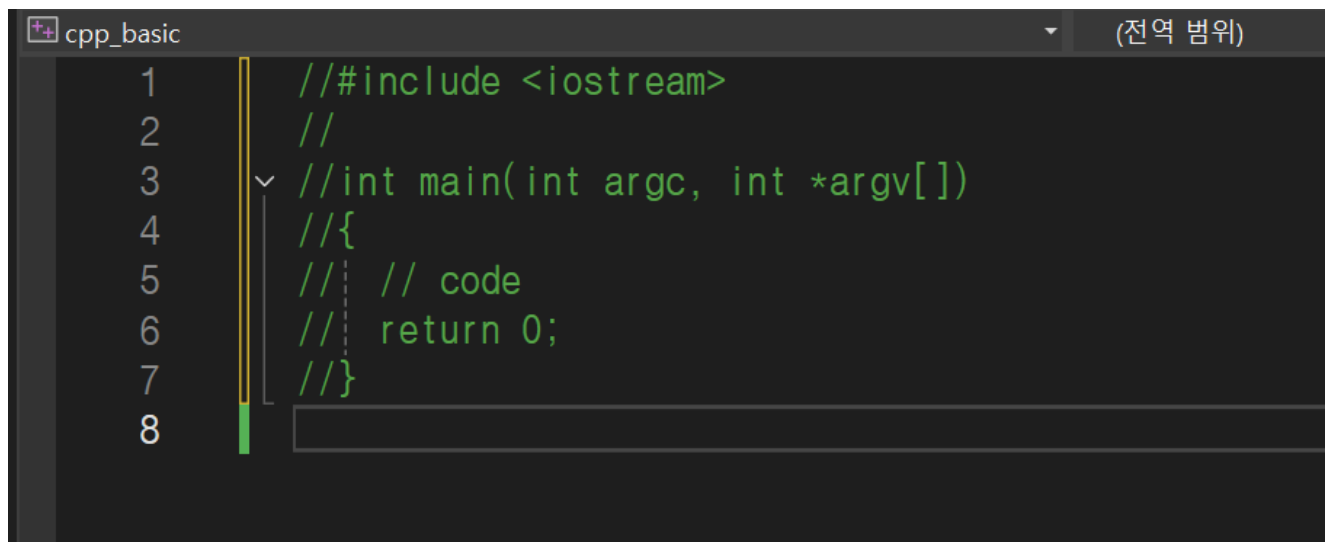
```
    int a = 0;
```

```
    return 0;
```

```
}
```

```
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
```

- 주석 (Ctrl + K, Ctrl + C / Ctrl + K, Ctrl + U)
 - 프로그래머가 읽을 수 있는 정보를 제공하기 위함 (협업, 유지보수)
 - 전처리 단계에서 무시되기 때문에 프로그램의 동작과는 무관
 - `"/"/`, `"/* */"`



The screenshot shows a code editor window titled 'cpp_basic' with a dropdown menu set to '(전역 범위)'. The code is as follows:

```
1 // #include <iostream>
2 //
3 // int main(int argc, int *argv[])
4 // {
5 //     // code
6 //     return 0;
7 // }
8
```

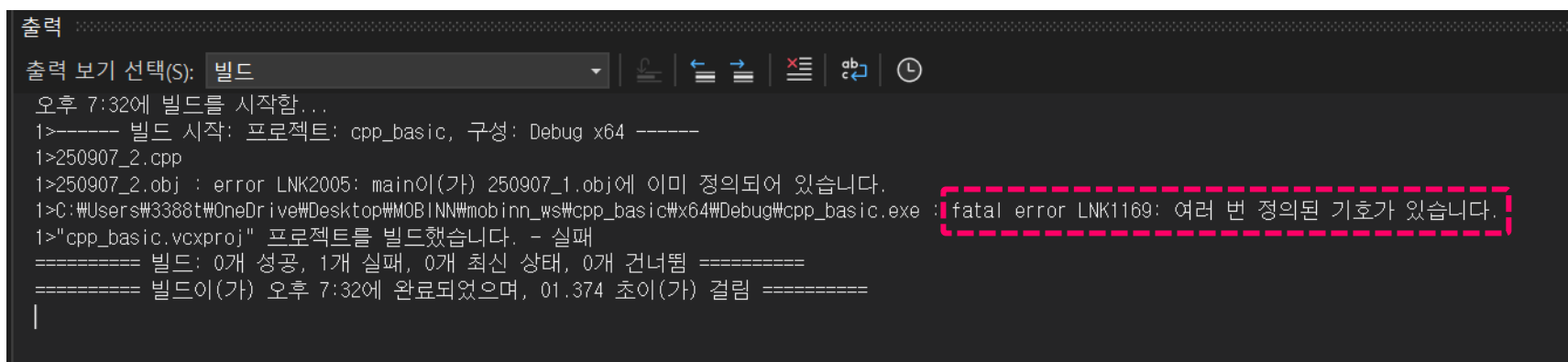
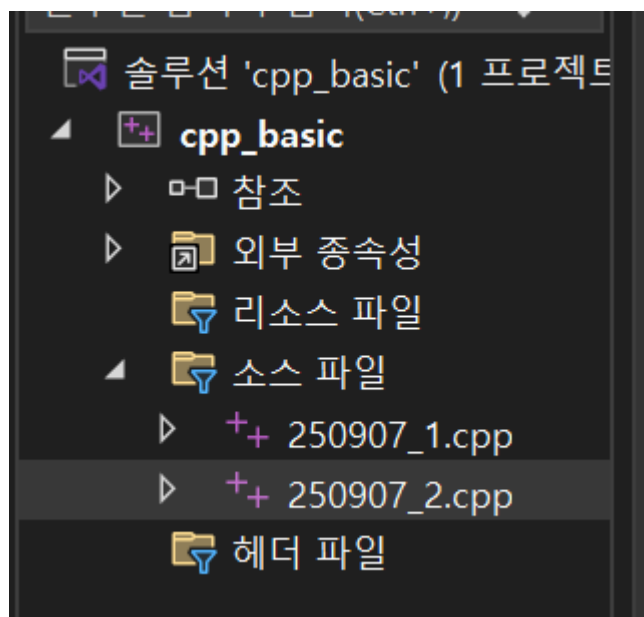
- 모든 C++ 프로그램은 하나의 main 함수를 가져야 함
 - 프로그램의 진입점 : 프로그램이 실행되면 가장 먼저 실행되는 함수
 - 리턴값 0이 올바른 프로그램 실행을 의미함! (명식적으로 써줄 것!)

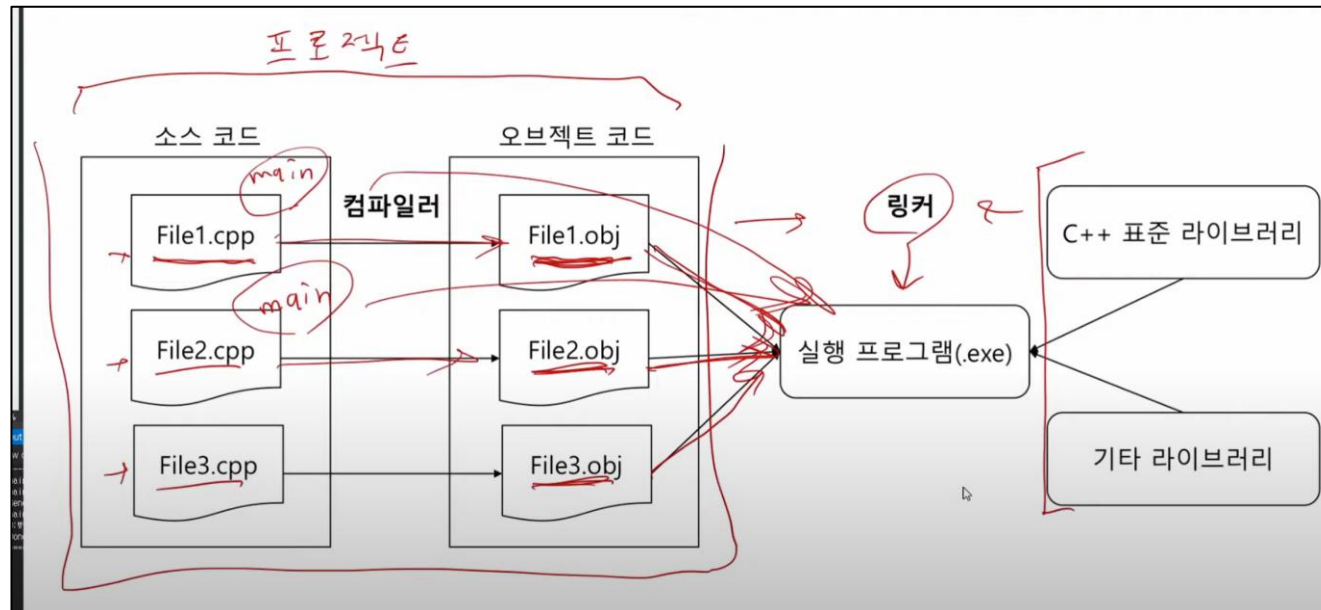
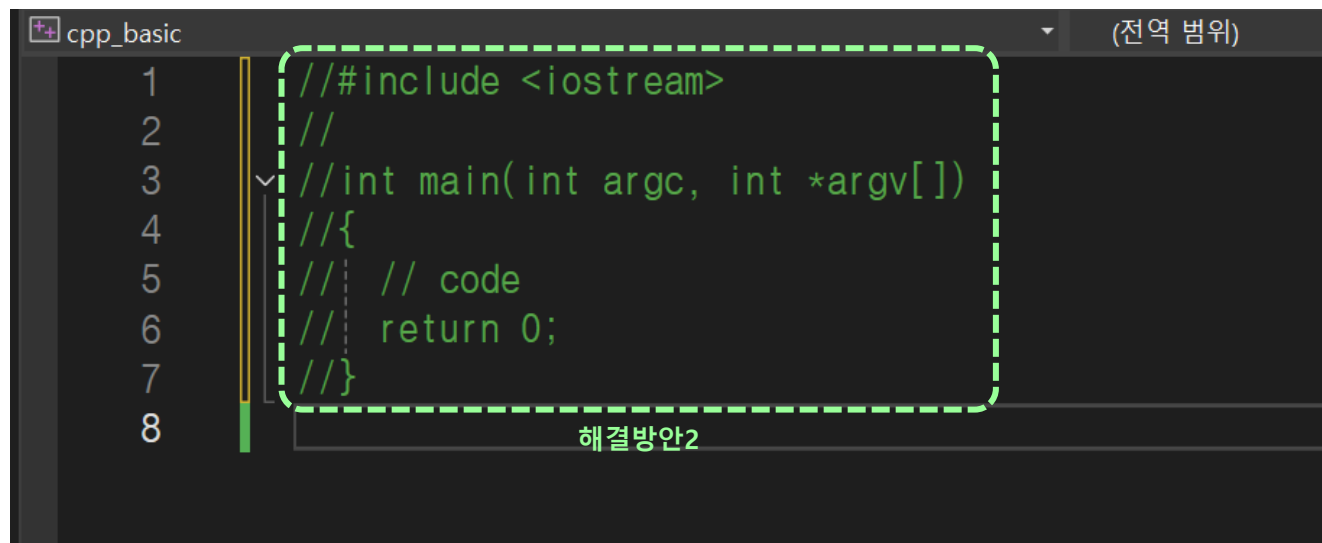
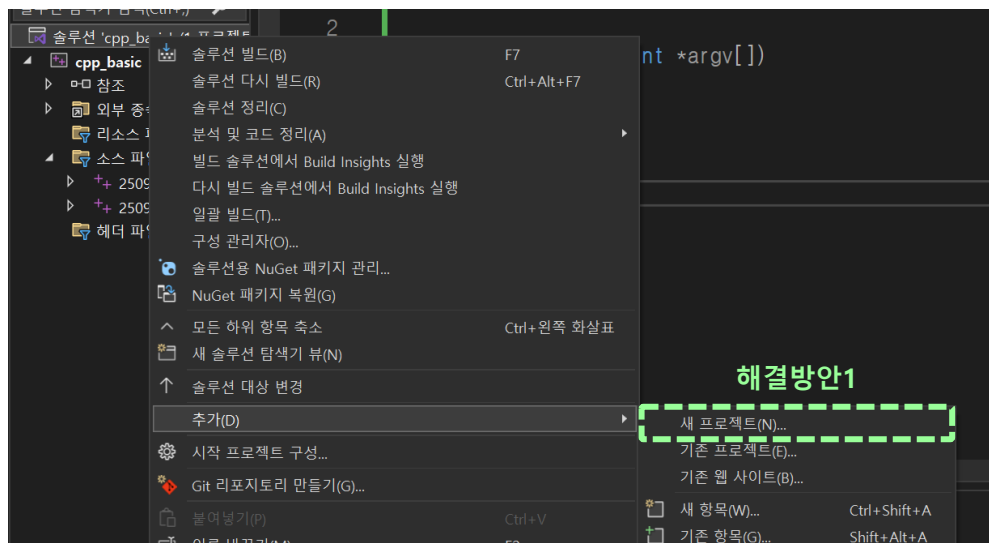
- 두 가지 버전

```
cpp_basic
1      #include <iostream>
2
3      int main()
4      {
5          // code
6          return 0;
7      }
```

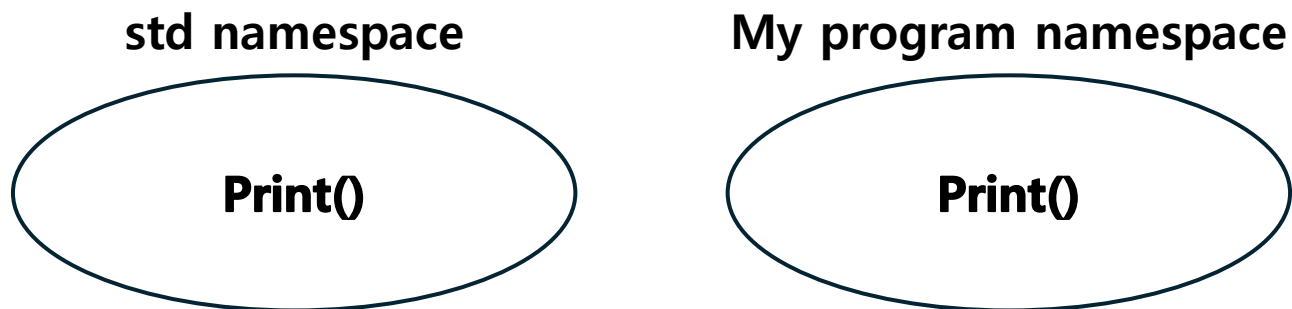
```
cpp_basic
1      #include <iostream>
2
3      int main(int argc, int *argv[])
4      {
5          // code
6          return 0;
7      }
8
```

- 모든 C++ 프로그램은 하나의 main 함수를 가져야 함
 - 동일 프로젝트에 cpp파일을 계속 추가하여 작성하면 오류 발생 → 하나의 프로그램에 2개 이상의 main 함수가 존재하기 때문에
- 해결방안 [1] 솔루션을 우클릭하여 추가 → '새 프로젝트'로 별도의 프로젝트를 만들어 사용
 - 이때, 현재 빌드하고자 하는 프로젝트를 우클릭하여 "시작 프로젝트로 설정" 필요
- 해결방안 [2] 현재 빌드하고자 하는 cpp이외의 코드는 전체 주석처리하고 빌드





- `std::cout`
- 충돌 방지를 위함 (쉽게 말해서 폴더의 개념)
 - 외부 라이브러리와 구현한 소스 코드간의 이름 충돌 가능성
- 코드의 "그룹화 " 로 이해하면 된다!
 - 서로 다른 namespace로 그룹화하여 충돌을 방지할 수 있음
 - 그래서 C++은 큰 프로그램을 만들기에 적합한 용어
- "::" : scope resolution operator



```
cpp_basic (전역 범위)
1 #include <iostream>
2
3 void fuction()
4 {
5     std::cout << "Function 1";
6 }
7
8 void fuction()
9 {
10    std::cout << "Function 2";
11 }
12
13 int main()
14 {
15     return 0;
16 }
```

99 % 문제가 검색되지 않음

출력

출력 보기 선택(S): 빌드

오후 7:40에 빌드를 시작함...

1>----- 빌드 시작: 프로젝트: cpp_basic, 구성: Debug x64 -----

1>250907_2.cpp

1>C:\Users#3388t\OneDrive\Desktop\MOBINN\mobinn_ws\cpp_basic\cpp_basic\250907_2.cpp(8,6): error C2084: 'void fuction(void)' 함수에 이미 본문이 있습니다.

1>C:\Users#3388t\OneDrive\Desktop\MOBINN\mobinn_ws\cpp_basic\cpp_basic\250907_2.cpp(3,6):

1>'fuction'의 이전 정의를 참조하십시오.

1>"cpp_basic.vcxproj" 프로젝트를 빌드했습니다. - 실패

===== 빌드: 0개 성공, 1개 실패, 0개 최신 상태, 0개 건너뛸 =====

===== 빌드(가) 오후 7:40에 완료되었으며, 00.410 초(가) 걸림 =====

The screenshot displays the Microsoft Visual Studio IDE. The main window shows a C++ source file named `cpp_basic.cpp`. The code defines two namespaces, `A` and `B`, each containing a `function()` that prints a message. The `main` function calls `A::function()`. The code is annotated with green dashed boxes highlighting the namespace definitions. The bottom panel shows the 'Output' window with the following text:

```

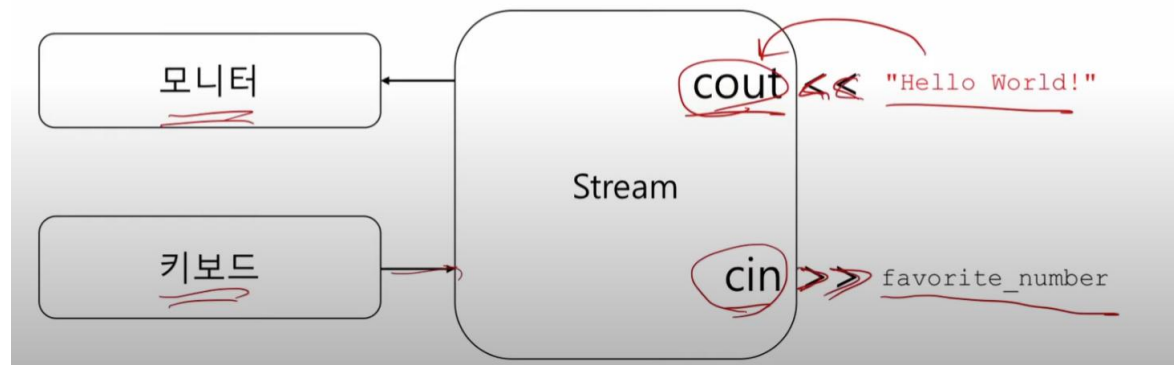
C:\Users\3388t\OneDrive\Desktop\MOBINN\mobinn_ws\cpp_basic\x64\Debug\cpp_b
종료되었습니다.
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅]
록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
  
```

- using namespace

- 특정 namespace 내의 함수들을 사용하겠다는 선언
- 남용은 금물!
 - Using namespace를 모든 코드에 넣는다면 namespace의 기능을 상실함...
 - 코드가 정말 길어질 경우에 코드의 단축 표현을 위해 사용할 뿐!

```
cpp_basic (전역 범위)
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int favorite_number;
8
9      cout << "Enter your favorite number between 1 and 100: ";
10     cin >> favorite_number;
11     cout << "Amazing!! That's my favorite number too!" << std::endl;
12
13     return 0;
14 }
```

- **cout**과 <<
 - C++의 표준 출력 스트림, 삽입 연산자
- **cin**과 >>
 - C++의 표준 입력 스트림, 추출 연산자



```
cpp_basic (전역 범위)
1  #include <iostream>
2
3  int main()
4  {
5      int age = 27;
6      float height = 170.5f;
7
8      std::cout << "my age is" << age << " Hello~! " << std::endl;
9      std::cout << height;
10
11     return 0;
12 }
```

IV. 변수와 메모리

- 0.5 byte = 4 bit = $2^4 = 16$ 개의 숫자 표현 가능 (0~15)
- 1 byte = 8 bit = $2^8 = 256$ 개의 숫자 표현 가능 (0~255)
- Hexadecimal(Hex, 16진수)
 - 16진수 2개로 "1 Byte"를 보기 쉽게 표현하기 위해서 ← Hex 사용 이유
 - Hex 표현임을 알리기 위해 앞에 0x

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

- **"메모리"**는 읽고 쓸 수 있는 바이트의 집합
 - 메모리에는 다양한 종류가 있으나 여기서는 'RAM'을 지칭
 - Hex만 쓰는 칸이 나눠진 커다란 메모장이라고 생각하기!
 - 실제로는 0, 1 binar로 저장되지만 , 불편하니 Hex로 표현하기로!
- 각 바이트에는 번호가 붙어 있고, 이를 **"주소"**라고 부름
 - 주소 또한 컴퓨터이기 때문에 Hex로 표현되어 있음

집합

이는 RAM을 지칭

근데 Hex만 쓰는

하니 Hex로 표현함

를 "주소"라고 부름

되어 있음

각 칸은 1 바이트의
데이터를 저장할
수 있음

예를 들어 이 칸에
10진수로 10이라는
값이 저장되어
있다면



- 우리 환경에서, 정수를 저장할 때 타입에 따라 아래와 같은 용량의 바이트를 사용하도록 되어 있음

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

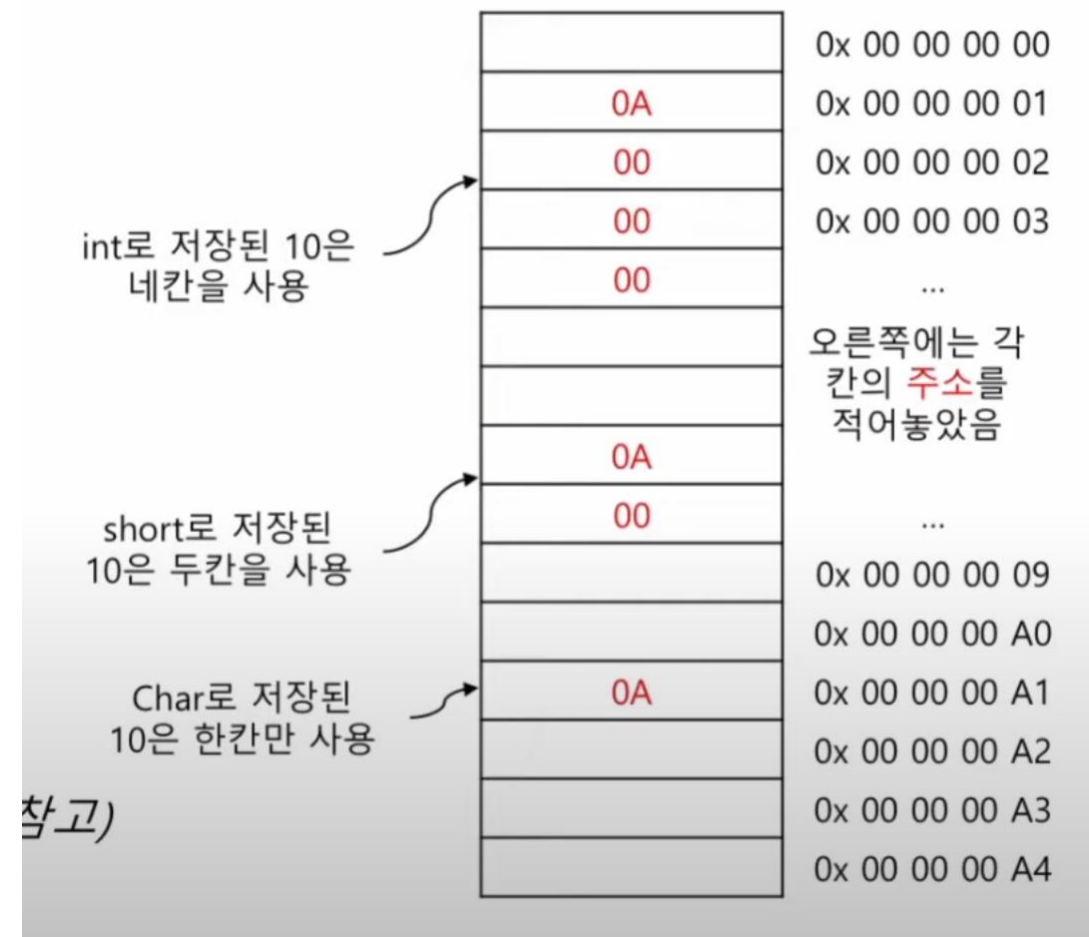
- 따라서 10이라는 값을 저장하는 경우

- Char: 0A로 저장
- Short: 00 0A로 저장
- Int: 00 00 00 0A로 저장

(당연히 Hex지만, 이제 0x는 생략)

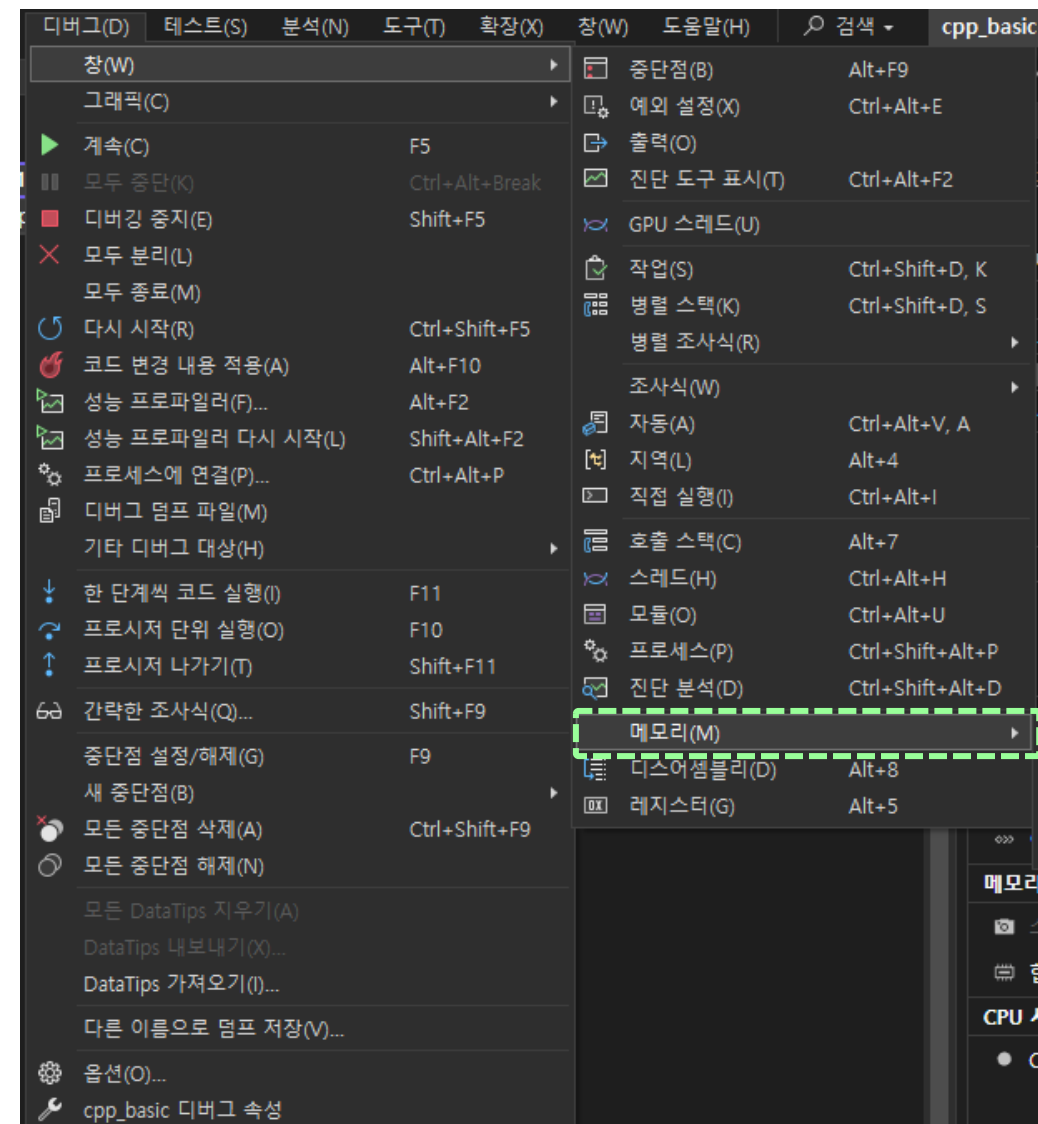
(왜 순서가 반대인지 궁금한 분들을 슬라이드 참고)

아래와 같은



```

cpp_basic (전역 범위) main()
1  #include <iostream>
2
3  int main()
4  {
5      char a = 10;
6      short b = 10;
7      int c = 10;
8      return 0;
9
10 }
```



Memory 1

Address: 0x010FF778

0x010FF778	0a
0x010FF779	00
0x010FF77A	00
0x010FF77B	00
0x010FF77C	cc
0x010FF77D	cc
0x010FF77E	cc
0x010FF77F	cc
0x010FF780	cc
0x010FF781	cc
0x010FF782	cc
0x010FF783	cc
0x010FF784	0a
0x010FF785	00
0x010FF786	cc
0x010FF787	cc
0x010FF788	cc
0x010FF789	cc
0x010FF78A	cc
0x010FF78B	cc
0x010FF78C	cc
0x010FF78D	cc
0x010FF78E	cc
0x010FF78F	cc
0x010FF790	cc
0x010FF791	cc
0x010FF792	cc
0x010FF793	0a
0x010FF794	cc
0x010FF795	cc

Handwritten notes: 10, 0000, 1010, 0, a, and an arrow pointing from the 'a' at address 0x010FF793 to the '0a' at address 0x010FF784.

메모리 1

주소: 0x00007FF7972C1810

주소	값	설명
0x00007FF7972C1810	40	@
0x00007FF7972C1811	55	U
0x00007FF7972C1812	57	W
0x00007FF7972C1813	48	H
0x00007FF7972C1814	81	?
0x00007FF7972C1815	ec	?
0x00007FF7972C1816	48	H
0x00007FF7972C1817	01	.
0x00007FF7972C1818	00	.
0x00007FF7972C1819	00	.
0x00007FF7972C181A	48	H
0x00007FF7972C181B	8d	?
0x00007FF7972C181C	6c	l
0x00007FF7972C181D	24	\$
0x00007FF7972C181E	20	.
0x00007FF7972C181F	48	H
0x00007FF7972C1820	8d	?
0x00007FF7972C1821	0d	.
0x00007FF7972C1822	53	S
0x00007FF7972C1823	f8	?
0x00007FF7972C1824	00	.
0x00007FF7972C1825	00	.
0x00007FF7972C1826	e8	?
0x00007FF7972C1827	31	1
0x00007FF7972C1828	fb	?
0x00007FF7972C1829	ff	.
0x00007FF7972C182A	ff	.
0x00007FF7972C182B	90	?
0x00007FF7972C182C	c6	?
0x00007FF7972C182D	45	E
0x00007FF7972C182E	04	.
0x00007FF7972C182F	0a	.
0x00007FF7972C1830	b8	?
0x00007FF7972C1831	0a	.
0x00007FF7972C1832	00	.
0x00007FF7972C1833	00	.
0x00007FF7972C1834	00	.
0x00007FF7972C1835	66	f
0x00007FF7972C1836	89	?
0x00007FF7972C1837	45	E
0x00007FF7972C1838	24	\$
0x00007FF7972C1839	c7	?
0x00007FF7972C183A	45	E

메모리 1

주소: &a

메모리 1

주소: 0x00000064122FF864

0x00000064122FF864	0a
0x00000064122FF865	00
0x00000064122FF866	00
0x00000064122FF867	00
0x00000064122FF868	00
0x00000064122FF869	00
0x00000064122FF86A	00
0x00000064122FF86B	00
0x00000064122FF86C	00
0x00000064122FF86D	00

메모리 1

주소: &b

메모리 1

주소: 0x00000064122FF864

0x00000064122FF864	0a
0x00000064122FF865	00
0x00000064122FF866	00
0x00000064122FF867	00
0x00000064122FF868	00
0x00000064122FF869	00
0x00000064122FF86A	00
0x00000064122FF86B	00
0x00000064122FF86C	00
0x00000064122FF86D	00

메모리 1

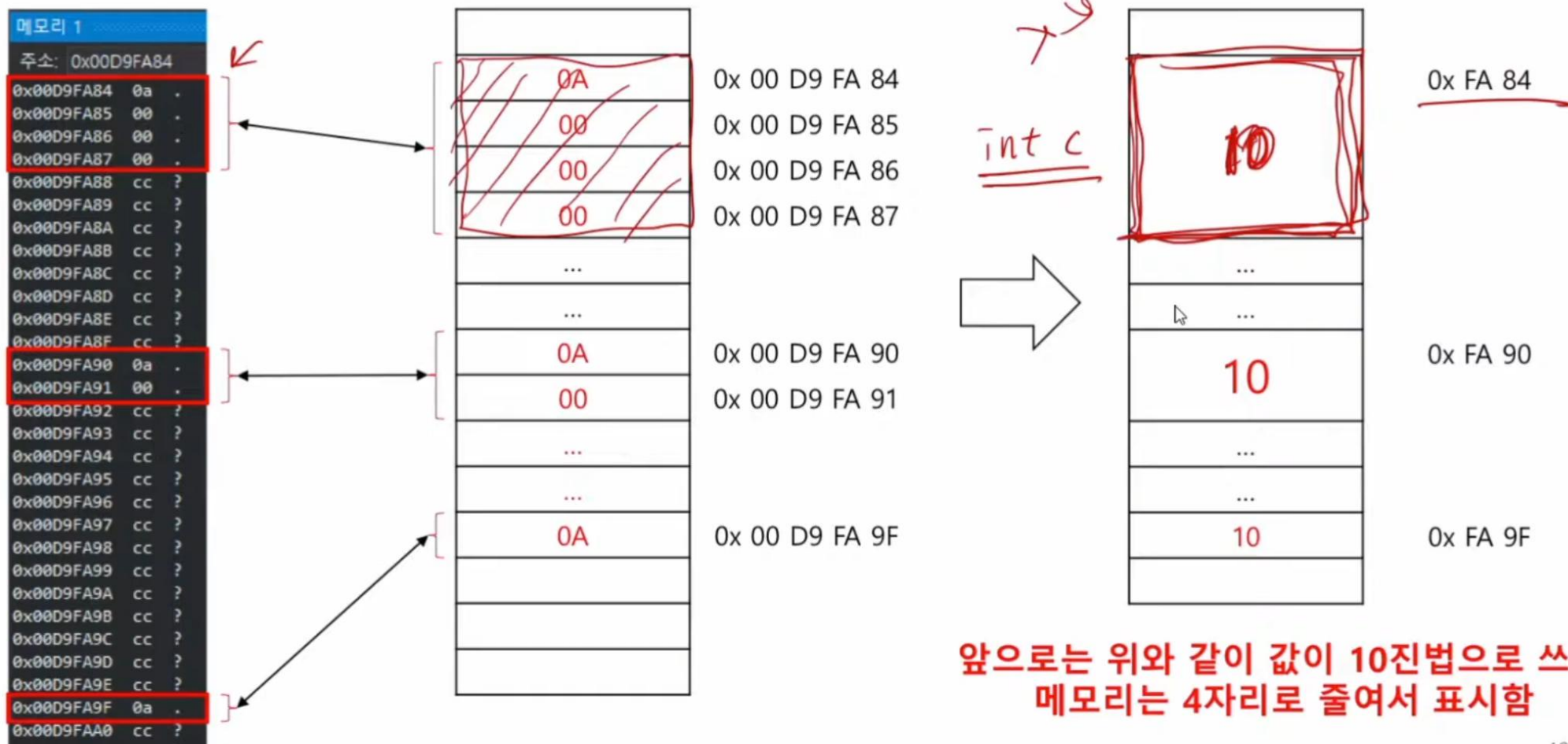
주소: &c

메모리 1

주소: 0x00000064122FF884

0x00000064122FF884	0a
0x00000064122FF885	00
0x00000064122FF886	00
0x00000064122FF887	00
0x00000064122FF888	c8
0x00000064122FF889	ed

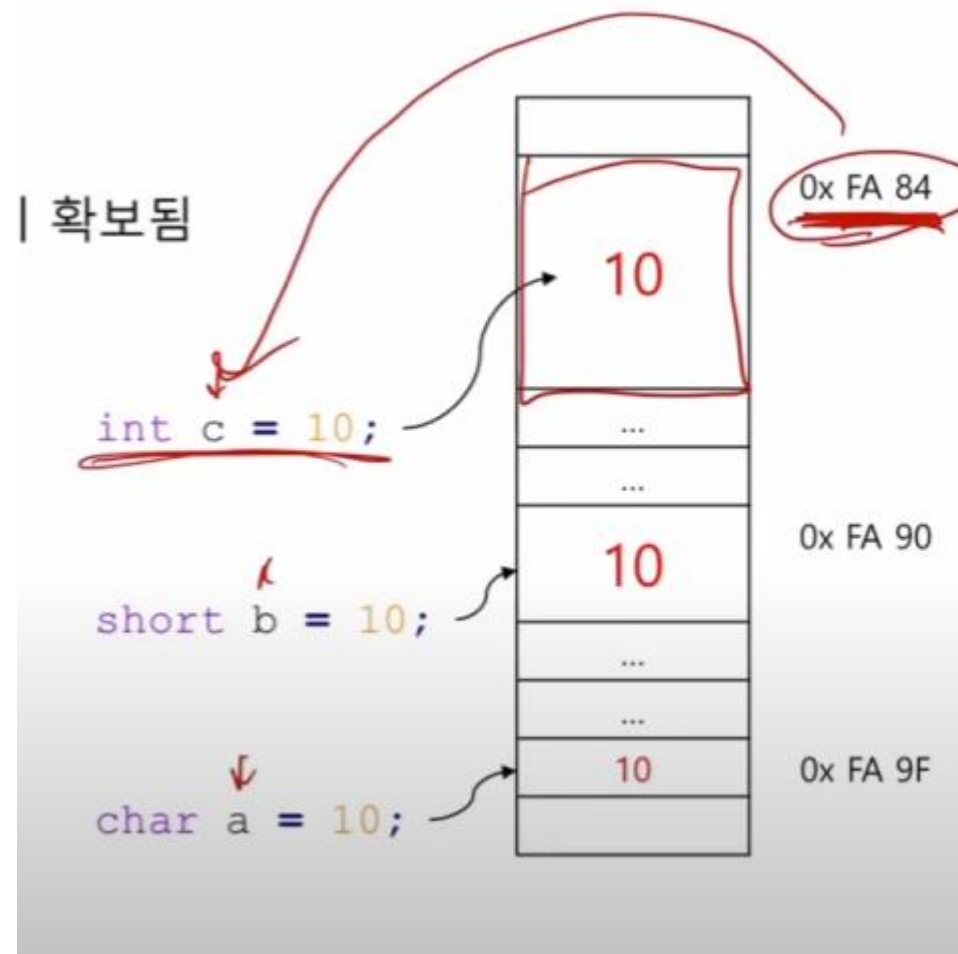
6. 위쪽 주소에 다시 &c를 입력하고 창 길이를 확장해 보면, 37페이지 그림과 동일



- 따라서 Variable(변수)란...
 - 메모리의 주소에 붙이는 이름!
 - 변수를 만들면, 메모리에 변수를 위한 공간(byte)이 확보됨
 - 공간의 크기(몇칸)는 변수의 타입에 따라 결정
 - 변수에 값을 대입하면, 해당 공간에 값이 기록됨
 - 변수를 만든다 → "변수를 정의"한다고 표현

- 변수라는 개념이 없다면...
 - 아래와 같이 불편하게 코드를 작성해야 했을 것..;

Ex) `int 0xFA84 = 10;`
`0xFA84 = 0xFA90 + 6;`



- 변수라는 개념이 없다면...
 - (C++에서는) 변수를 정의할 때는 반드시 타입을 명시해야 함!
 - 바이트를 얼마나 확보해야 하는지 알아야 하기 때문에!
- 변수의 초기화
 - 변수를 정의하면서 초기값을 설정하는 것을 초기화라고 함
 - 변수를 생성하는 시점부터, 메모리에 값이 저장되어 있음
 - 즉, 변수를 생성하는 시점부터 선언되어 있는 것(초기화) vs 변수를 생성하고 이후 값을 넣는 것

```
char a = 10;
```

```
char b;
```

```
b = 10;
```

```
int main()
{
    int a;
    char b;
    double c;
}
```

```
int main()
{
    int a = 21;
    char b = 10;
    double c = 0.78;
}
```

- 변수의 사용

- 변수가 정의된 이후에는 해당 메모리에 접근하기 위해 사용됨
- 메모리에 값을 읽고 쓰는 것이 변수의 사용임

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

- 법칙

- 이름 앞에 타입(int, float...)이 붙어 있다면 → 변수의 정의
- 이름 앞에 아무것도 없다면 → 변수의 사용

V. 변수의 타입

Integer

- 정수를 표현

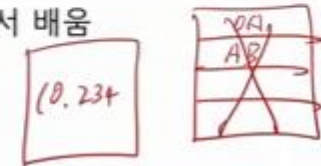
형식 이름	바이트	기타 이름	값의 범위
<u>int</u>	4	signed	-2,147,483,648 ~ 2,147,483,647
<u>unsigned int</u>	4	unsigned	0 ~ 4,294,967,295
<u>_int8</u>	1	char	-128 ~ 127
<u>unsigned _int8</u>	1	unsigned char	0 ~ 255
<u>_int16</u>	2	short, short int 및 signed short int	-32,768 ~ 32,767
<u>unsigned _int16</u>	2	unsigned short, unsigned short int	0 ~ 65,535
<u>_int32</u>	4	signed, signed int 및 int	-2,147,483,648 ~ 2,147,483,647
<u>unsigned _int32</u>	4	unsigned, unsigned int	0 ~ 4,294,967,295
<u>_int64</u>	8	long long, signed long long	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
<u>unsigned _int64</u>	8	unsigned long long	0 ~ 18,446,744,073,709,551,615
<u>short</u>	2	short int, signed short int	-32,768 ~ 32,767
<u>unsigned short</u>	2	unsigned short int	0 ~ 65,535
<u>long</u>	4	long int, signed long int	-2,147,483,648 ~ 2,147,483,647
<u>unsigned long</u>	4	unsigned long int	0 ~ 4,294,967,295
<u>long long</u>	8	없음(그러나 _int64와 동일)	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
<u>unsigned long long</u>	8	없음(그러나 unsigned _int64와 동일)	0 ~ 18,446,744,073,709,551,615

▪ Floating point

- 실수를 표현
- 실수는 메모리에 정수와 다르게 저장됨! → 이는 “컴퓨터 구조” 강의에서 배움

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

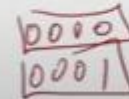
!게 저장되나요 → 2학년 “컴퓨터 구조” 강의에서 배움



▪ Boolean

- True / false

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



1 0

▪ Sizeof 연산자

- 타입 또는 변수의 byte 단위 크기를 return
- sizeof(int), sizeof(double), sizeof(favorite_number) ...

▪ 변수의 최대/최소값

- INT_MAX
- INT_MIN
- FLT_MAX
- FLT_MIN

```
cpp_basic
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << sizeof(int) << std::endl;
6      std::cout << sizeof(float) << std::endl;
7      std::cout << sizeof(double) << std::endl;
8      std::cout << sizeof(bool) << std::endl;
9
10     std::string name = "cho";
11     std::cout << sizeof(name) << std::endl;
12 }
```

Microsoft Visual Studio 디버그

```
4
4
8
1
40
```

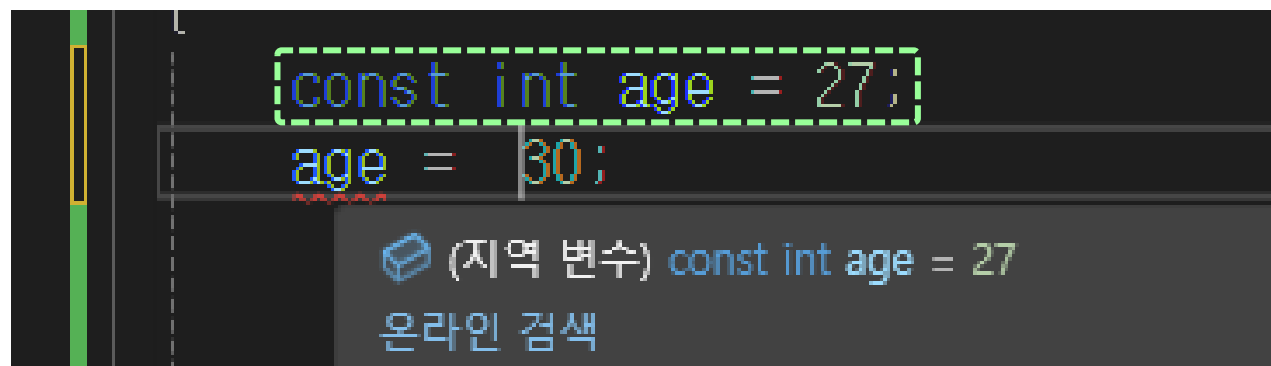
```
cpp_basic
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << INT_MAX << std::endl;
6      std::cout << INT_MIN << std::endl;
7      std::cout << FLT_MAX << std::endl;
8      std::cout << FLT_MIN << std::endl;
9 }
```

Microsoft Visual Studio 디버그

```
2147483647
-2147483648
3.40282e+38
1.17549e-38
```

VI. 상수

- 상수 (Constant)
 - 변수와 유사하지만, 초기화 이후 변할 수 없는 값
 - 사용 목적 → 실수 방지, 프로그램의 견고함
- 상수의 종류
 - 리터럴 상수 : 12, 3.14, "Cho" (r-values)
 - 선언 상수 : `const int age = 27;`
 - 상수 표현 : `constexpr int age = 27;`
 - 열거형
 - Defined : `#define pi 3.1415926`



```
const int age = 27;  
age = 30;
```

(지역 변수) `const int age = 27`
온라인 검색