

독하게 시작하는 C 프로그래밍

전문 개발자를 희망하는 분들을 위한
프로그래밍 입문

v1.22 (2023-10-01)

넌넌한 개발자 최호성 (cx8537@naver.com)

YouTube: 넌넌한 개발자 TV

문서 개정이력

[illegible]

시작에 앞서...

- 독하게 시작하는 C 프로그래밍 재개정
- Visual Studio 2022 Community + Win11환경에 맞춰 현행화
- C언어 프로그래밍 입문자 강의
- 2023년 현재 C언어의 위치

적정 학습 기간

- C언어 2개월, 자료구조 2개월, C++
혹은 Java 2개월
- 1일 3시간 수업 + 3시간 실습 조건
- 왕도는 없음!

Part 1

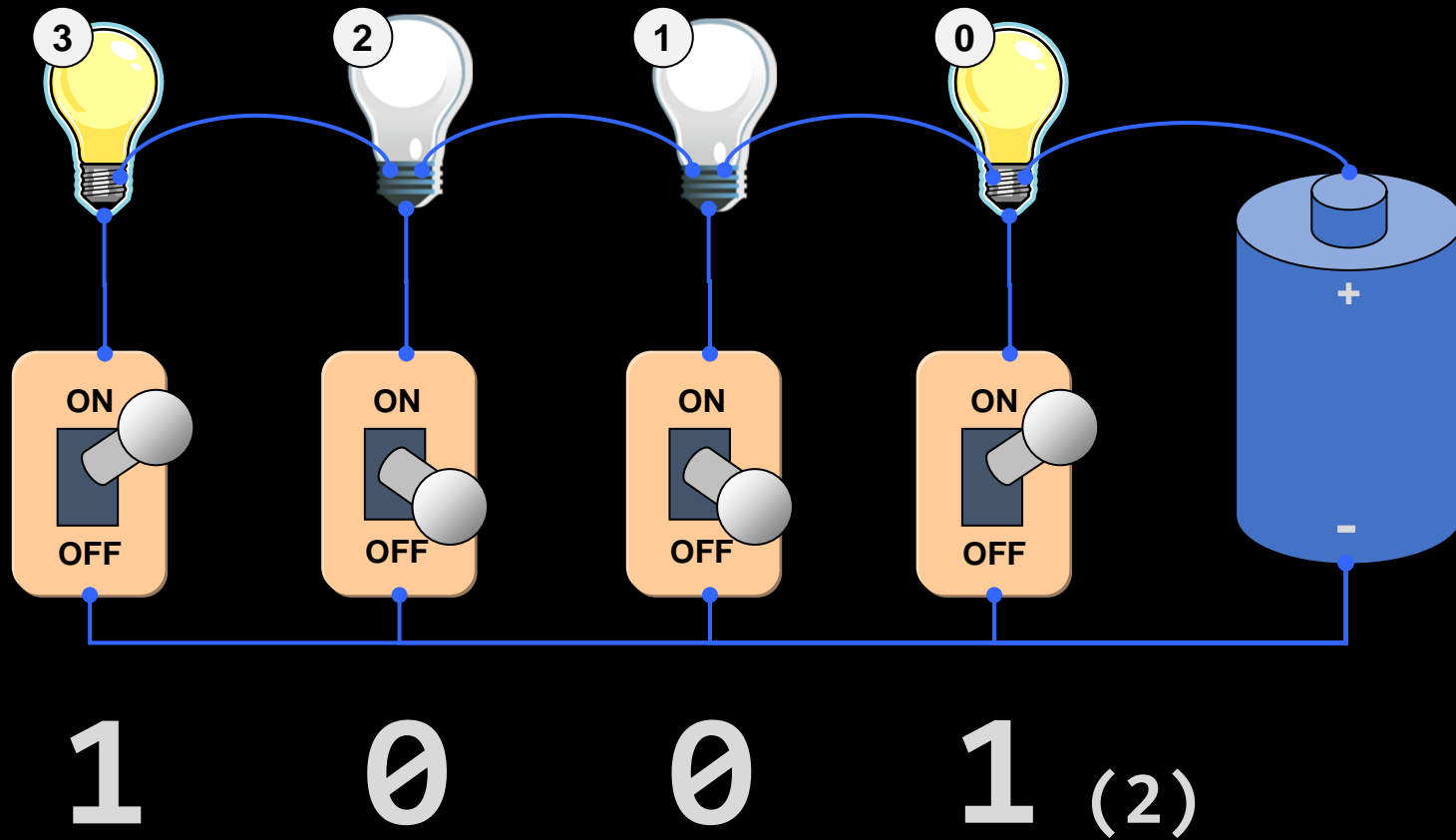
**C를 배우기 전에
알아야 할 것들**

1. 디지털 세계

컴퓨터와 2진법

- 1bit는 전기 스위치 1개를 의미
- 스위치가 On 상태는 1, 흐르지 않는 Off 상태는 0으로 표기
- 여러 스위치(혹은 전선)를 4개씩 묶어주면 4bit

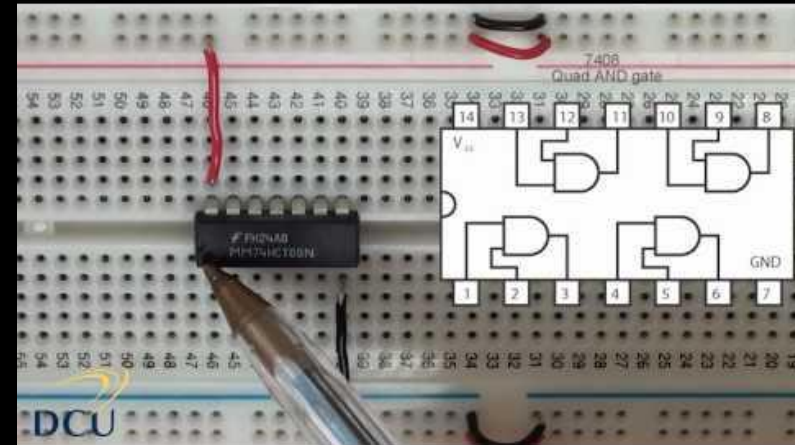
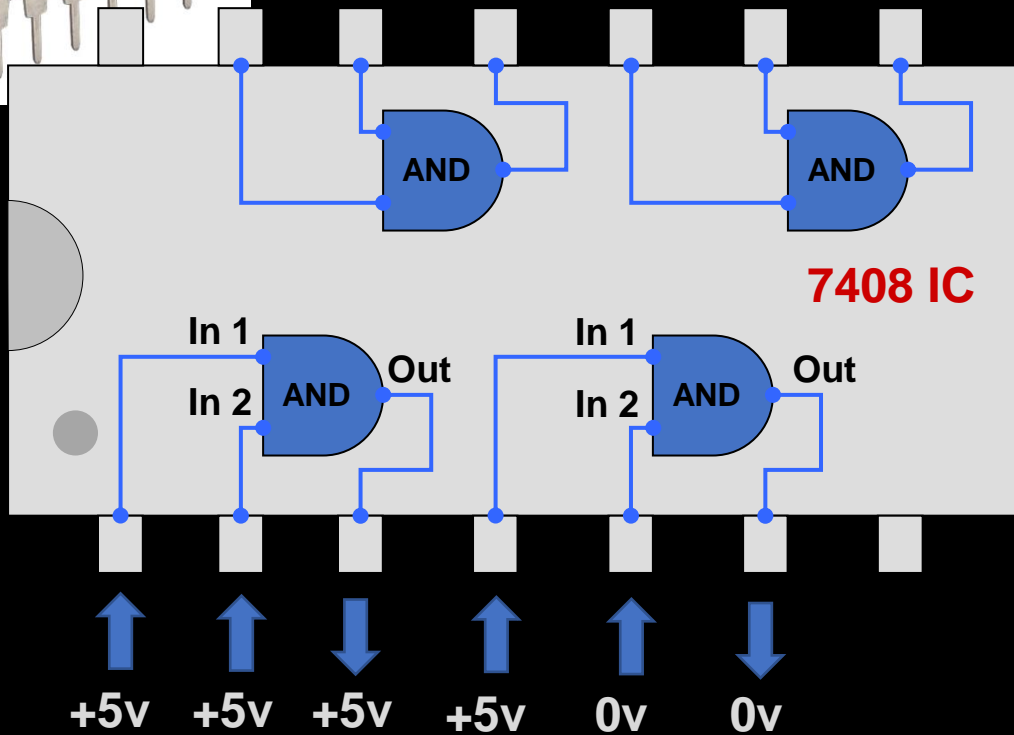
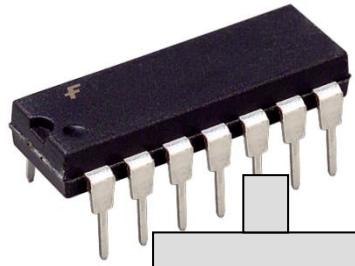
컴퓨터와 2진법



게이트 회로

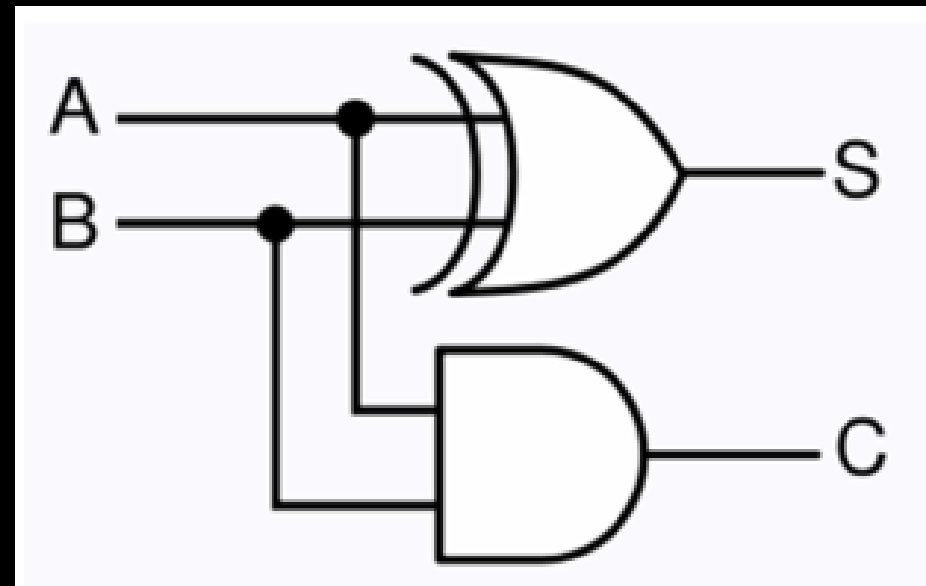
A	B	Out S			
					
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	
1	1	1	1	0	

디지털 회로

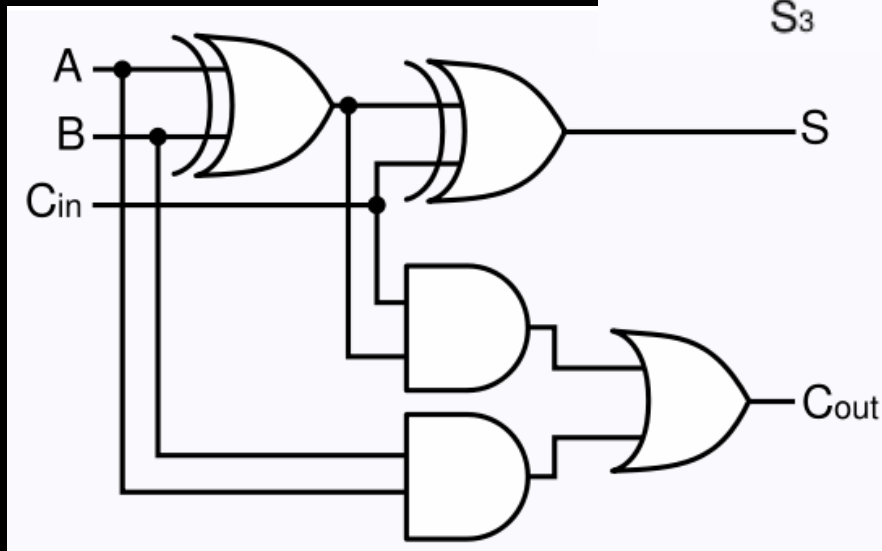
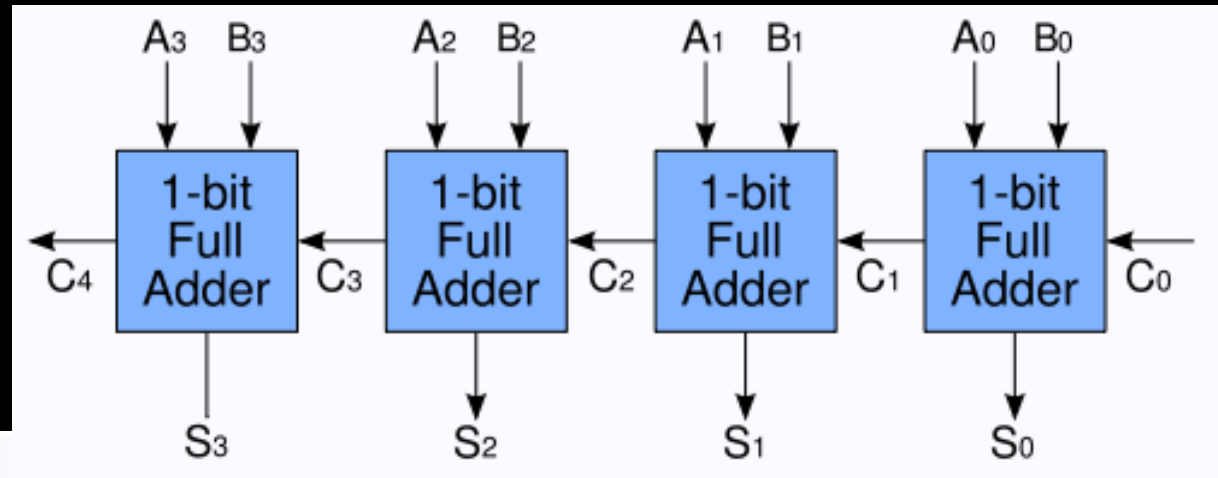


컴퓨터가 덧셈 하는 방법 #1

- 2진수 $1 + 1$ 은 2진수 $10_{(2)}$ 이다.
- A가 1, B가 1이면 XOR 연산결과 S는 0이다.
- 동시에 A가 1, B가 1이면 AND 연산결과 C는 1이다.
이 1은 자리 올림(Carry)이다.

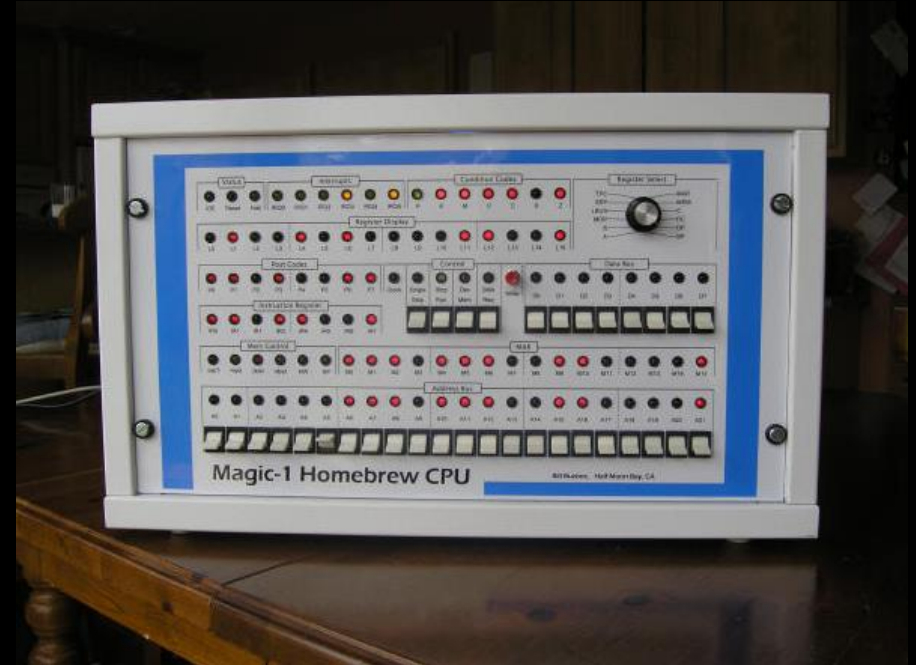
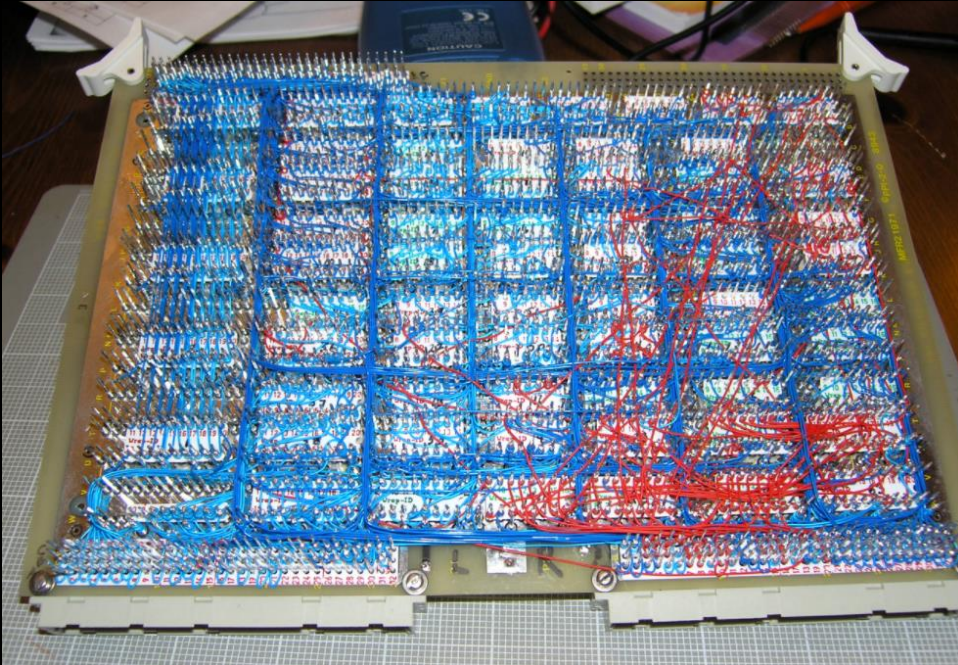


컴퓨터가 덧셈 하는 방법 #2



결론!
두 수를 더할 수 있으면
CPU를 만들 수 있다.

수제 CPU



<http://www.homebrewcpu.com/>

2. 진법 변환

10진법

- 인간이 사용하는 일반적인 진법체계
- 한 자리 숫자는 10가지 경우 존재
(0~9, 가장 큰 수는 9)
- 자릿수가 올라갈 때마다 10^n 을 곱함

2진수, 16진수 변환

2진수(4비트)	16진수
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

2진수(4비트)	16진수
1010	A (10진수 10)
1011	B (10진수 11)
1100	C (10진수 12)
1101	D (10진수 13)
1110	E (10진수 14)
1111	F (10진수 15)

※ 4비트는 16진수 한 자리 숫자다.

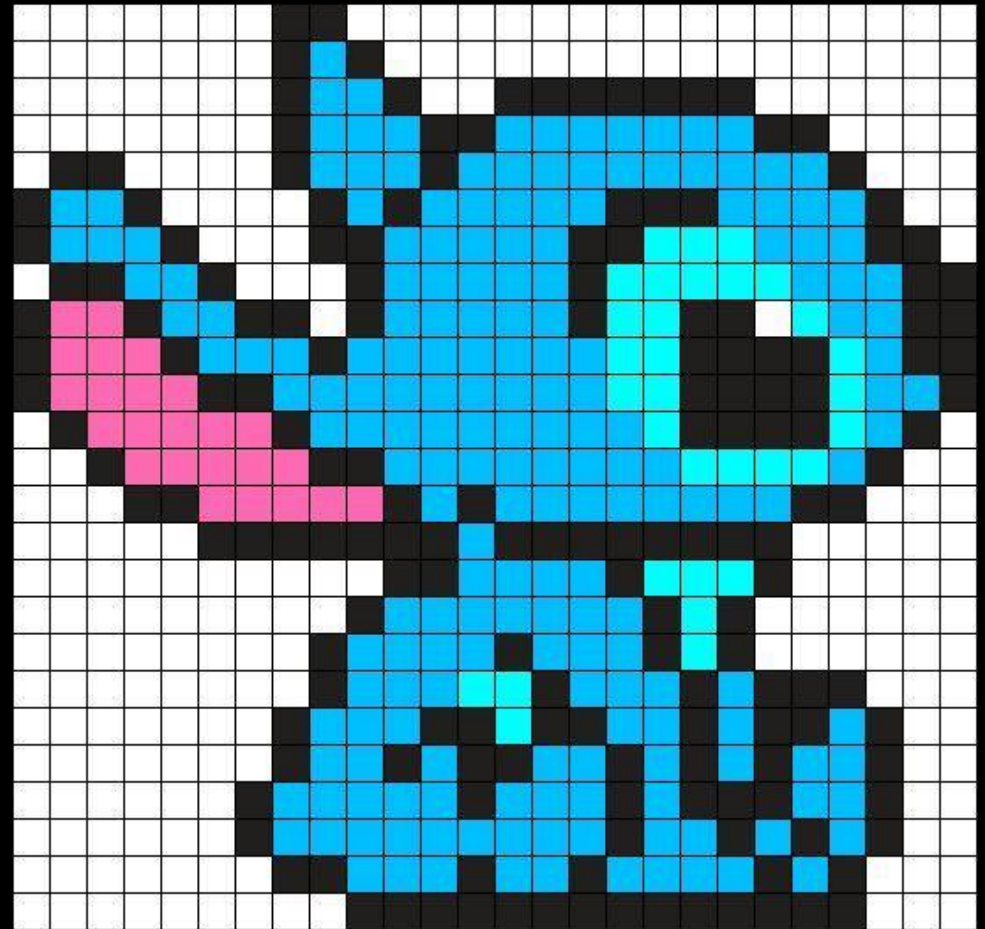
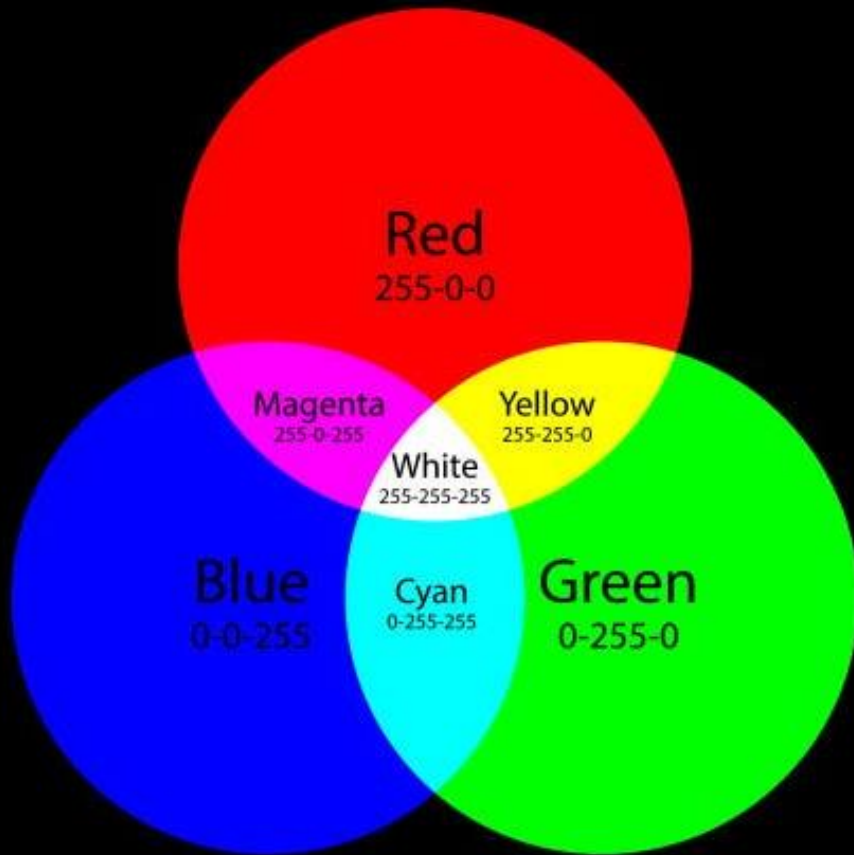
※ 16진수는 0~F(십진수 15) 까지 한 자리에 쓴다.

16진수(저수준) 표기가 사용되는 예

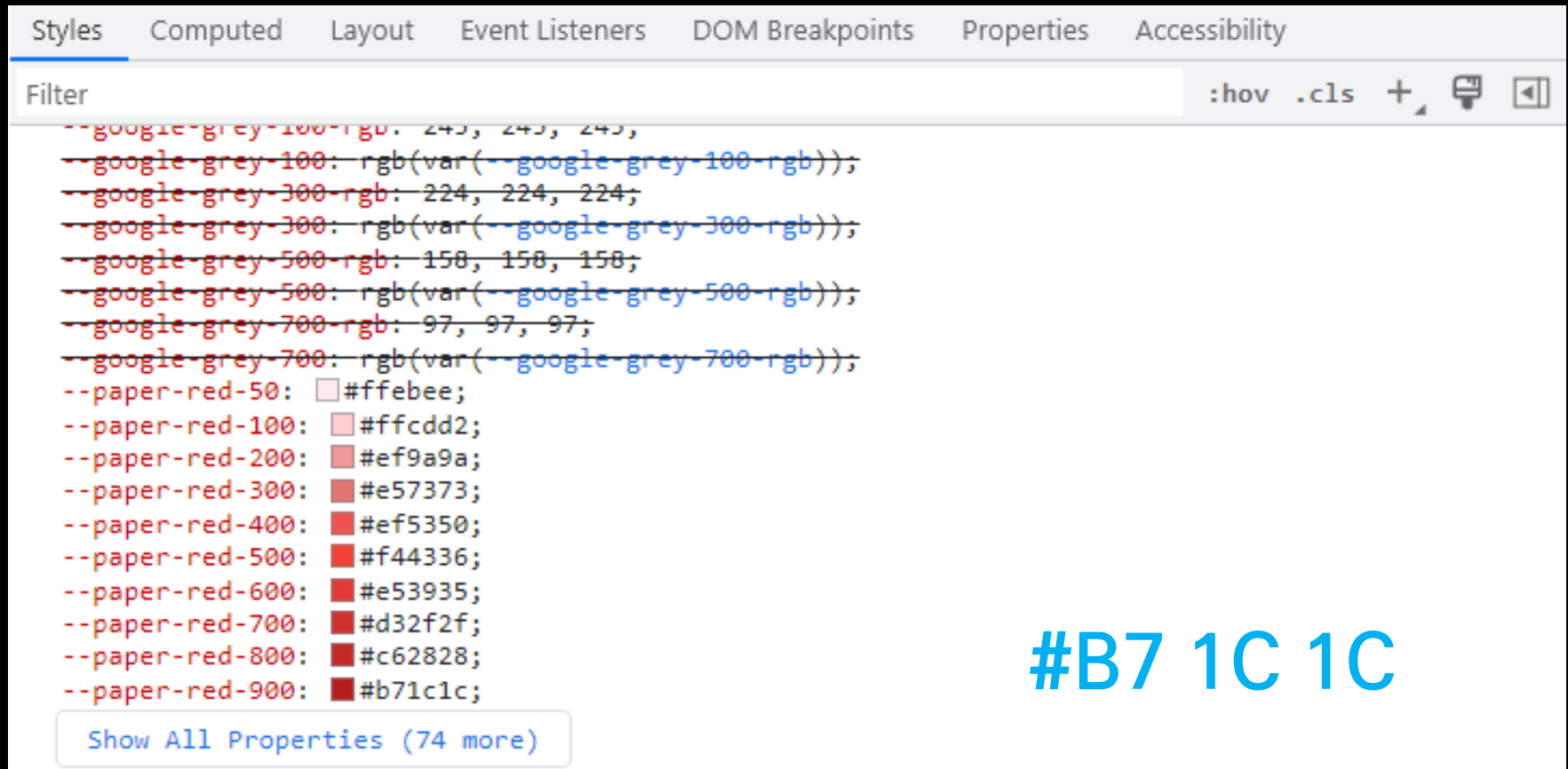
- 색상 표현 (RGB 컬러)
- 컴퓨터 하드웨어 주소 표현
- 메모리에 저장된 값 표현

RGB 색상 표현과 픽셀

Additive color mixing



저수준 정보 표현의 예 – RGB 컬러



저수준 정보 표현의 예 – 메모리 화면

```
메모리 1
주소: 0x00000281848B1070 열: 16
0x00000281848B1070 54 45 53 54 30 33 00 fe fe fe fe fe fe fe fe TEST03.????????
0x00000281848B1080 fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe ??????????????
0x00000281848B1090 fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe ??????????????
0x00000281848B10A0 fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe ??????????????
0x00000281848B10B0 e0 0f 8b 84 81 02 00 00 fd fd fd fd 41 00 70 00 ?.???...???A.p
0x00000281848B10C0 70 00 44 00 61 00 74 00 cc 01 3d b9 93 58 00 00 p.D.a.t.?.=??X.
0x00000281848B10D0 50 01 8a 84 81 02 00 00 40 40 8a 84 81 02 00 00 P.???...@@???..
0x00000281848B10E0 6f 00 63 00 6f 00 6c 00 61 00 74 00 65 00 79 00 o.c.o.l.a.t.e.y
0x00000281848B10F0 49 00 6e 00 73 00 74 00 61 00 6c 00 6c 00 3d 00 I.n.s.t.a.l.l.=
0x00000281848B1100 43 00 3a 00 5c 00 50 00 72 00 6f 00 67 00 72 00 C.:.\.P.r.o.g.r
0x00000281848B1110 61 00 6d 00 44 00 61 00 74 00 61 00 5c 00 63 00 a.m.D.a.t.a.\.c
0x00000281848B1120 68 00 6f 00 63 00 6f 00 6c 00 61 00 74 00 65 00 h.o.c.o.l.a.t.e
0x00000281848B1130 79 00 00 00 43 00 68 00 6f 00 63 00 6f 00 6c 00 y...C.h.o.c.o.l
0x00000281848B1140 61 00 74 00 65 00 79 00 4c 00 61 00 73 00 74 00 a.t.e.y.L.a.s.t
0x00000281848B1150 50 00 61 00 74 00 68 00 55 00 70 00 64 00 61 00 P.a.t.h.U.p.d.a
0x00000281848B1160 74 00 65 00 3d 00 31 00 33 00 32 00 38 00 31 00 t.e.=.1.3.2.8.1
0x00000281848B1170 36 00 37 00 34 00 37 00 31 00 38 00 32 00 35 00 6.7.4.7.1.8.2.5
0x00000281848B1180 32 00 39 00 35 00 32 00 37 00 00 00 43 00 4c 00 2.9.5.2.7...C.L
0x00000281848B1190 41 00 53 00 53 00 5f 00 50 00 41 00 54 00 48 00 A.S.S._.P.A.T.H
0x00000281848B11A0 3d 00 43 00 3a 00 5c 00 6a 00 61 00 76 00 61 00 =.C.:.\.j.a.v.a
0x00000281848B11B0 31 00 2e 00 38 00 5c 00 e1 02 00 e3 bb 58 00 00 1...8.\.?.???X.
```

외워야 할 단위 체계 #1

- 8개 비트를 하나로 묶어 1 바이트(byte)
- 1 바이트는 영문자 한 글자가 저장될 수 있는 메모리 크기이며 관리의 최소단위
(한 글 한 글자를 저장하려면 2 바이트가 필요)

외워야 할 단위 체계 #2

- 4비트는 16가지, 8비트는 256가지,
16비트는 65,536가지 (64KB)
- 2의 10 제곱은 1024

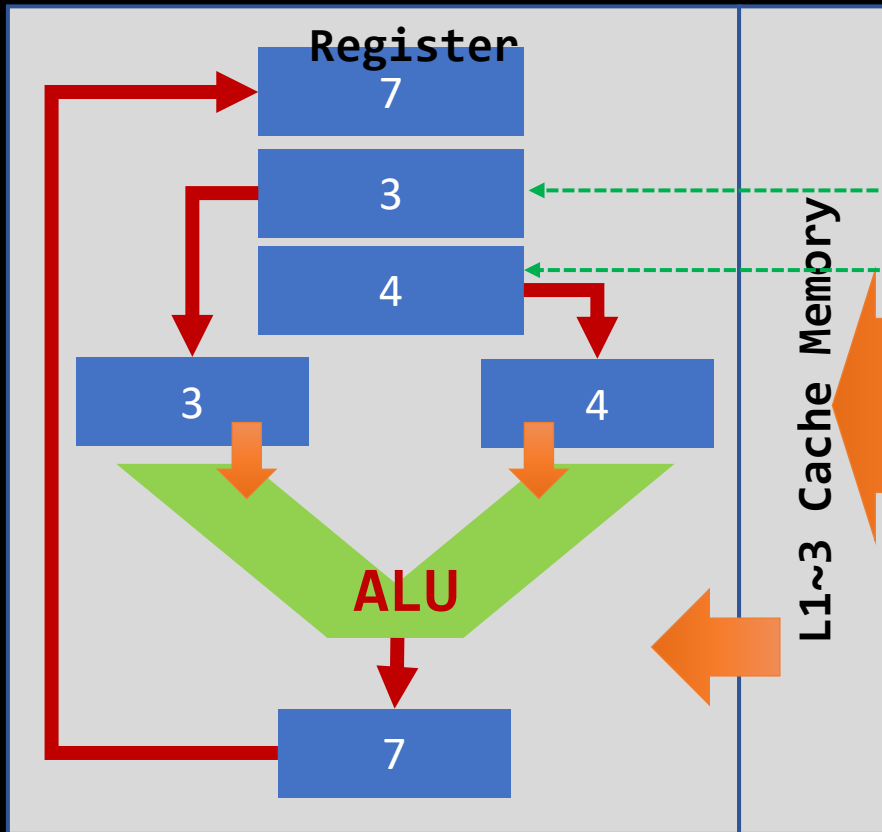
외워야 할 단위 체계 #3

- 2의 32 제곱(2^{32})은 4,294,967,296 (약 42.9억). 4,294,967,296 바이트는 4GB(기가바이트)
- 2의 32 제곱은 32 bit를 의미. 즉, 32비트 구조상 관리할 수 있는 한계 용량은 4GB

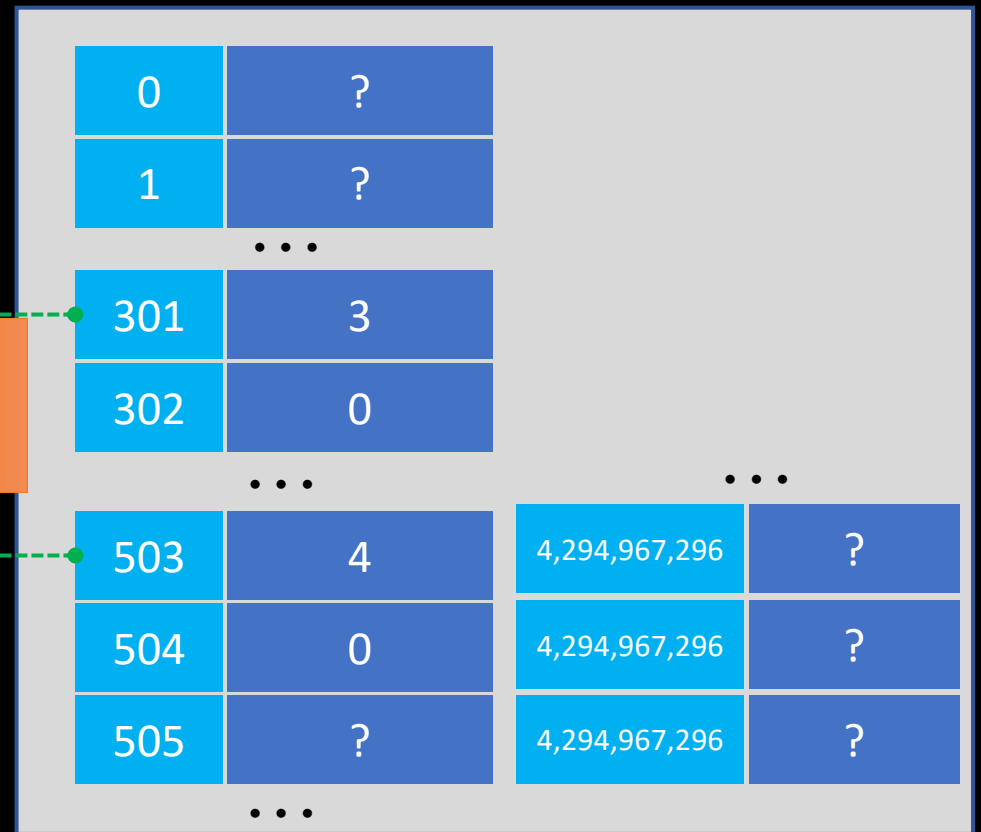
3. 컴퓨터 구조에 대한 이해

CPU와 메모리

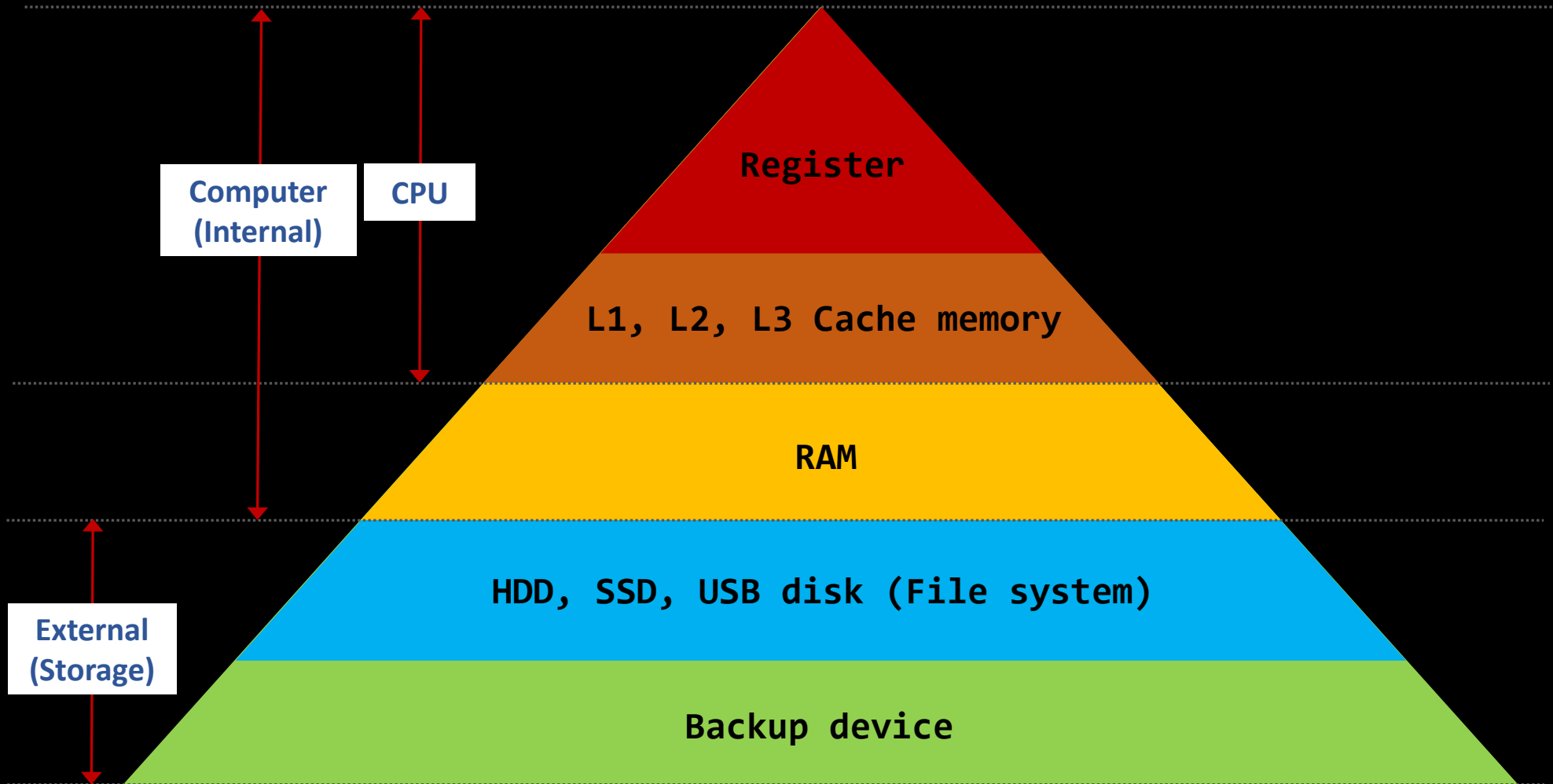
CPU



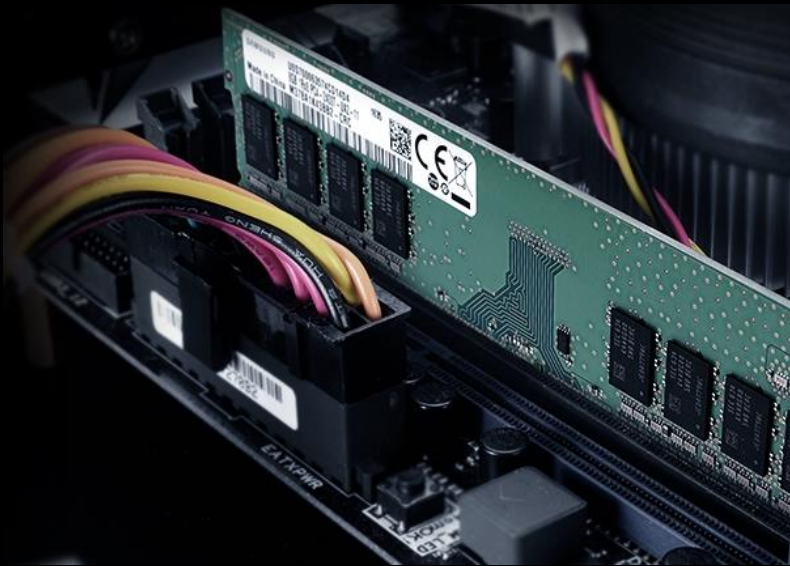
RAM



기억장치의 종류와 역할



주 기억장치 공간은 ‘일련번호’로 관리한다.



0	?
1	?

...

301	3
302	0

...

503	4
504	0
505	?

...



...

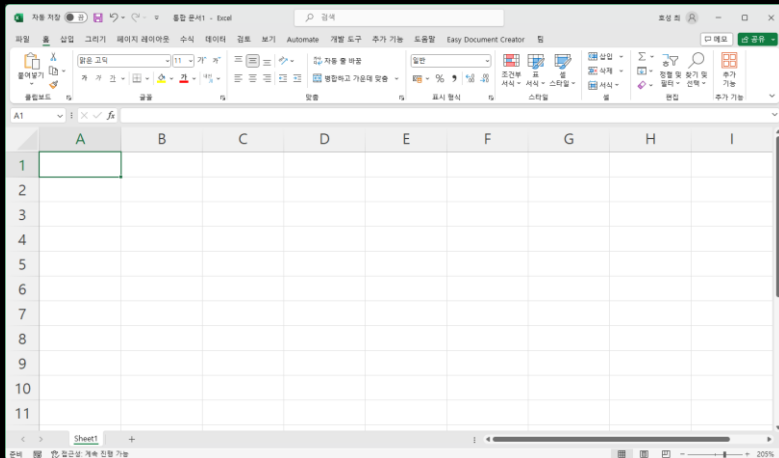
4,294,967,296	?
4,294,967,296	?
4,294,967,296	?

실행이란?

- 실행의 실체는 CPU 연산
- 연산에 필요한 정보는 메모리에서 가져와야 함
- 메모리에서 CPU 레지스터로 전달된 후 연산
- 연산 결과가 저장된 레지스터 값을 다시 메모리로 보내는 과정을 반복

참조에 대한 개념적 이해

- 참조의 대상이 먼저 존재
- 참조자는 대상체에 대한 정보 중 하나로 대상체에 접근 할 수 있는 근거자료
- Excel을 생각하면 쉽게 이해 할 수 있음



CPU의 다른 이름 Machine

User mode

Kernel mode

S/W

H/W

CPU

4. CPU 수준 자료형

자료형

- C언어에서 자료형(Data type)은 일정 길이의 메모리에 저장된 정보를 해석하는 방법
- 자료는 결국 숫자
- C언어의 변수는 메모리를 사용하기 위한 문법으로 이해 할 수 있음
- 자료는 변수와 상수 두 종류가 있음

C언어 자료형

- 정수형 (부호유무, 크기)
- 실수형 (크기)
- 유도형 (*, [], 구조체, 공용체)
- 함수형
- 무치형 (void)

상수 정의 및 종류

- 값이 확정되어 앞으로 변할 가능성이 없는 수
- 리터럴 상수와 심볼릭 상수가 있음
(*심볼릭 상수는 별도로 다룸)

리터럴 상수

- 문자 상수 (ASCII 코드)
 - 'A'
- 문자열 상수
 - "Hello, World"
- 정수 상수
 - 3, 4L
- 실수 상수
 - 3.4F, 123.45

변수의 정의

- 구체화하지 않았거나 앞으로 변경될 가능성이 있는 수 (혹은 미지의 수)
- 메모리를 사용하는 가장 일반적인 방법
- 변수는 메모리가 가지는 특성(위치정보인 주소, 공간의 크기)을 가짐

정수형

자료형	크기 (byte)	설명
char	1	부호비트(1비트)와 나머지 전체를 데이터 비트로 해석
short	2	
int	4	
long	4	
long long int	8	
Unsigned 조합	-	부호비트도 데이터로 해석

정수형 표현 범위

자료형	설명
char	$-128 \sim 127$ ($-2^7 \sim 2^7-1$)
unsigned char	$0 \sim 255$ ($0 \sim 2^8-1$)
short	$-32768 \sim 32767$ ($-2^{15} \sim 2^{15}-1$)
unsigned short	$0 \sim 65535$ ($0 \sim 2^{16}-1$)
int	$-2147483648 \sim 2147483647$ ($-2^{31} \sim 2^{31}-1$)
unsigned int	$0 \sim 4294967295$ ($0 \sim 2^{32}-1$)

정수형

```
#include <stdio.h>

int main(void)
{
    char ch = 'A';
    wchar_t wch = L'A';
    short sData = 10;
    int nData = 10;
    long lData = 10L;
    long int lnData = 10L;
    long long int llnData = 10LL;

    // 부호비트가 없는 자료형
    unsigned char byNewData = 0;
    unsigned short sNewData = 65535U;
    unsigned long int ulNewData = 1UL;
    unsigned long long int ullNewData = 1ULL;

    return 0;
}
```

컴퓨터가 뺄셈 하는 방법#1

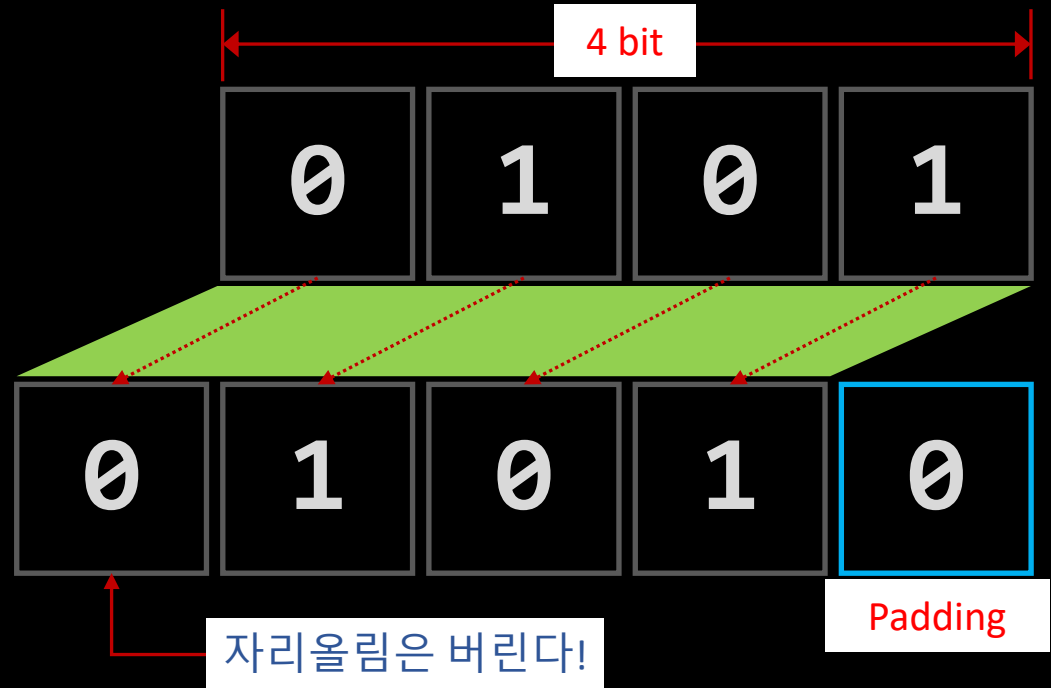
- 6에 4를 더하면 10이다. 즉, 4는 6에 대한 10의 보수
- $13 - 6$ 은 7
- 13에 6에 대한 10의 보수 4를 더하고 10자리에서 1을 빼도 역시 7

컴퓨터가 뱌셈 하는 방법#2

- 2진수에서 0은 1로, 1은 0으로 뒤집으면 1의 보수
- 1의 보수에 1을 더하면 2의 보수
- 어떤 숫자에 2의 보수를 더하면 자동으로 2진수 뱌셈
(단, 자리올림은 절사)

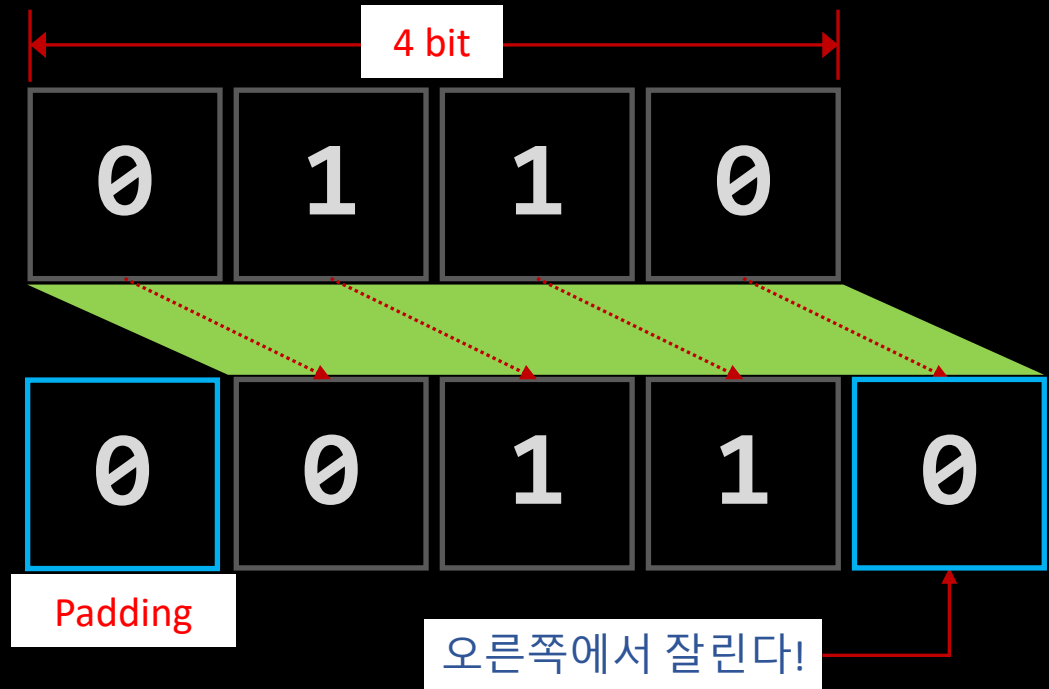
컴퓨터가 곱셈 하는 방법

- 4비트로 5를 표현하면 0101이다.
- 4비트 0101을 왼쪽으로 한 칸 씩 밀면(Shift) 1010이다.
- 맨 오른쪽에 0이 채워진다. (Padding)
- 4비트로 표현하는 2진수 1010은 10이다.
- 왼쪽으로 한 칸 밀면 곱하기 2, 두 칸 밀면 곱하기 4가 된다.



컴퓨터가 나눗셈 하는 방법

- 4비트로 6를 표현하면 0110이다.
- 4비트 0110을 오른쪽으로 한 칸씩 밀면 0011이다.
- 맨 오른쪽에 0이 채워진다. (Padding)
- 4비트로 표현하는 2진수 0011은 3이다.



컴퓨터가 나눗셈 하는 방법 #2

- 7을 0을 나누면?
- 7에서 0을 빼면 7이고 7은 0보다 크다.
- 7에서 0을 계속 빼면 언젠가는 0보다 작은 숫자를 만날 수 있는가?
- 만날 수 없다면 뺄셈 연산은 언제 끝날까?



<https://www.youtube.com/watch?v=mZ7pUADoo58>

실수형

- 소수점 이하 정보를 표시할 수 형식
- 부동 소수점 표현
- 100.0 , $10.0 * 10$, $1.0 * 10^2$ 은 같은 값에 대한 표현
- 두 정수 사이에는 무수히 많은 실수가 존재하기 때문에 일정 수준이 오류(부동소수점 오차)를 인정함

실수형

- **IEEE**(Institute of Electrical and Electronics Engineers, 전기전자 기술자협회)가 규정한 표준사용
- IEEE는 **ANSI**(American Standard National Institute, 미국표준협회)가 인증한 단체
- **IEEE 754** 표준

실수형

자료형	크기 (byte)	설명
float	4	부호비트, 지수, 가수 등 세 부분으로 구성되며 지수값과 가수값을 연산해 값을 표현
double	8	
long double	8 이상	

실수형 표현 범위

자료형	설명
float	$1.17 * 10^{-38} \sim 3.4 * 10^{38}$ (유효 자릿수: 소수점 이하 6자리)
double	$2.22 * 10^{-308} \sim 1.79 * 10^{308}$ (유효 자릿수: 소수점 이하 15자리)
long double	double 이상

실수형 표현 범위

IEEE 754 single format

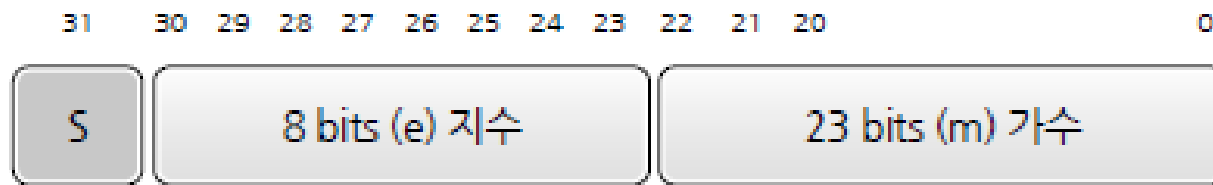


그림 2-8 IEEE 754 단정도형의 구조

IEEE 754 double format

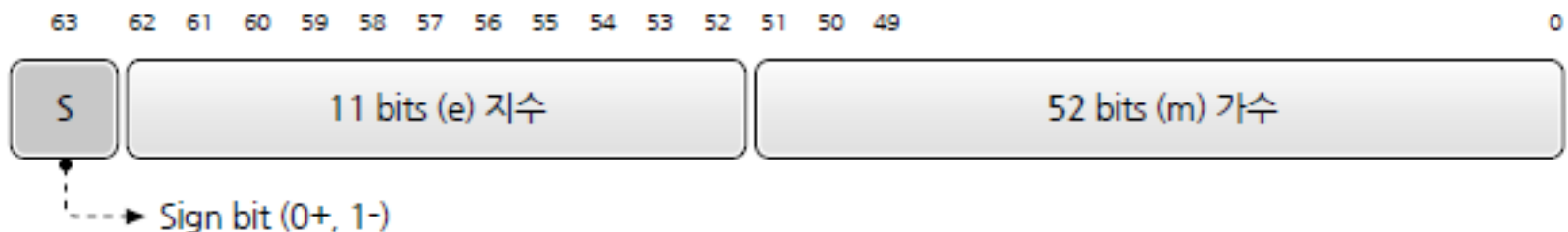


그림 2-9 IEEE 754 배정도형의 구조

실수형 표현 범위

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

int main(void)
{
    double dData = 123.456;

    printf("%f\n", dData);
    printf("%E - %E\n", DBL_MIN, DBL_MAX);
    return 0;
}
```

실수형 - 부동산수점 오차

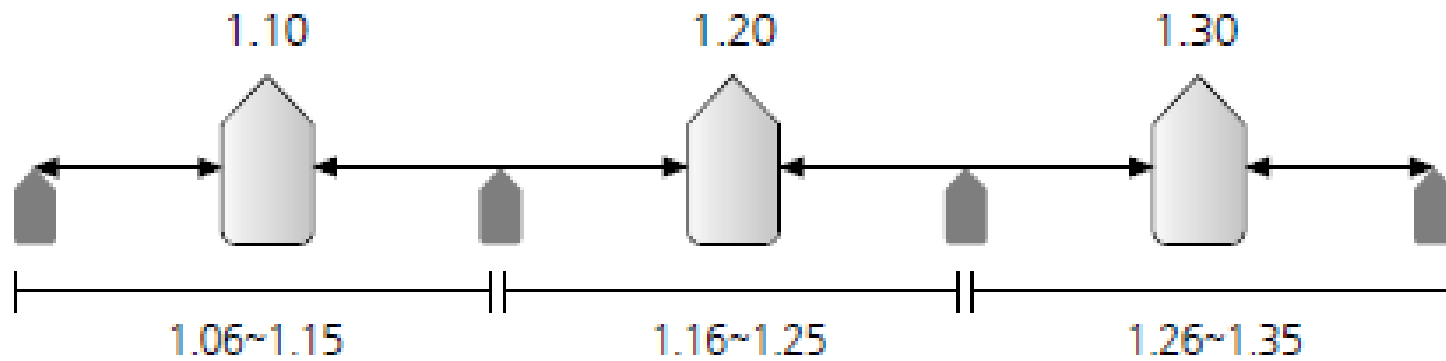


그림 2-7 일정 구간의 실수를 대표하는 실수 표현

실수형 - 부동소수점 오차

```
printf("%f\n",  
    0.1F + 0.1F + 0.1F + 0.1F + 0.1F +  
    0.1F + 0.1F + 0.1F + 0.1F + 0.1F +  
    0.1F + 0.1F + 0.1F + 0.1F + 0.1F +  
    0.1F + 0.1F + 0.1F + 0.1F + 0.1F +  
    0.1F + 0.1F + 0.1F + 0.1F + 0.1F +  
    0.1F + 0.1F + 0.1F + 0.1F + 0.1F +  
    0.1F + 0.1F + 0.1F + 0.1F + 0.1F +  
    0.1F + 0.1F + 0.1F + 0.1F + 0.1F +  
    0.1F + 0.1F + 0.1F + 0.1F + 0.1F  
);
```

4.999998

C:\Users\cx853\Desktop\nullnull_C_20230918\nullnu
ll_C\x64\Debug\09_floatError.exe(프로세스 44440개
)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

5. 코드 체계

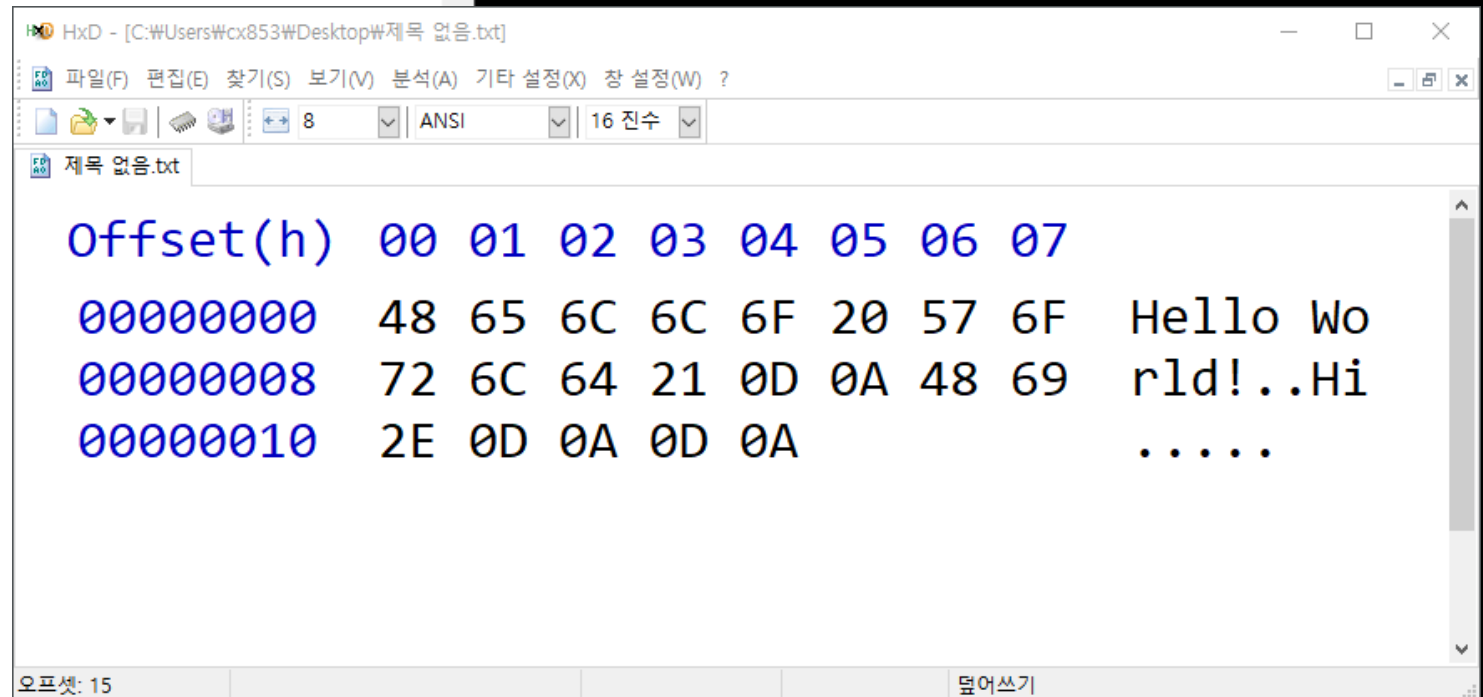
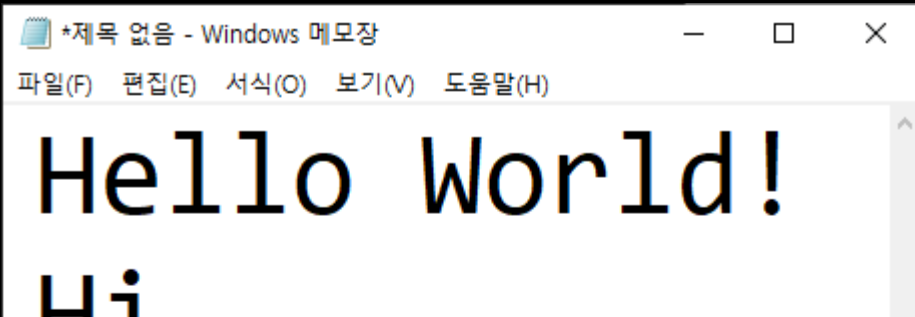
컴퓨터가 글자를 다루는 방법 - ASCII

- 십진수 65
- 컴퓨터에겐 영문 대문자 'A'
- 16진수로는 0x41

컴퓨터가 글자를 다루는 방법 - ASCII

- ASCII(American Standard Code for Information Interchange)는 미국에서 사용하는 표준 코드체계
- 숫자와 글자를 구별하지 않고 정보를 말할 때는 바이너리(Binary)

문자와 문자열의 실체



문자와 문자열

```
#include <stdio.h>

int main(void)
{
    char c1 = 'A', ch2 = 'B', ch3 = 'C';
    char szData[4] = { 'A', 'B', 'C' };
    char szNewData[4] = { "ABC" };
    printf("%s\n", szData);
    printf("%s\n", szNewData);
    // Array에 명시된 값이 없으면 0으로 초기화
    printf("%d\n", szData[3]);
    return 0;
}
```

Unicode

- C언어의 문자열은 **MBCS**(Multi-bytes Character Sets)과 **Unicode** 문자열로 나눔
- 유니코드는 한글 영문 모두 한 글자가 **2바이트**

프로그래밍을 배우려는 사람들을 위해...

**문자를 다루는 인코딩 규칙에
대한 모든 것! (※매우 중요)**

**UTF-8, UTF-8 BOM, UTF-16,
Unicode, 한글 완성형/조합형**

6. 프로그래밍 언어 구조

CPU의 다른 이름 Machine

User mode

Kernel mode

S/W

H/W

CPU

기계어 코드 예

메모리 1									
주소: 0x00007FF65C4F1AC0				↺		열: 8		▼	
0x00007FF65C4F1AC0	48	89	4c	24	08	55	57	48	H?L\$.UWH
0x00007FF65C4F1AC8	81	ec	08	01	00	00	48	8d	??....H?
0x00007FF65C4F1AD0	6c	24	20	48	8d	0d	36	05	l\$ H?.6.
0x00007FF65C4F1AD8	01	00	e8	b4	f8	ff	ff	b9	..???..?
0x00007FF65C4F1AE0	48	00	00	00	ff	15	1e	f8	H.....?
0x00007FF65C4F1AE8	00	00	48	89	45	08	41	b8	..H?E.A?
0x00007FF65C4F1AF0	48	00	00	00	33	d2	48	8b	H...3?H?
0x00007FF65C4F1AF8	4d	08	e8	e0	f7	ff	ff	48	M.???..H
0x00007FF65C4F1B00	8b	45	08	4c	8b	85	00	01	?E.L??..
0x00007FF65C4F1B08	00	00	ba	40	00	00	00	48	..?@...H
0x00007FF65C4F1B10	8b	c8	ff	15	08	f8	00	00	??....?..

디스어셈블 코드

메모리 1

주소: 0x00007FF65C4F1AC0

열: 8

0x00007FF65C4F1AC0 48 89 4c 24 08 55 57 48 H?L\$.UWH

0x00007FF65C4F1AC8 81 ec 08 01 00 00 48 8d ??....H?

0x00007FF65C4F1AD0 6c 24 20 48 8d 0d 36 05 1\$ H?.6.

0x00007FF65C4F1AD8 01 00 e8 b4 f8 ff ff b9 ..???...?

0x00007FF65C4F1AE0 48 00 00 00 ff 15 1e f8 H.....?

0x00007FF65C4F1AE8 00 00 48 89 45 08 41 b8 ..H?E.A?

0x00007FF65C4F1AF0 48 00 00 00 33 d2 48 8b H...3?H?

0x00007FF65C4F1AF8 4d 08 e8 e0 f7 ff ff 48 M.???..H

0x00007FF65C4F1B00 mov qword ptr [rsp+8],rcx

0x00007FF65C4F1B04 push rbp

0x00007FF65C4F1B08 push rdi

sub rsp,108h

lea rbp,[rsp+20h]

lea rcx,[__AA290256_SingleList@c (07FF65C502010h)]

call __CheckForDebuggerJustMyCode (07FF65C4F1393h)

C언어 코드

```
57 int InsertAtTail(char* pszData)
58 {
59     NODE* pNode = (NODE*)malloc(sizeof(NODE));
60     memset(pNode, 0, sizeof(NODE));
61     strcpy_s(pNode->szData, sizeof(pNode->szData), pszData);
62
63     if (IsEmpty())
64         //리스트에 추가된 첫 번째 데이터 처리
65         g_head.next = pNode;
66     else
67         g_pTail->next = pNode;
68
69     g_pTail = pNode;
70 }
```

컴파일러(Compiler)

- 고급어 소스코드를 기계어로 번역하는 프로그램
- 전체 소스코드를 모두 기계어로 변환한 후 실행
- 성능 최적화가 용이하여 다수 언어가 채택
- C, C++등이 여기에 해당

인터프리터(Interpreter)

- 고급어 **소스코드**를 직접 실행하는 프로그램이나 환경을 의미
- 보통 한번에 **한 줄 단위로 실행**
- 성능(특히 속도)면에서 컴파일러 방식보다 느림
- **JavaScript나 Python**등이 여기에 해당

Part 2

C언어 프로그래밍 시작


7. 개발환경 구축

Windows 11기반 개발환경 구축

- Windows 11
- 64bit CPU
- Visual Studio 2022 Community
- Excel

Visual Studio 2022 환경설정

설정 가져오기 및 내보내기 마법사



설정 가져오기 및 내보내기 마법사 시작

이 마법사를 사용하여 특정 설정 범주를 가져오거나 내보낼 수 있습니다. 다시 설정할 수도 있습니다.


원하는 작업을 선택하세요.

- ☐ 선택한 환경 설정 내보내기(E)
설정이 파일로 저장되므로 나중에 언제든지 어떤 컴퓨터로도 사용할 수 있습니다.
- ☐ 선택한 환경 설정 가져오기(I)
파일에서 환경에 적용할 설정을 가져옵니다.
- ☒ 모두 다시 설정(R)
모든 환경 설정을 기본 설정 모음 중 하나로 다시 설정합니다.

< 이전(P)

다음(N) >

설정 가져오기 및 내보내기 마법사



기본 설정 모음 선택

어느 설정 모음으로 다시 설정하시겠습니까(W)?

- JavaScript
- Visual Basic
- Visual C#
- Visual C++**
- 웹 개발
- 웹 개발(코드만)
- 일반

설명:
네이티브 및 관리 C++ 애플리케이션을 개발하는 데 필요한 도구를 갖추도록 환경을 설정합니다. 이 설정 모음에는 Visual C++ 6 스타일의 바로 가기 키와 기타 사용자 지정이 포함됩니다.

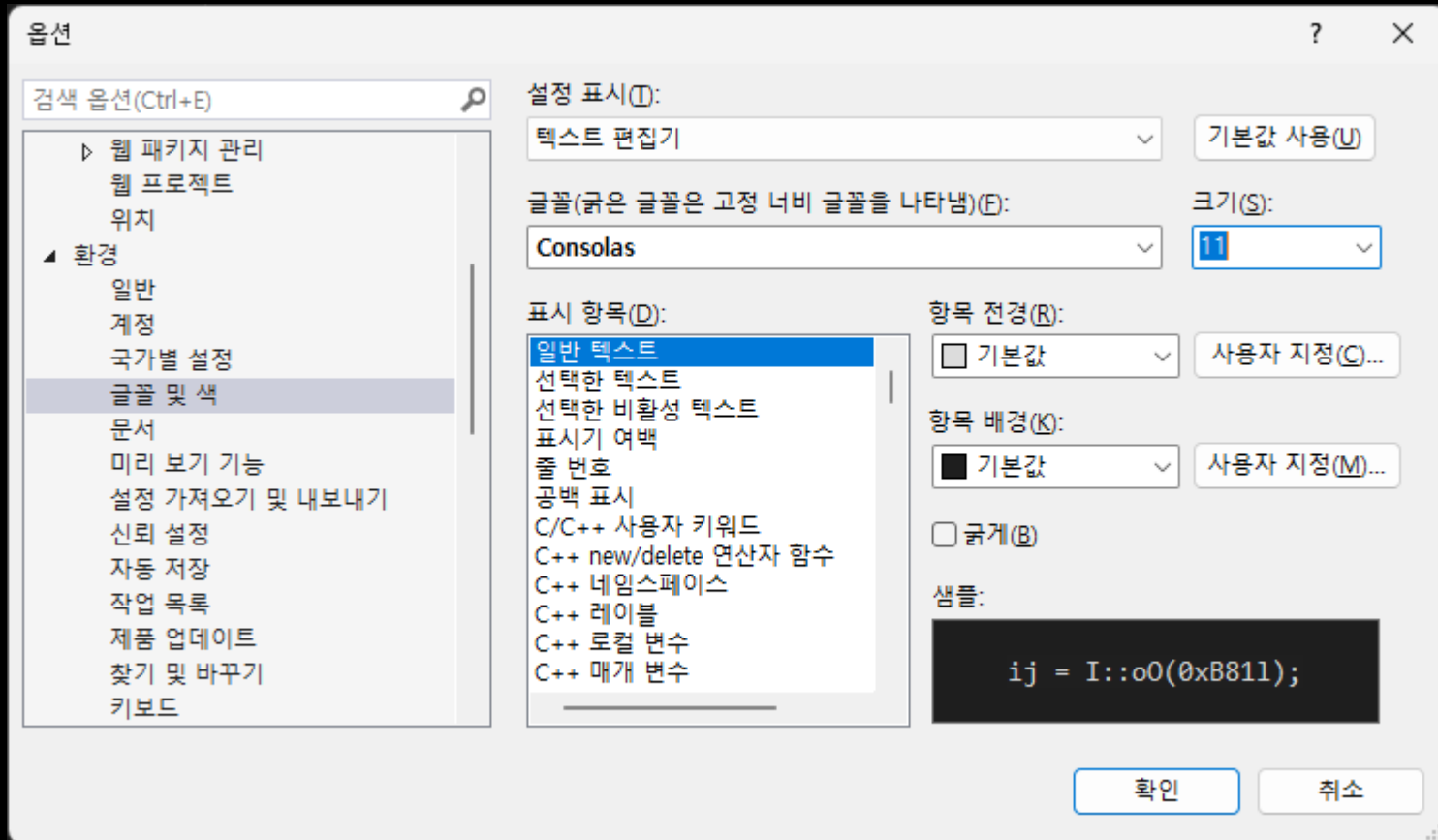
< 이전(P)

다음(N) >

마침(F)

취소

Visual Studio 2022 환경설정



Visual Studio 2022 핵심 단축키

- F5 – 디버그 모드 실행
- Shift + F5 디버그 모드 강제 중단
- F7 – 프로젝트 빌드
- F9 – 중단점 설정
- F10 – 디버그 모드에서 한 행 실행
- F11 – 디버그 모드에서 함수 추적
- Ctrl + F5 – 실행

Hello, World!

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

```
int main(void)  
{  
    printf("Hello, World\n");  
    return 0;  
}
```


중요한 세 가지 시점

- 컴파일 타임

- .c 소스코드를 목적 파일 .obj로 번역

- 링크 타임

- 목적 파일들과 라이브러리를 실행 파일로 합성

- 빌드 타임 (컴파일 + 링크 타임)

- 런타임 (실행)

소스코드와 목적파일

- C언어 소스코드 파일은 확장명이 .c
- 목적파일은 확장명이 .obj
- 외부 라이브러리 파일은 .lib (.obj와 유사)
- .obj와 .lib를 합성해 실행파일 .exe 생성

솔루션 파일과 프로젝트

- VS는 여러 프로젝트를 한 **솔루션**으로 묶어서 관리 할 수 있음
- 한 **프로젝트**는 여러 C언어 소스 코드 파일을 하나로 묶어 관리 할 수 있음
- 한 솔루션에 여러 프로젝트 파일이 존재할 경우 '**시작 프로젝트**'로 설정된 프로젝트에 대해 빌드/디버그 단축키 적용

8. C언어 기초 문법

사용 가능한 문자

- ASCII 코드 기준 문자 중 일부
- a~z, A~Z, 0~9
- 특수 문자
- White space (공백 문자)
- Escape sequence

예약어

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	While			
_Bool	_Complex	_Imaginary	Inline	

변수 이름(식별자, Identifier) 작성 규칙

- 영문 대/소문자, '_', 숫자 가능
 - int nData, int _Data, int _nData1
 - int \$nData, int #nData (X)
- 첫 글자는 숫자 사용 불가
 - int 1Data (X)
- 이름 중간에 공백 문자 사용 불가
 - int data length (X)

변수 이름(식별자, Identifier) 작성 규칙

- 예약어 사용 불가

- int for (X)

- 너무 긴 이름 금지 (권장사항)

- int multibytestringtowidestring;

- 의미를 알 수 없는 이름

- int aaa, bbb, ccc, ddd;

항, 식, 구문, 실행

- 문자(피연산자, 변수나 상수)로 항 기술
 - 소괄호를 이용해 여러 항을 한 항으로 묶음
- 항과 연산자가 모여 식 완성
- 식에 대한 평가
- 식이 모여 구문 완성
 - 여러 구문을 중괄호로 묶을 수 있음
- 여러 줄의 구문을 연속 실행 (함수)

코드와 주석

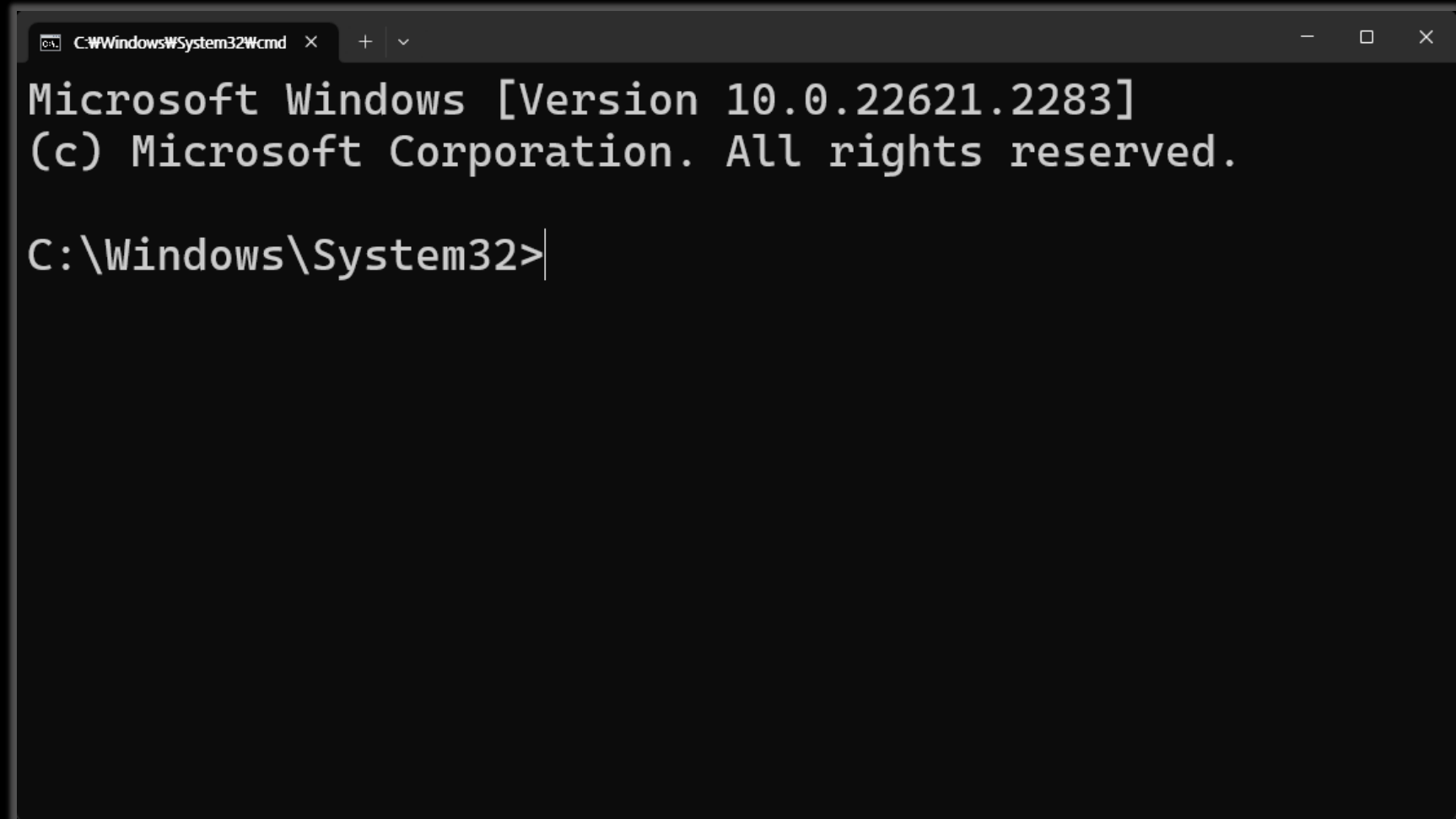
- 프로그램 코드에 **메모**를 남기는 문법
- 프로그램 코드에 포함되지 않음
- `//` (한 행 전체)
- `/* */` (구간 전체, 여러 행 가능)
- **Ctrl + K + C** (주석 지정)
- **Ctrl + K + U** (주석 해제)

9. 표준 입/출력

Console과 CLI

- CLI(Command Line Interface) 기반 HCI(Human Computer Interface)는 키보드 입력으로 구현
- 키보드 입력 시 그 값은 메모리(I/O Buffer)에 연속적으로 저장
- I/O Buffer에서 한 글자 단위로 처리

Console



```
C:\Windows\System32\cmd
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>
```

Console



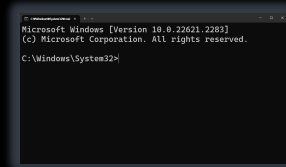
Console

User mode

Kernel mode

S/W

H/W



문자 입/출력

- `getchar()` / `putchar()`
 - Buffered I/O
 - 값이 저장된 메모리 값을 읽거나 출력
- `_getch()` / `_getche()`
 - Non-buffered I/O
 - 키보드 입력 자체에 대한 감지

문자열

- 문자열은 문자 배열의 줄임말
- 각 문자들이 메모리에 연속적으로 저장되어 있으며 통상 첫 글자가 저장된 메모리의 주소로 관리
- 문자열의 끝은 ' \0' (null)

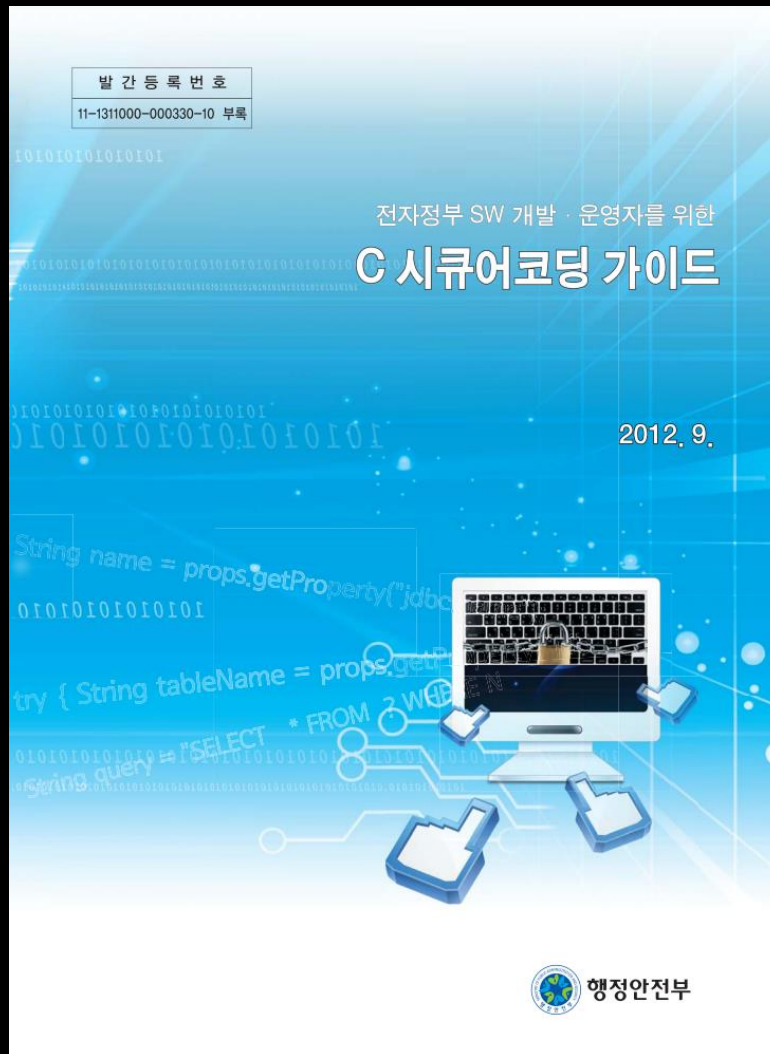
문자열 입/출력

- `gets()` / `puts()`
- `gets_s()`
- `printf()` / `scanf()`, `scanf_s()`

gets() 함수의 보안 결함

- 매개변수로 메모리의 주소를 받지만 얼마나 써도 되는지 크기를 확인하지 않아 발생
- 메모리 경계를 벗어난 쓰기를 수행
- 보안 문제가 발생하지 않도록 코드 수준에서 대응 하는 것이 중요
(시큐어 코딩)

gets() 함수의 보안 결함



형식 문자

형식문자	자료형	형식문자	자료형
%c	int, char	%d	Int
%o	int	%u	Unsigned
%x, %X		%e, %E	
%f	double	%g	double
%s	String	%zd	size_t
%lld		%l64u	

이스케이프 시퀀스 (Escape sequence)

문자	의미	문자	의미
<code>\a</code>	경고음	<code>\\</code>	Backslash 문자 자체
<code>\b</code>	Backspace	<code>\?</code>	
<code>\f</code>	Form feed	<code>\'</code>	
<code>\n</code>	New line	<code>\"</code>	
<code>\r</code>	Carriage return	<code>\ooo</code>	8진수
<code>\t</code>	Tab	<code>\xhh</code>	16진수
<code>\v</code>	Vertical tab		

이스케이프 시퀀스

```
int x = 10;

// 문자상수를 화면에 출력한다.
putchar('B');
// '\n'은 개행문자이므로 알파벳 문자가 화면에 보이는 것이 아니다.
putchar('\n');

printf("%c\n", 'A');

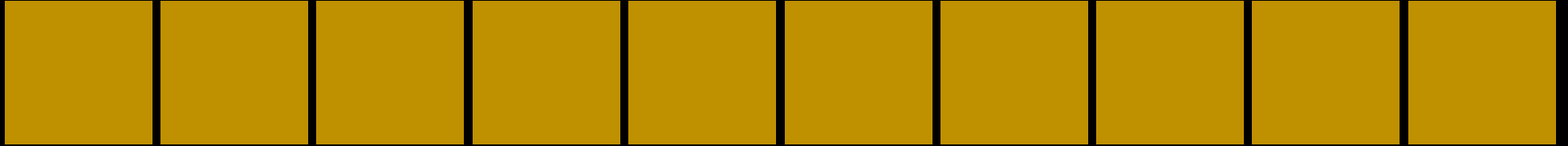
// %d라는 형식 문자에 맞춰 변수 x에 담긴 정보를 출력한다.
printf("x는 %d 입니다.", x);
```

printf() – 문자와 정수 출력

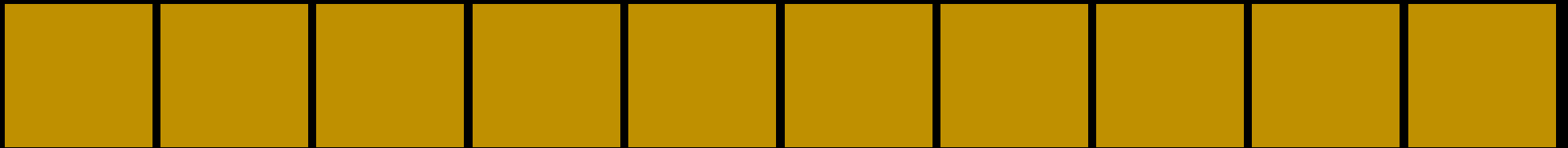
```
//문자상수를 ASCII코드 문자형식으로 출력한다.  
printf("%c\n", 'A');  
//문자 'A'의 ASCII 코드값(65)에 1을 더한 값을 문자로 출력한다.  
printf("%c\n", 'A' + 1);  
//문자상수를 ASCII코드 문자형식으로 출력한다.  
printf("%c\n", 'C');  
  
//문자 'A'의 ASCII 코드값을 10진수(%d)로 출력한다.  
printf("%d\n", 'A');  
//문자 'A'의 ASCII 코드값에 1을 더하고 10진수(%d)로 출력한다.  
printf("%d\n", 'A' + 1);  
//문자 'C'의 ASCII 코드값을 10진수(%d)로 출력한다.  
printf("%d\n", 'C');  
  
//10진수(정수)를 문자(ASCII)로 출력한다.  
printf("%c\n", 65);  
printf("%c\n", 65 + 1);  
printf("%c\n", 67);  
return 0;
```


scanf()와 입력 버퍼 – 문자 입력

`"%c"`

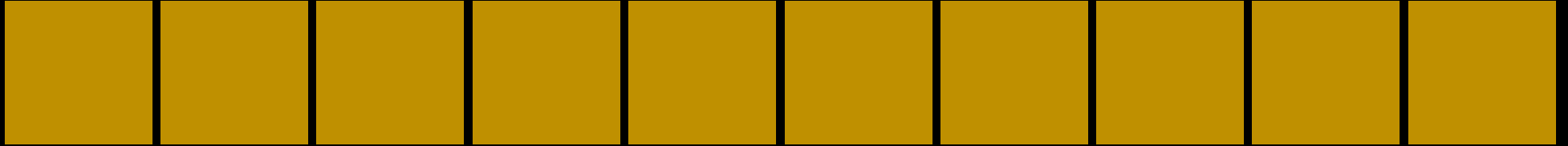


`"%c%*c"`

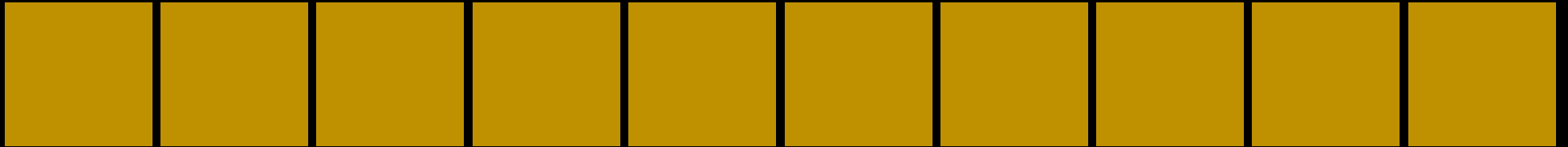


scanf()와 입력 버퍼 – 정수 입력

"%d%c"



"%d%d"



필수 실습 문제

난이도 2, 제한시간 20분

사용자로부터 이름과 나이를 키보드로 입력 받아 출력하는 프로그램을 작성.

이름은 `gets_s()`, 나이는 `scanf_s()` 함수로 입력 받고 `printf()` 함수로 출력

나이를 입력하세요: 20

이름을 입력하세요: 철수

당신의 나이는 20살이고 이름은 '철수' 입니다.

10. 연산자

연산자

- 연산자는 CPU 연산과 직결되는 문법
- 연산자 자체는 하나의 항
- 여러 항을 모아 연산식 작성
- 연산자와 피연산자로 구성
- 피연산자가 2개 항이면 2항 연산자
(하나는 단항, 셋이면 3항 연산자)

연산자 종류

- 산술 연산자
- 대입 연산자
- 형변환 연산자
- 단항 증/감 연산자
- 비트 연산자
- 관계, 논리 연산자, 조건 연산자
- sizeof 연산자

연산자 우선 순위

우선 순위	연산자	결합성	우선 순위	연산자	결합성
1	() [] . ->	L -> R	9	^	L -> R
2	* & ! ++ -- (type) sizeof -	L <- R	10		L -> R
3	* % /	L -> R	11	&&	L -> R
4	+ -	L -> R	12		L -> R
5	<< >>	L -> R	13	? :	L <- R
6	< > <= >==	L -> R	14	= += 1= *= %= /= &= = ^= <<= >>=	L <- R
7	== !=	L -> R	15	,	L -> R
8	&	L -> R			

연산자 결합성

- 우선순위가 같은 경우 어떤 것을 먼저 연산할 것인지 나타내는 것
- $3 + 4 + 5$ 연산에서 두 덧셈 연산은 우선 순위가 같고 결합성이 $L \rightarrow R$ 이므로 $3 + 4$ 연산을 먼저 수행

산술 연산자

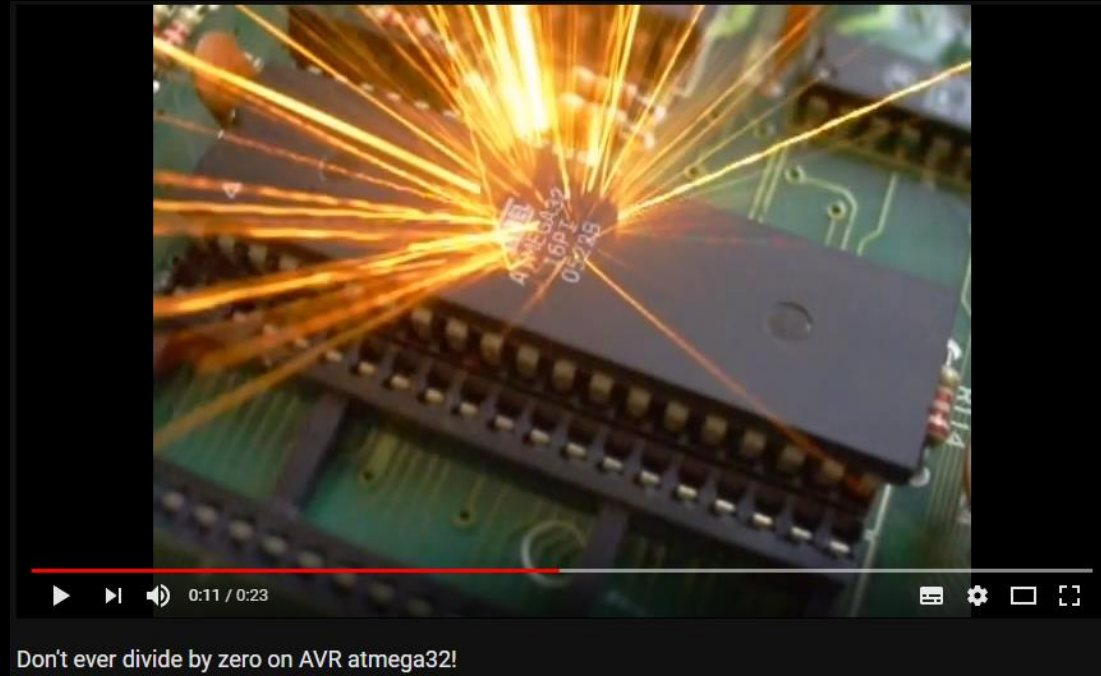
- 대표적인 2항 연산자
- $+$, $-$, $*$, $/$, $\%$
- 연산의 결과로 임시결과 발생
- 정수 간 나눗셈의 결과는 반드시 정수가 되며 소수점 이하는 절사

산술 연산과 형승격(Type promotion)

- 임시결과는 피연산자 표현범위 이상의 표현이 가능해야 함
- `char + int` 결과는 `int`
- `double * int` 결과는 `double`
- `4 / 3`과 `4.0 / 3`은 전혀 다른 연산

0으로 나누면 안 되는 이유

- 7을 0을 나누면?
- 7에서 0을 빼면 7이고 7은 0보다 크다.
- 7에서 0을 계속 빼면 언젠가는 0보다 작은 숫자를 만날 수 있는가?
- 만날 수 없다면 뽀빠 연산은 언제 끝날까?



<https://www.youtube.com/watch?v=mZ7pUADoo58>

필수 실습 문제

난이도 1, 제한시간: 15분

사용자로부터 두 정수를 입력 받아 평균을 출력하는 예제를 작성. 입력은 반드시 `scanf_s()` 함수를 사용하고 출력은 `printf()` 함수를 이용할 것.

평균값 출력 시 반드시 소수점 둘째 자리까지만 표시.

두 정수를 입력하세요.: 10 20

AVG: 15.00

필수 실습 문제

난이도 2, 제한시간: 30분

사용자로부터 정수로 초(Second)를 입력 받아
‘시:분:초’ 형식으로 출력. 시, 분, 초 정보는 모두
두 자리 정수로 표시하고 한 자리 숫자의 경우 앞에 0을
붙여 출력

4000

4000초는 1시간 06분 40초 입니다.

대입 연산자

- 단순 대입연산자는 두 피연산자 중 오른쪽 피연산자(r-value)의 값을 왼쪽 피연산자(l-value)에 저장하는 연산자
- l-value는 Overwrite가 발생하며 기존 값이 사라짐
- 상수는 l-value가 될 수 없음

필수 실습 문제

난이도: 2, 제한시간: 15분

두 변수의 값을 교환하는 코드를 작성. 사용자로부터 두 정수를 입력 받아 각각을 int형 변수 a, b에 저장하고 임시변수 tmp를 활용해 a, b의 값을 교환한 후 출력하는 프로그램을 작성.

3 4

a:4, b:3

복합 대입 연산자

- 기능상 단순 대입 연산자와 산술 연산자, 비트 연산자가 조합된 연산자
- 누적 연산 시 += 연산자 활용

필수 실습 문제

난이도 2, 제한시간: 15분

사용자로부터 세 정수를 입력 받아 총합을 출력. 사용자 입력이 저장되는 변수 하나와 값을 누적하는 변수 하나만 사용해 구현

1

2

3

Total: 6

형변환 연산자

- 피연산자의 형식을 강제로 변경해주는 단항 연산자
- 부적절한 변환 시 정보가 유실될 수 있음

단항 증/감 연산자

- 피연산자에 저장된 값을 1씩 증가시키거나 감소시키는 연산
- 피연산자는 반드시 쓰기가 가능한 l-value라야 함
- 전위식, 후위식 표기가 가능하며 후위식이 될 경우 연산자 우선순위가 전체 식을 평가한 후로 미뤄짐

비트 연산자

- 자료를 비트 단위로 논리 식을 수행하는 연산
- 보통 2진수로 변환해 판단
- AND, OR, NOT, XOR, Shift left, Shift right
- NOT은 단항, 나머지는 모두 2항 연산자

비트 마스크 연산

- 데이터에서 특정 영역의 값이 모두 0이 되도록 지우는 연산
- AND의 특징을 이용
- 0과 AND연산을 수행하면 결과는 무조건 0

필수 실습 문제

난이도 2, 제한시간: 15분

사용자로부터 두 정수를 입력 받아 뺄셈을 수행하고 결과를 출력하는 프로그램을 작성. 단, 절대로 뺄셈 연산자를 사용하지 말고 비트 연산자를 이용해 구현.

9 6

결과: 3

sizeof

- 피연산자의 자료형에 대해 수행하는 컴파일 타임 연산자
- sizeof(5)와 sizeof(int)는 결과가 같음
- 배열에 대해서도 적용 가능
- 최대한 자주 사용할 것!

관계 연산자

- 두 피연산자의 값을 비교(뺄셈)해 결과 도출
- 상등, 부등, 관계 연산자로 분류
- 상등(==), 부등(!=) 연산은 좌항에서 우항을 뺀 결과를 비교하는 관계연산
- 실수형에 대해 상등, 부등연산은 불가
- >, <, <=, >= (비교연산)

논리 연산자

- 항 혹은 연산식을 피연산자로 두는 논리합(OR), 논리곱(AND) 2항 연산자
- 논리 부정 연산자는 단항 연산자
- 값의 범위 표현 시 사용되는 것이 보통
- 결합성이 $L \rightarrow R$ 이므로 왼쪽에 등장하는 연산식을 우선 평가하고 이어지는 연산식을 수행할 것인지 판단
- 0은 거짓이고 0이 아니면 모두(음수포함) 참

Short circuit

- 논리 식은 왼쪽부터 실행
- 피연산자 항이 식일 경우 식부터 평가
- 논리합의 경우 왼쪽 조건이 만족되면 이후 식은 연산하지 않음
- 논리곱의 경우 마지막 식까지 모두 평가해 모든 결과가 참인지 확인

조건 (3항) 연산자

- C언어의 유일한 3항 연산자
- 조건식 ? 항A : 항B
- 논리적 오류를 피하려면 선택 대상 항은 괄호로 묶어 표기

필수 실습 문제

난이도 2, 제한시간: 20분

사용자로부터 점수(0~100)를 입력 받아 80점 이상이면 '합격' 그렇지 않으면 '불합격'이라고 출력하는 프로그램을 작성.

반드시 3행 연산자를 사용

점수를 입력하세요: 80

결과: 합격

필수 실습 문제 (최댓값 서바이벌)

난이도 3, 제한시간: 30분

사용자로부터 입력 받은 정수 중 가장 큰 수를 출력하는 프로그램을 작성. 정수는 부호가 있는 32비트 정수로 한정하며 `scanf_s()` 함수로 한 번에 한 값만 입력 받고 내부적으로 최댓값을 계속 갱신하는 구조로 작성.

10

20

30

MAX: 30

필수 실습 문제 (최댓값 토너먼트)

난이도 3, 제한시간: 30분

사용자로부터 입력 받은 정수 중 가장 큰 수를 출력하는 프로그램을 작성. 정수는 부호가 있는 32비트 정수로 한정하며 `scanf_s()` 함수로 한 번에 세 값을 모두 입력 받아야 함. 최댓값은 `printf()` 함수로 출력

10 20 30

MAX: 30

11. 기본 제어문

if

- 분기문(Branching statement)
- 조건식의 결과에 따라 절차상 수행할 구문이 달라질 수 있는 제어문
- 3항 연산자와 논리적으로 유사

if문 실습 시 주의사항

- 조건식은 괄호로 묶이며 뒤에 **세미콜론을 붙이지 않음**
- 실행할 구문이 여러 행이면 **중괄호로 묶어(Scope)주며 반드시 들여 쓰기를 일치시켜 주어야 함**

필수 실습 문제 (최댓값 구하기3)

난이도 2, 제한시간: 10분

사용자로부터 입력 받은 정수 중 가장 큰 수를 출력하는 프로그램을 작성. 정수는 부호가 있는 32비트 정수로 한정하며 scanf_s() 함수로 한 번에 한 값만 입력 받고 내부적으로 최댓값을 계속 갱신하는 구조로 작성. 반드시 if문을 사용할 것!

10

20

30

MAX: 30

필수 실습 문제

난이도 2, 제한시간: 20분

버스 기본 요금을 1000원으로 가정하고 20세 미만인 경우에는 요금을 25% 할인하고 20세 이상 성인은 할인하지 않음. 단, 나이가 20을 넘는 경우 20으로 강제 조정해 출력할 것.

15

나이: 15, 최종요금: 750원

25

나이: 20, 최종요금: 1000원

대표적인 수강생 실수 사례

- 조건문이 제대로 실행되지 않아요.
- 변수가 마음대로 변해요.
- 저는 잘못된 것이 없는데 12살이 성인이라고 나와요.

if else

- 조건식을 만족하는 경우와 그렇지 않은 경우 수행되는 식을 배타적으로 기술할 수 있는 제어문
- else 오른쪽에는 조건식도 세미콜론도 없음에 주의

if else 중첩

- 각종 제어문 내부에 다시 제어문을 넣을 수 있음
- 여러 발생가능한 경우의 수를 나열한 후 피라미드 식으로 분류하는 경우 유용
- 조건에 의한 분류와 선택 구조

필수 실습 문제

난이도 3, 제한시간: 30분

버스 기본 요금을 1000원으로 가정하고 나이에 따라 다음과 같이 요금을 할인율 적용

0~3	영유아	100% (무료)
4~13	어린이	50%
14~19	청소년	25%
20 이상	성인	0%

필수 실습 문제

난이도 3, 제한시간: 30분

버스 기본 요금을 1000원으로 가정하고 나이에 따라 다음과 같이 요금을 할인율 적용

0~3	영유아	100% (무료)
4~13	어린이	50%
14~19	청소년	25%
20 이상	성인	0%
65 이상	어르신	100%

다중 if

- 마치 if else를 여럿 연이어 기술한 것과 같은 형식
- 다수의 조건식이 등장

스코프 중첩과 식별자 검색 순서

- 변수의 접근 가능 범위는 선언된 **스코프로 제한됨** (지역변수)
- 현재까지 배우고 사용한 모든 변수는 **지역변수 및 자동변수**
- 자동변수는 선언된 **스코프를 넘어서면 자동으로 소멸**
- 식별자가 같을 경우 **최근 스코프가 우선**

switch-case

- if문처럼 정보를 분류하고 경우를 선택하는 제어문
- 조건식의 결과는 반드시 정수
- 각 case는 결국 상등 연산 시 일치하는 값에 해당하는 정수
- case는 콜론으로 끝남
- 한 case에 대해 break문 조합 (생략가능)

무조건 goto

- 프로그램의 흐름을 조건 없이 즉시 변경하는 제어문
- 프로그램의 흐름이 엉킬 수 있어 대부분 사용을 권하지 않음

12. 반복문

while

- if문과 유사한 구조
- 조건식이 참일 경우 구간 코드를 계속 반복해서 수행
- 반복 수행 구간코드 내부에서 반복을 멈출 수 있는 연산이 반드시 있어야 함
- 무한루프는 최악의 논리 오류 중 하나

필수 실습 문제

난이도 3, 제한시간: 20분

사용자로부터 1~9 범위 정수를 입력 받아 그 수만큼
‘*’를 출력하는 프로그램을 작성. 사용자 입력이 범위를
넘어서면 강제로 보정. *은 한 행에 이어서 출력하고
전체 출력이 끝나면 개행

3

* * *

5

* * * * *

for

- 계수 기반 반복문
- 반복 회수 조절에 관련된 코드를 한 행에서 볼 수 있어 while문에 비해 상대적으로 실수 가능성이 적음

필수 실습 문제

난이도 3, 제한시간: 20분

1~10까지 총합을 출력하는 프로그램을 작성.
while문과 for문으로 각각 작성하며 반복회수는
10회로 제한. 반드시 총합을 누적하는 변수를 활용할 것

Total: 55

반복문의 중첩

- 1차원적 반복 출력 코드를 다시 반복문으로 묶어 2차원적 구조를 출력
- 반복에 대한 반복이 발생해 논리적으로 한 층 더 복잡

필수 실습 문제

난이도 3, 제한시간: 20분

5행 5열 사각형을 *로 출력하는 프로그램을 작성.
반드시 for문을 중첩하는 구조로 개발.

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

필수 실습 문제

난이도 3, 제한시간: 20분

아래 예처럼 삼각형을 *로 출력하는 프로그램을 작성.
반드시 for문을 중첩하는 구조로 개발.

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

필수 실습 문제

난이도 3, 제한시간: 30분

아래 예처럼 삼각형을 *로 출력하는 프로그램을 작성.
'*' 왼쪽 여백은 Tab으로 조절

```

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

필수 실습 문제

난이도 3, 제한시간: 30분

아래 예처럼 삼각형을 *로 출력하는 프로그램을 작성.
'*' 왼쪽 여백은 Tab으로 조절

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

do while

- 반복 대상 구간코드가 조건식의 만족 여부와 관련 없이 무조건 한 번은 실행하는 반복문
- 기존 while문과 달리 조건식 뒤에 세미콜론이 있음

break

- 반복문과 switch-case문에 사용되며
수행 시 **스코프를 즉시 벗어나 그 다음
구문으로 이동**
- 반복문 내부에서 break 수행 시 조건과
상관 없이 **반복문 종료**

continue

- break문과 달리 반복을 멈추지 않으며 반복문 내부에서 continue문 이후 코드를 수행하지 않고 그냥 넘어가 반복 시작 조건식 비교
- 반복은 계속되며 논리적으로 복잡해지는 원인이 될 수 있음

break와 중첩 Loop

- 반복문이 중첩됐을 경우 내부 반복문에서 break문이 수행되면 내부 반복만 멈춤
- 외부 반복문은 계속 수행

Part 3

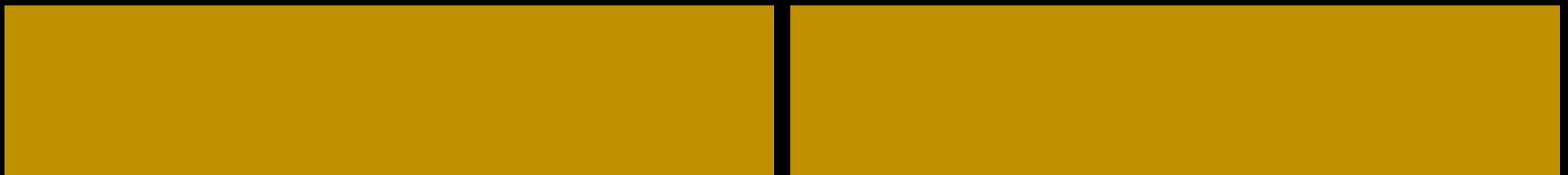
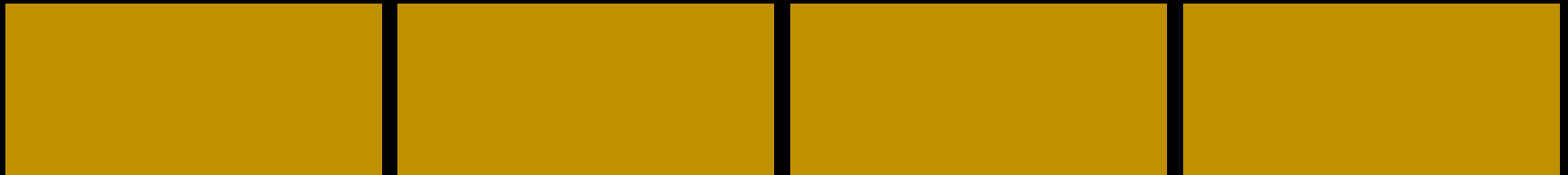
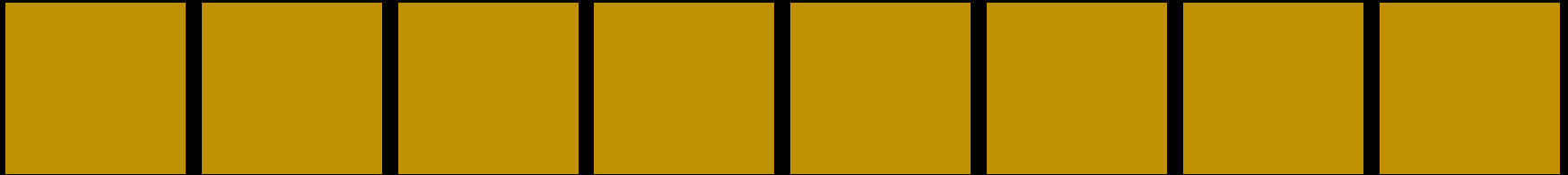
C언어 수준 향상

13. 배열과 프로그래밍 기법

배열 (Array)

- 형식이 같은 자료 여러 개를 모아 한 덩어리로 관리하는 문법
- 여러 요소를 식별하기 위해 **인덱스**를 사용하며 0부터 시작 (**Zero-based index**)
- **배열의 이름**은 0번 요소의 **메모리 주소**에 대한 식별자이며 **상수** (l-value 아님)

배열 (Array)



문자 배열 (문자열)

- 문자열의 실체는 `char[]`
- 문자열 상수란 쓰기가 허용되지 않는 이름이 없는 `char[]`
- 문자열의 끝은 반드시 `\0 (null)`
- 문자열 상수는 보통 포인터를 사용해 관리

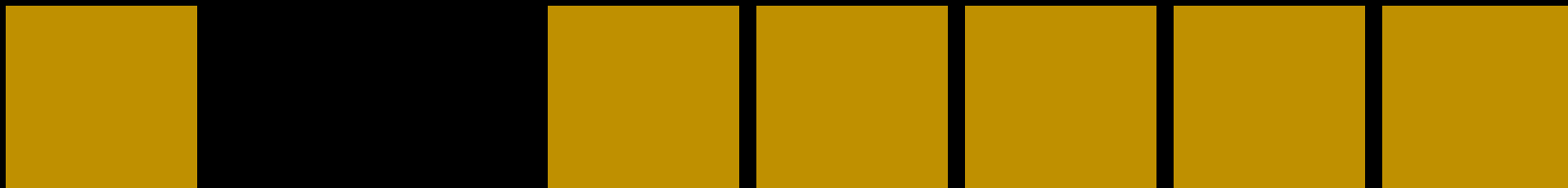
필수 실습 문제

난이도 2, 제한시간: 20분

for문을 사용해 `int[5]` 배열에 저장된 값 중 가장 큰 값을 찾아 출력하는 프로그램을 작성. 최댓값이 저장된 변수는 `int nMax`로 선언.

50 40 10 50 20

MAX: 50



필수 실습 문제

난이도 3, 제한시간: 20분

for문을 사용해 `int[5]` 배열에 저장된 값 중 가장 작은 값을 찾아 출력하는 프로그램을 작성. `int[0]` 요소에 최솟값이 저장될 수 있도록 개발. 각 배열 요소를 비교하는 과정에서 두 배열 요소의 값은 덮어써서 사라지지 않도록 교환.



필수 실습 문제

난이도 3.5, 제한시간: 30분

for문을 사용해 `int[5]` 배열에 저장된 값들을
오름차순으로 정렬하는 프로그램을 작성. 정렬 방식은
버블정렬 알고리즘을 사용

필수 실습 문제

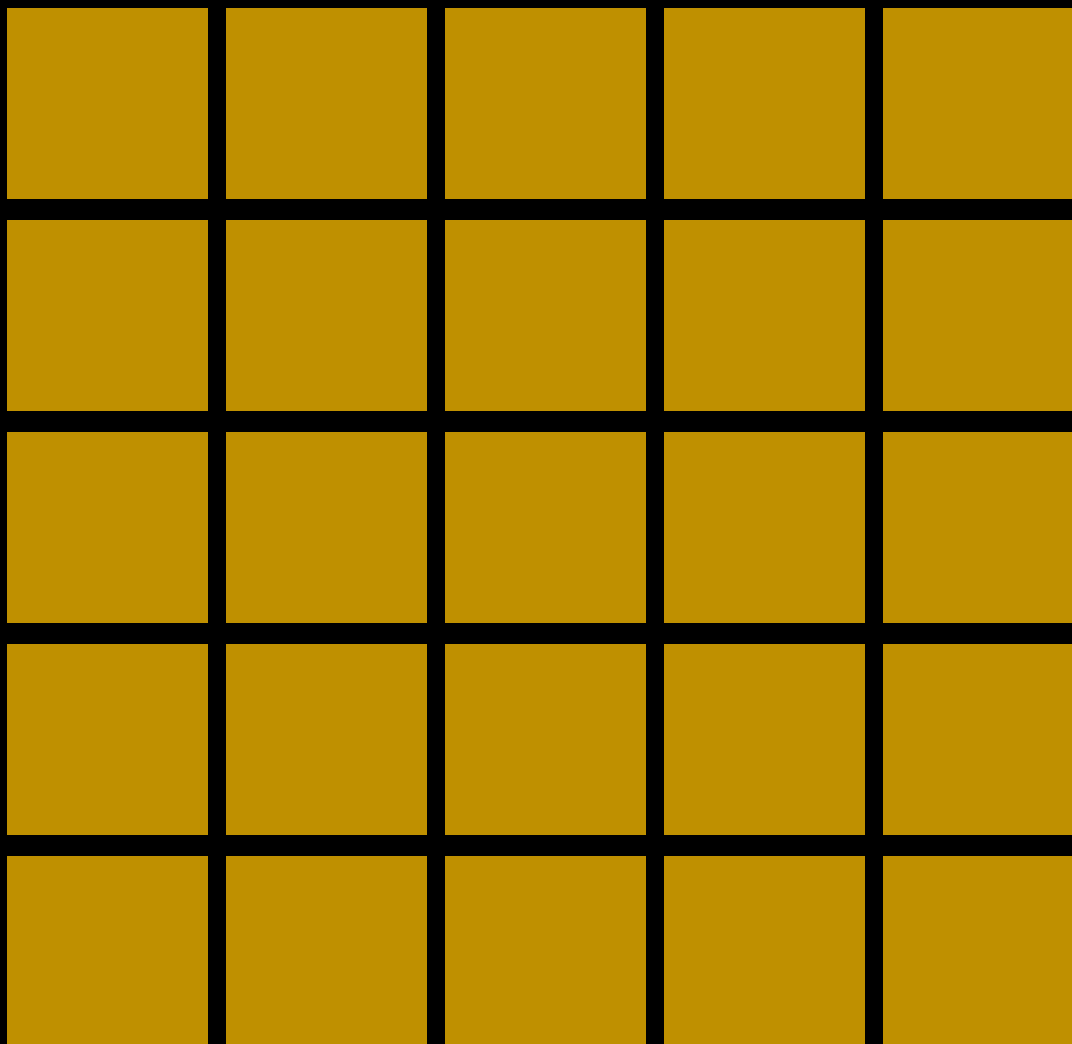
난이도 4, 제한시간: 30분

for문을 사용해 `int[5]` 배열에 저장된 값들을
오름차순으로 정렬하는 프로그램을 작성. 정렬 방식은
선택정렬 알고리즘을 사용.

다차원 배열

- 배열의 요소가 배열인 배열
- 2차원 구조일 경우 [행][열]
- 3차원 구조일 경우 [면][행][열]

2차원 배열



필수 실습 문제

난이도 3, 제한시간: 20분

```
int aList[3][4] = {  
    {10, 20, 30},  
    {40, 50, 60}  
};
```

상기 배열 요소의 행, 열 총합을 다음과 같이 출력하는 프로그램 작성.

```
10 20 30 60  
40 50 60 150  
50 70 90 210
```

필수 실습 문제

난이도 3, 제한시간: 30분

for문을 사용해 `int[5][5]` 배열에 다음과 같이 저장하는 프로그램을 작성.

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

필수 실습 문제

난이도 3.5, 제한시간: 30분

for문을 사용해 `int[5][5]` 배열에 다음과 같이 저장하는 프로그램을 작성.

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

필수 실습 문제

난이도 3.5, 제한시간: 30분

for문을 사용해 `int[5][5]` 배열에 다음과 같이 저장하는 프로그램을 작성. (달팽이 배열)

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

Lookup 배열

- 데이터를 검색 혹은 참조하기 위해 사용하는 배열
- 보통 기준에 따라 인덱스 값을 계산하고 식별한 요소를 활용하는 구조
- 배열의 한 요소가 처리할 하나의 경우로 활용

14. 함수에 대한 기본 이론

사용자 정의 함수

- 절차적 흐름을 갖는 여러 구문을 하나로 묶어 사용하는 단위 코드
- C언어 프로그램을 구성하는 단위
- **main()** 함수의 시작과 끝이 곧 프로그램의 시작과 끝
- 여러 함수가 존재 할 경우 호출을 통해 연결 (bind)

사용자 정의 함수

- 함수는 반환자료형 이름 (매개변수 목록) 형식으로 기술
- 호출자 함수와 피호출자 함수로 관계를 규정할 수 있음 (Binding)
- 호출자는 피호출자 함수의 매개변수 초기값을 기술해야 할 의무가 있음
- 피호출자 함수는 호출자 함수에게 값을 반환

두 가지 함수 설계 원칙

- UI와 기능은 반드시 분리
- 재사용 가능한 단위 코드는 함수로 구현 (DRY 원칙)

필수 실습 문제

난이도 3.5, 제한시간: 30분

사용자로부터 세 정수를 입력 받아 최댓값을 반환하는 함수를 작성. 사용자 입력을 받는 부분과 최댓값을 계산하는 코드는 반드시 별도 함수로 분리.

이벤트 루프

- `main()` 함수에서 사용자 인터페이스 출력 및 사용자 입력을 반복하는 구조
- 보통 메뉴출력과 사용자 선택을 확인
- 메뉴 선택에 따라 기능 수행
- 대부분의 응용 프로그램이 채택하는 일반적 구조

함수 원형 선언

- 함수의 선언과 정의를 분리
- 컴파일러에게 함수의 존재를 알리기 위해 분리한 선언을 코드 상단에 기술
- 함수 바디 없이 원형만 기술할 경우 함수 시그니처라고 부르기도 함
- 함수 원형 선언에서는 매개변수 이름은 생략 가능

분할 컴파일

- 한 프로젝트에 여러 소스코드 파일(.c)을 운영하는 경우 각각을 개별 컴파일
- 개별 소스코드에 대한 목적파일 생성
- 변수나 함수 선언과 정의가 다른 파일로 분리 될 수 있음
- 링크 오류에 주의

전역변수와 식별자 검색 순서

- 전역변수는 함수 바디 밖에 선언
- 식별자 검색 순서
 1. 지역 스코프
 2. 최대 함수 바디
 3. 최대 로컬 파일
 4. 외부 파일까지 확장

15. 메모리와 포인터

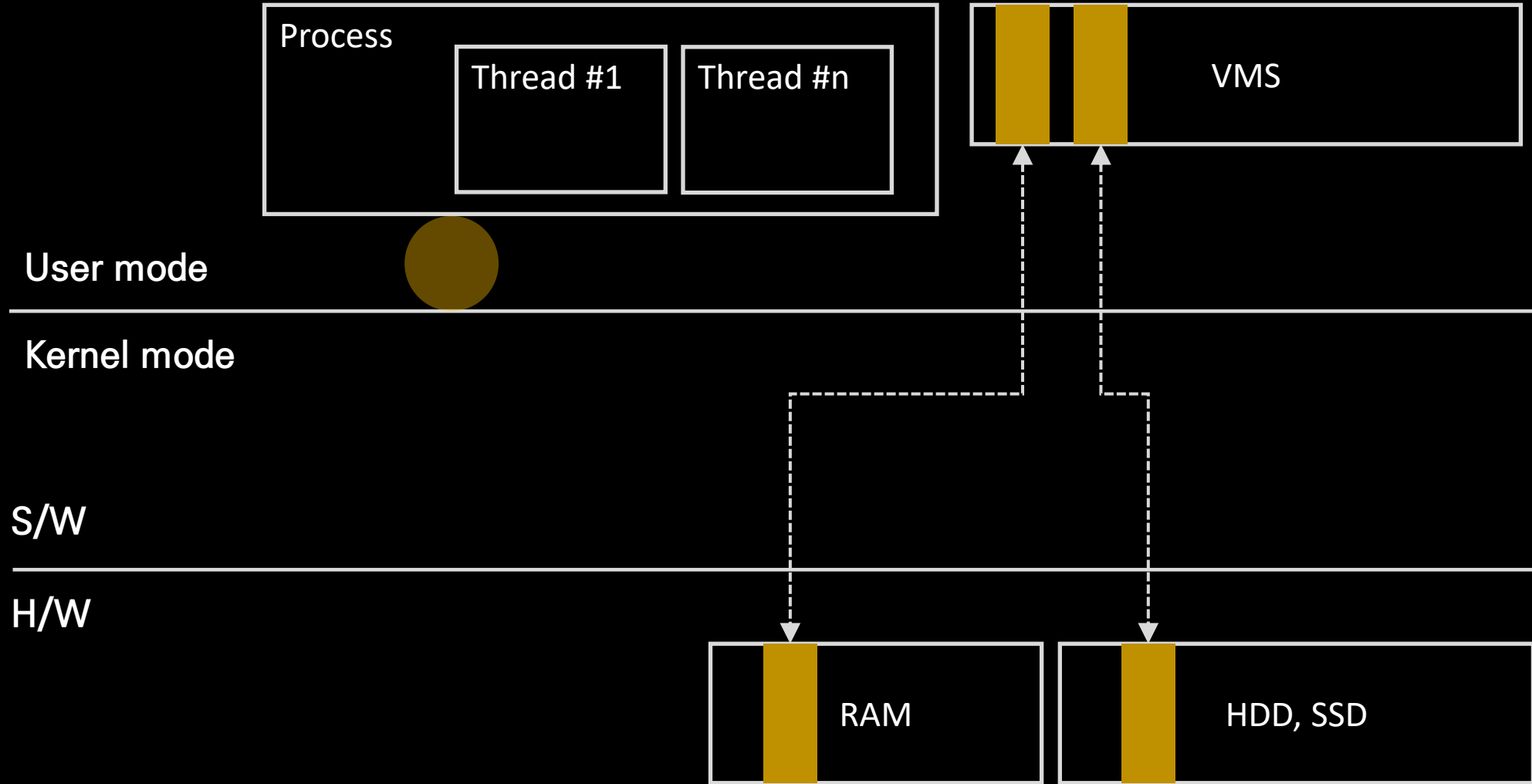
컴퓨터와 메모리

- CPU는 연산할 코드와 대상 정보를 모두 메모리로부터 가져옴
- 메모리는 변수를 통해 사용
- 모든 메모리는 고유한 주소를 가짐 (바이트 단위)
- 64bit 시스템에서 메모리 주소 길이는 64bit임

메모리 종류

- **Stack** (자동변수, 보통 1MB)
- **Heap** (동적 할당 메모리)
- 실행 코드
 - **text section** (실행 코드 기계어)
 - **data section**
 - Read only (문자열 상수)
 - Read/Write (정적 메모리)

Windows 가상 메모리 시스템



메모리 관리 함수

- malloc(), calloc() / free()
- realloc()
- memcpy(), strcpy()
- memcmp(), strcmp()
- sprintf()

포인터 변수

- 메모리의 주소를 저장하기 위한 전용 변수
- 64bit 시스템에서 주소 상수, 포인터 변수는 모두 64bit(8 bytes)
- 1byte char형 변수의 메모리 주소는 64bit
 - `char *pszData = &ch; //char ch = 'a';`

직접 지정과 간접 지정

- 특정 메모리 공간을 int로 지정할 때 상수로 지정하면 직접 지정
- 포인터 변수로 지정하면 간접 지정
- 간접지정은 변경될 수 있는 임의의 기준주소로 상대적인 위치를 식별하는 방식

포인터와 1차원 배열

- 배열을 이루는 **요소 형식에 대한 포인터** 변수를 선언하는 것이 일반적
- **char []**은 **char ***로 관리
- **int []**은 **int ***로 관리
- 간접 지정 연산 (*)의 결과는 **형식이 있는 변수**로 생각할 수 있음

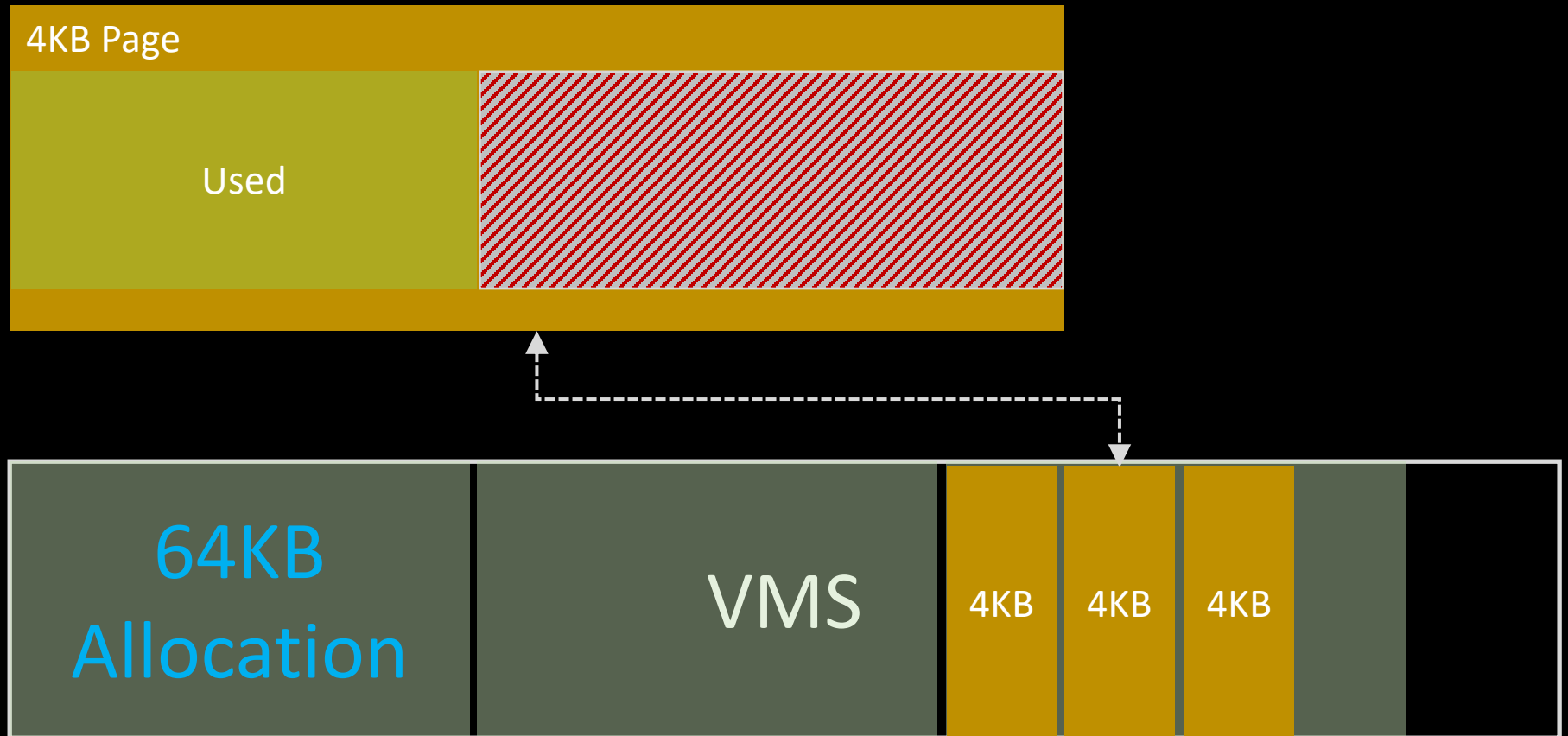
포인터와 1차원 배열

- 포인터 변수나 배열 이름에 대해 덧셈, 뺄셈 연산을 할 수 있음
- 이 덧셈, 뺄셈은 산술 연산이 아니라 상대 위치를 계산하기 위한 연산이며 배열 요소의 개수를 의미
- 포인터 변수에 대해서는 단항 증/감 연산도 가능

메모리 동적 할당 및 관리

- Heap 영역을 사용하는 방법
- 프로그램 실행 중 필요한 메모리를 OS에 요청(할당)해 사용하며 반환(해제)의 책임이 있음
- 할당 받은 메모리는 쓰레기 값이 들어 있음
- `malloc()` / `free()`

할당 단위와 페이지



메모리 값 복사

- 단순 대입 연산자의 두 피연산자가 모두 변수라면 메모리 값을 복사하는 것으로 생각할 수 있음
- 배열에 대해서는 단순 대입 연산으로 배열 전체를 복사 할 수 없으며 반복문을 통해 개별 요소를 하나씩 복사(단순 대입)
- `memcpy()`

필수 실습 문제

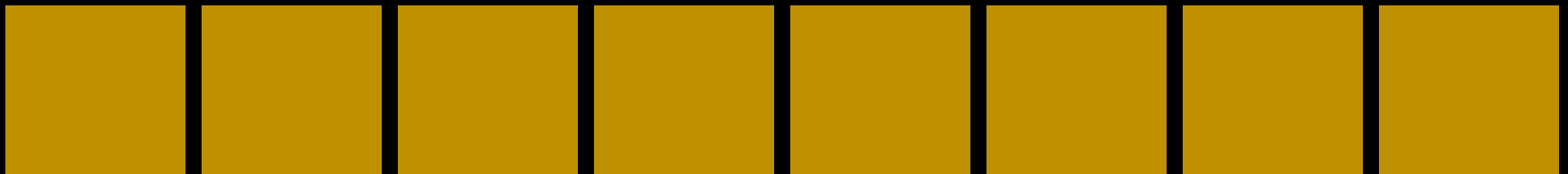
난이도 4, 제한시간 20분

아래 코드의 결함에 대해 말하고 바르게 수정하시오

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    char szBuffer[12] = {"HelloWorld"};
    char *pszData = NULL;
    pszData = (char*)malloc( sizeof(char) * 12 );
    pszData = szBuffer;
    puts(pszData);
    return 0;
}
```


배열 연산자 포인터로 풀어쓰기

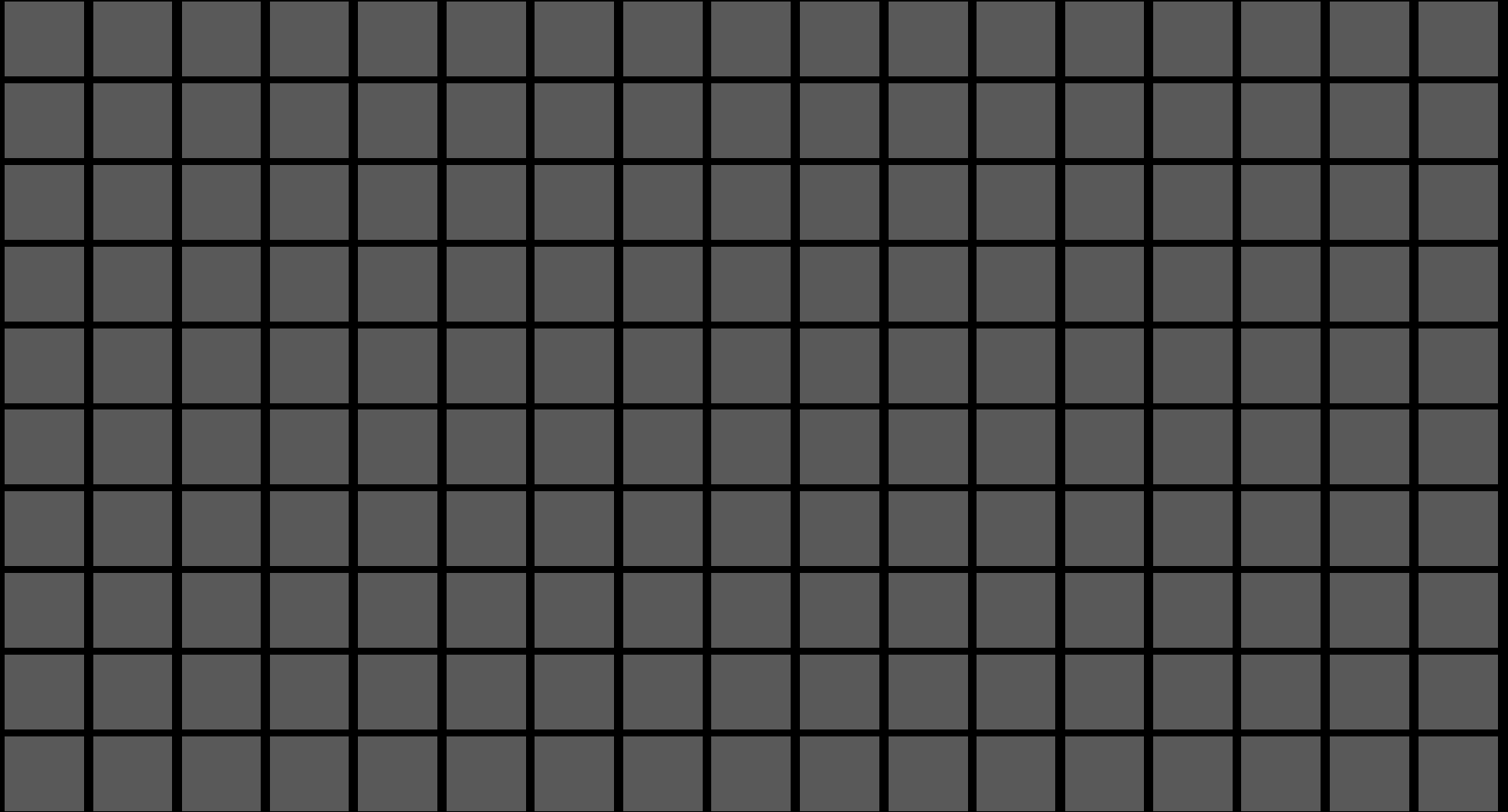
- 배열의 이름은 기준주소
- 인덱스를 이용해 상대위치를 계산
- $*(\text{기준주소} + \text{인덱스})$ 는
 $\text{기준주소}[\text{인덱스}]$ 로 쓸 수 있음



realloc()

- 기존에 할당 받은 메모리의 크기를 조정해 다시 할당
- 메모리 Chunk 크기 조절에 실패할 경우 전혀 새로운 위치로 이동

동적 할당된 메모리 구조와 realloc()



다중 포인터

- `char*`에 대해 `*(char*) == char`
- `char**`에 대해 `*(char**) == char*`
- `char***`에 대해 `*(char***) == char**`

다차원 배열에 대한 포인터

- 2차원 배열은 1차원 배열을 요소로 갖는 1차원 배열로 이해
- `char aStrList[2][12]`
 - `char[12]`를 요소로 갖는 배열
- `char (*pStrList)[12]`
 - 요소가 `char[12]`인 배열에 대한 포인터

기억부류 지정자 (Storage-class specifier)

- extern, auto, static, register
- 자동변수는 **Stack**을 사용하며 일반적인 지역변수는 모두 **자동변수**
- **정적 메모리**는 프로그램 시작 시 확보되는 영역이며 프로그램 종료 시까지 유지 (동시성 이슈 있음)

16. 함수 응용

매개변수 전달 기법

- Call by value
- Call by reference
 - C언어에서는 참조형을 포인터로 구현
- 인수, 매개변수, 파라미터, 아규먼트 등은 다 같은 말
- 매개변수는 Stack 영역 사용

필수 실습 문제

난이도 4, 제한시간 30분

두 `char[]`의 주소를 매개변수로 받아 문자열을 Deep copy하는 `MyStrcpy()` 함수를 작성. 함수의 두 번째 매개변수는 첫 번째 매개변수의 메모리 크기가 되도록 구현.

지역변수 주소 반환오류

- 피호출 함수의 지역변수는 함수의 반환과 함께 모두 소멸
- 소멸된 메모리 영역의 주소를 호출자 함수에게 반환하고 접근하는 것은 매우 심각한 오류

메모리 동적 할당 및 해제와 함수

- Callee가 메모리를 동적 할당한 후 반환하는 구조는 문제의 여지가 있음
- 메모리 해제에 대한 확실한 안내필요
- 할당된 메모리 크기 전달문제 고려

재귀 호출

- 함수 코드 내부에서 다시 자신을 호출하는 것
- 반복문과 **Stack** 자료구조를 합친 것
- 비선형 자료구조에서 매우 중요하게 활용
- 함수 호출 오버헤드는 감수
- 논리 오류 발생 시 **Stack overflow** 발생

문자 처리 함수

- `isalpha()`, `isdigit()`, `isxdigit()`, `isalnum()`,
`islower()`, `isupper()`, `isspace()`,
`toupper()`, `tolower()`
- `gets()`, `gets_s()`, `puts()`
- `sprintf()`, `printf()`, `scanf_s()`
- `strcpy()`, `strcat()`, `strstr()`
- `strpbrk()`, `strtok()` ...
- ChatGPT를 사용할 것

유틸리티 함수

- `atoi()`, `atol()`, `atof()`
- `time()`, `localtime()`, `ctime()`
- `srand()`, `rand()`
- `system()`, `exit()`

17. 구조체와 공용체

구조체

- 여러 자료형을 모아 새로운 하나의 형식으로 기술 (선언)
- 배열은 같은 것이 모인 것이며 구조체는 서로 다른 것들이 모인 것으로 이해 할 수 있음 (단, 인덱스는 없음)
- 구조체는 하나의 새로운 사용자 정의 형식으로 작동

구조체

- 구조체를 이루는 요소를 **멤버**라고 부름
- 구조체 변수를 통해 개별 요소에 접근할 때는 **멤버접근 연산자**를 사용
- 구조체 변수(혹은 인스턴스) 선언 시 초기값을 기술했을 때는 반드시 멤버 선언 순서에 맞춰야 함
- **typedef** 선언을 동반하는 것이 일반적

구조체 관리

- 구조체도 배열 선언 가능
- malloc() 함수로 동적 선언해 관리하는 경우도 일반적
- 구조체에 대한 포인터 변수 선언 시 멤버 접근 연산자가 달라짐
- 함수 반환형이나 매개변수로 사용 가능

구조체를 멤버로 갖는 구조체

- 구조체 변수도 다른 구조체의 멤버가 될 수 있음
- 이 경우 컴파일러에게 선언을 알려야 하므로 기술 순서가 중요
- 자기 자신에 대한 포인터를 멤버로 갖는 구조체를 자기 참조 구조체라 함

필수 실습 문제

난이도 4, 제한시간 30분

간단한 연결 리스트 예제를 활용해 연결 리스트 전체의 내용을 출력하는 함수를 작성하시오. 함수의 매개변수로는 출력할 노드의 주소를 받아야 하며 재귀호출을 통해 다음 노드로 이동하도록 구현.

비트 필드

- 1바이트 (8비트) 정보를 쉽게 잘라 쓰기 위한 문법
- 멤버는 비트 단위 데이터
- 멤버 선언 시 먼저 기술한 멤버는 8비트 중 오른쪽 부터 적용

공용체

- 한 대상에 대해 여러 해석방법(자료형)을 부여하는 문법
- 32비트 정수에 대해 `int`, `short[2]`, `char[4]`으로 해석 가능

구조체를 멤버 맞춤

- 구조체를 이루는 멤버의 메모리 공간은 관리 편의를 위해 **완전히 연접하지 않고 일정 단위로 메모리를 구성**
- 완벽히 연접해야 할 경우 전처리기를 이용해 설정 변경
 - `#pragma pack()`

18. 파일 입/출력

파일 시스템 기본 이론

- 2차 메모리를 관리하는 방법으로 파일 시스템 사용
- 윈도우 파일 시스템(**NTFS**)에서는 논리 드라이브 단위 및 경로 사용
- .은 현재, ..은 상위 디렉토리를 의미
- 절대 경로와 상대 경로가 있음

볼륨 및 디스크 정보

```
Microsoft Visual Studio 디버그 콘솔
Volume name : SSD 512, File system : NTFS
(Disk free space: 149/465 GB)


F:\Windows 시스템 프로그래밍\FileInfoSample\Debug\FileInfoSample.exe(프로세스 35596개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요....
```


SSD 512 (C:) 속성

일반 도구 하드웨어 공유 보안 이전 버전 할당량

 SSD 512

종류: 로컬 디스크
파일 시스템: NTFS

 사용 중인 공간:	339,440,631,808바이트	316GB
 여유 공간:	160,021,278,720바이트	149GB
용량:	499,461,910,528바이트	465GB


드라이브 C: 디스크 정리(D)

파일 시스템 기본 이론

- 파일은 프로세스가 접근 주체
- 여러 프로세스가 존재하는 시스템에서 한 파일에 대한 동시접근 시 데이터가 엉키는 문제가 발생할 수 있음
- 파일 입/출력에 사용되는 내부 버퍼가 있을 수 있음
- 장치는 파일로 추상화 됨

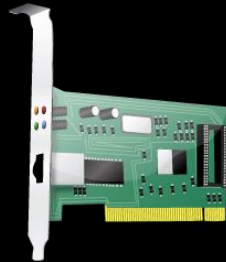
데이터 파일과 장치 파일

User mode

Kernel mode

S/W

H/W



HDD, SSD

파일 스트림

File stream

파일명, 확장명, 바이너리와 텍스트

- 윈도우 시스템에서 파일명은 파일이름과 확장명으로 구성
- 파일 형식은 바이너리와 텍스트로 나누며 텍스트는 바이너리에 포함되는 것으로 볼 수 있음
- 텍스트 파일은 문자열로 해석할 수 있는 바이너리만 담긴 파일

파일 입/출력 함수

- `fopen()` / `fclose()`, `_fcloseall()`
- `fprintf()` / `fscanf()`
- `fputc()` / `fgetc()`
- `fputs()` / `fgets()`
- `fflush()`
- `fread()` / `fwrite()`, `fseek()`, `rewind()`, `ftell()`

fopen() 접근 모드 문자

모드	의미	존재하는 경우	없는 경우
r	Text read	파일 열기	에러
w	Text write	기존 내용 지우고 열기	새로 만듦
a	Text append	기존 내용 뒤에 추가	새로 만듦
r+	Text read and update	파일 열기	에러
w+	Text write and update	기존 내용 지우고 열기	새로 만듦
a+	Text append and update	기존 내용 뒤에 추가	새로 만듦
rb	Binary read	파일 열기	에러
Wb	Binary write	기존 내용을 지우고 열기	새로 만듦
ab	Binary append	기존 내용 뒤에 추가	새로 만듦
rb+	Binary read and update	파일 열기	에러
wb+	Binary write and update	기존 내용을 지우고 열기	새로 만듦
ab+	Binary append and update	기존 내용 뒤에 추가	새로 만듦

파일 플러싱

User mode

Kernel mode

S/W

H/W

HDD, SSD

파일 포인터 위치 제어



Part 4

고급 이론

19. 변수와 상수 고급 이론

형한정어 (Type qualifier)

- 변수에 적용하는 문법으로 컴파일러 최적화에 깊이 관련
- `const`
- `volatile`
- `extern`
- `typedef`

const

- 변수를 상수화 하는 문법
- 개념상 읽기 전용으로 만들어주는 것
- 변수의 개수가 줄어들기 때문에 성능 최적화(Release mode)에 유리

심볼릭 상수

- 프로그램 내부에서 특별한 의미를 갖는 기준 값에 대해 이름을 부여
- 읽기 좋은 코드를 만드는 방법
- 심볼릭 상수 (이름이 있는 상수) 정의에 사용 (#define 전처리기로도 가능)

const와 포인터

- 포인터가 가리키는 대상 상수화
- 포인터 변수 자체를 상수화
- Call by reference 상황에서 피호출자 함수에서 호출자가 제시한 메모리를 읽기 전용으로 차단가능

extern 선언

- 외부변수선언 시 사용되는 문법
- 한 프로젝트 내부에 여러 C파일이 있을 경우 다른 .c 파일에 정의되어 있는 전역변수에 접근하기 위해 선언
- 같은 .c 파일에서는 내부, 다른 파일이면 외부

형 재선언

- 새로운 자료형을 선언하는 문법
- C언어가 제공하는 기본 형식의 이름을 바꾸는 용도로도 사용
- 구조체나 공용체 선언과 조합해 편의성 확보
- 남발 시 읽기 어려운 코드가 될 우려발생

열거형 상수

- 여러 개의 심볼릭 상수를 한번에 정의할 수 있는 사용자 정의 형식
- 정수 값으로 계산 될 수 있음
- switch-case 문과 결합해 읽기 좋은 코드를 만들 수 있음

20. 전처리기

전처리기

- 컴파일 전에 선행처리를 위한 문법
 - 헤더 포함
 - 조건부 컴파일
 - 심볼릭 상수 정의
 - 매크로 정의
- #기호로 시작

#include

- 헤더 파일(선언 코드가 들어있는 파일)을 소스코드에 **합쳐주는 기능**
- **<>**로 포함 시 컴파일러가 정의하고 있는 **시스템 헤더 파일들** 중 검색
- **“”**로 포함 시 **현재 경로에서 파일** 검색

#define

- 형한정어 const처럼 **심볼릭 상수**를 **정의**할 수 있는 전처리기
- 정의한 상수는 컴파일 전에 적용되어 **소스코드를 치환**

매크로

- `#define` 전처리로 정의
- 함수처럼 보이지만 **함수가 아님**
- 함수 호출에 의한 오버헤드가 없음
- 컴파일러 최적화에 따라 **사용빈도가 현저히 줄어듦**

조건부 컴파일

- 상수 정의 여부에 따라 실제 컴파일 되는 코드가 달라지도록 구성하는 것이 목적
- `#ifdef`, `#else`, `#endif` 로 구성
- Debug/Release 빌드 선택
- 문자열(MBCS, Unicode) 선택

21. 함수에 대한 고급 이론

함수 포인터

- 함수의 이름(상수)을 저장할 수 있는 포인터
- 함수호출 연산자의 피연산자가 될 수 있음
- 반환 자료형 (호출규칙 *변수명)(매개변수) 형식으로 선언

함수 호출 규칙

- 32비트 호출규칙
 - `__cdecl` (C언어 기본)
 - `__stdcall`
 - `__fastcall`
- 64비트 호출규칙
 - `__fastcall`

Call back 구조

- 함수가 호출하는 것이 아니라 호출 되는 구조
- 함수의 이름(주소)를 라이브러리나 프레임워크에 전달하면 그 내부에서 호출되는 구조
- 호출 시점과 횟수를 정확히 특정하기 어려운 경우가 많음

감사합니다!