

C Programming Language

$$C = \sum f(x)$$

Part I

Structured programming in C

등장 인물 소개

폭소 멀더	외모	남자, 키 185cm 에 마른편
	경력	전 FBI 요원, XXX-Files 부서에서 퇴출됨 현재 탐정 사무소 직원
	성격	강한 신념의 소유자, 통찰력이 있다고 스스로 생각함 약간 사이코(?)
	기타	아버지가 FOX TV 직원이었음(생사 모름)
다보 스컬리	외모	여자, 키 160cm 에 글래머
	경력	전 FBI 요원, XXX-Files 부서에서 멀더 감시 임무 현재 탐정 사무소 사장
	성격	논리적이고 합리적이거나 차가움
	취미	시체 해부
부국장	외모	남자, 대머리
	성격	입이 가벼움
담배 피는 남자	외모	남자, 키 190cm 에 중절모와 롱코트, 골초
	성격	모름
	기타	전 FOX TV 직원

#Scene 1 스컬리 & 멀더의 탐정 사무소

<멀더가 계단을 올라 사무실 문을 열려고 하는데 문 밑에 무엇인가가 떨어져 있었다>

멀더 음? 이건 뭐지?
 (문 밑에서 손으로 집어 들며)
 USB 메모리 스틱이군!
 뭐라 써 있는데?
 씨는 시그마 에프엑스(C = $\sum f(x)$) ?
 (한 손으로는 물체를 두리번 거리며 한 손으로는 사무실 문을 연다)
 스컬리! 문 앞에 뭔가가 있어요.

스컬리 그래요? 어떤 건가요?
 (멀더가 손을 들어 엄지와 검지로 물건을 뒤틀며 보여준다)
 USB 메모리 인가요?

멀더 네, 그런거 같군요.
 누군가 떨어뜨렸거나 우리가 보기를 원하는 것처럼 생각되는 군요.

스컬리 멀더, 그냥 거기에 놔두는 건 어떨까요?
 주인이 찾으러 다시 올지도 모르잖아요.

멀더 음~ 하지만 우리에게 의도적으로 떨어뜨렸다면 주인이 안 올 수도 있잖아요?
 나는 이 안에 있는 내용이 궁금해지는군요

스컬리 그런 궁금증은 옳지 않은 것 같은데요, 주인이 와서 왜 보고 있냐고 하면 어쩌려고요?

멀더 찾아 온 주인이 꼭 주인일까요? 알 수 없잖아요.
 주인인지 아닌지 확인하려면 우리가 내용을 알고 있어야 할겁니다.
 컴퓨터에 꽂아서 일부를 확인해 보죠

스컬리 그렇게 보고 싶다면 당신 컴퓨터에 꽂아 보세요. 저는 그 일에 참여하지 않겠어요!

<멀더는 급하게 자신의 데스크톱 컴퓨터에 USB를 먼저 꽂고 컴퓨터 전원을 켜다>

#Scene 2 멀더의 데스크톱 모니터

멀더 (발을 동동 구르며) 빨리 빨리

 컴퓨터가 구형이라 부팅이 느려요, 찻

스컬리 그 컴퓨터는 1년 전에 산 거 아닌가요? 제 꺼는 3년 전 노트북이라고요.
 조바심은 금물이에요.

<컴퓨터는 부팅이 완료되고 멀더는 마우스를 움직여 USB 드라이브를 클릭한다>

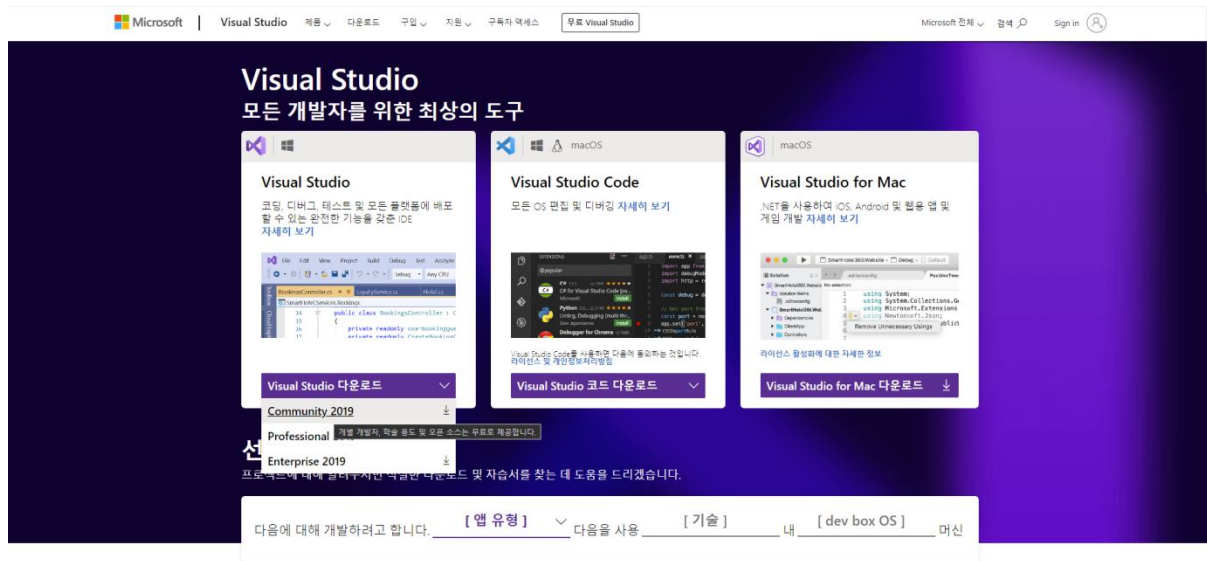
멀더 c_sigma_fx 폴더만 안에 있군요. (폴더 클릭)

 part_1 폴더가 있군요. 일단 part_1 폴더를 열어 보죠.

<part_1 폴더 안에는 c_sigma_fx_part_1_00 ~ c_sigma_fx_part_1_22 폴더들이 보인다>

```
c_sigma_fx_part_1_00.c  
c_sigma_fx_part_1_00.sln  
c_sigma_fx_part_1_00.vcxproj  
c_sigma_fx_part_1_00.vcxproj.filters  
c_sigma_fx_part_1_00.vcxproj.user
```

- 멀더 같은 이름에 확장자가 다른 파일들이 있군요.
스컬리 스컬리 와서 좀 봐봐요. (스컬리에게 오라는 손짓을 한다)
- 스컬리 관여하고 싶진 않지만 파일명만 볼 게요. (머리를 가로 저으며 멀더의 책상으로 이동한다)
다행이 제가 아는 파일들이네요.
이 파일들은 Microsoft Visual Studio 솔루션 파일들이에요.
c_sigma_fx_part_1_00.c 의 확장자로 보아 C 프로그래밍 언어(이하 C 언어) 솔루션이 맞아요.
제가 학생 시절 [AT&T 벨 연구소](#) 에서 잠깐 일했을 때 사용한 솔루션과 비슷해요.
- 멀더 Microsoft Visual Studio 라는 것은 뭔가요?
C 언어는 저도 아주 젊었을 때 쓴 기억이 있어서 알고 있어요.
- 스컬리 C 언어를 안다니 다행이네요.
그럼 컴파일러도 알고 있겠군요. Microsoft Visual Studio 는 컴파일러예요.
C 언어를 컴파일 해서 실행 가능한 파일을 생성하는 기능이에요.
그럼 인터넷을 통해서 설치를 해 보아요.



스컬리

인터넷 웹 브라우저에서 Microsoft Visual Studio 를 검색하고 해당 사이트를 클릭하면 위와 같은 화면이 나와요.

우리는 상업용으로 사용하는 것이 아니니까 Community 버전을 선택하면 무료로 사용할 수 있어요.

아래 인터넷 주소에 접속하면 설치 방법이 자세히 나와있어요.

[Visual Studio 2019 설치하기](https://visualstudio.microsoft.com/ko-kr/thank-you-downloading-visual-studio/?sku=Community&rel=16)

<멀더는 스컬리가 알려준 대로 Visual Studio 를 설치 완료한다>

#Scene 5

C 프로그래밍 언어

멀더

스컬리. 아까 AT&T 벨 연구소에 있었다고 했죠?
C 언어가 그 곳에서 태어났다는 것도 알고 있겠군요?
Brian W. Kernighan & Dennis M. Ritchie 도 알고 있겠쥬.
하하하

스컬리

물론 저도 알고 있어요.
Brian W. Kernighan 는 컴퓨터 언어 학자이네요
Dennis M. Ritchie 는 컴퓨터 공학자로 UNIX 운영 체제를 위한 C 언어를 만들었
으며
이후 C 언어는 [ANSI C](#) 로 표준 프로그래밍 언어가 되기도 해요.
[UNIX](#) 는 [DEC](#) (Digital Equipment)사에서 [PDP-11](#) 이라는 컴퓨터를 운영하기 위
한 운영 체제로 탄생하게 되요.

멀더

그렇군요.
나도 C 언어가 강력하고 유연한 범용 언어라고 생각합니다. 물론 장점과 단점이
있기에 지금은 입지가 좁아져 가고 있지만요.
하지만 이후에 개발된 언어들에 지대한 영향을 미친 것만은 사실이쥬. 한 시대
를 풍미한 언어라고 확신합니다.

이제 내용을 살펴 볼 때가 된 것 같군요.


```
#include <stdio.h>

void main()
{
    printf("Hello World!\n");
}
```

멀더 이런! 지금 장난해요?

이건 너무 간단한 코드잖아요.

내용이 실망스럽군요.

스컬리 정말 간단한 코드네요. 그리고 가장 기본적인 코드고요.

하지만 여기서 적지 않은 것들을 유추할 수 있어요.

main 이라는 것을 볼 수 있죠? 이건 함수(function)예요.

printf 또한 함수고요.

C = $\sum f(x)$ 라고 USB 에 쓰여 있었다고 했죠?

멀더 네

스컬리 C 언어는 함수로 이루어져 있다는 것을 의미하는 것 같네요.

함수만 다 알아도 C 언어로 프로그래밍 할 수 있다는 거예요.

C 언어는 함수를 통해서 모듈화(Module)화를 할 수 있었는데 그 당시에는 진보적인 방식이었어요.

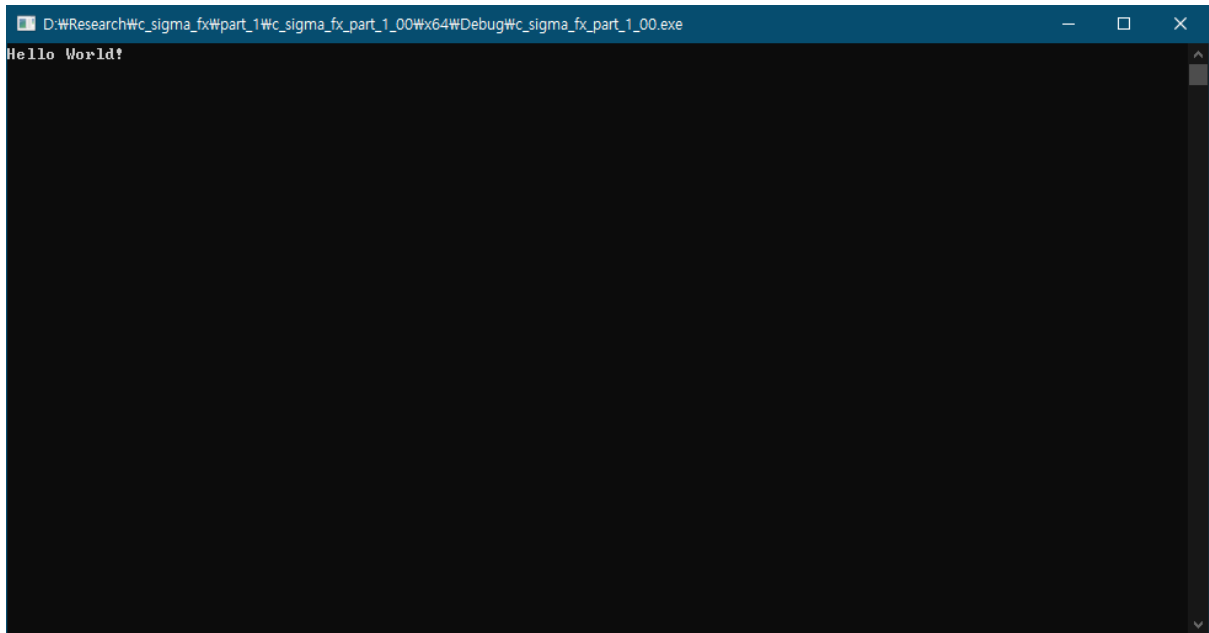
main() 함수는 가장 먼저 실행되는 특수 목적 함수예요.

printf() 함수는 뭔가를 출력하는 함수인데 기본적으로 제공되는 라이브러리(library) 함수고요.

stdio.h 헤더 파일에 선언되어 있는 거예요.

그리고 나머지는 사용자가 임의로 만들어 쓰는 함수(User-defined)가 있는데요
이 작업을 프로그래밍(Programming), 그리고 프로그래밍하는 사람을 프로그래머(Programmer)라고 하죠.

<출력 결과>



멀더 결과가 어이 없네요

 이것도 프로그램인가 싶군요.

스컬리 물론 싱거운 결과이긴 해요.

 하지만 처음 접하는 사람이라면 신기하기도 하답니다.

 결과 화면을 보시면 글자만 있는데요

 Console (콘솔) 또는 Command Prompt 라고 합니다.

 콘솔은 화면 출력뿐만 아니라 키보드를 통한 입력도 가능해요.

 Unix 계열의 운영 체제, DOS 계열의 운영 체제는 콘솔이 기본 입출력 장치이고요.

 Microsoft Windows 나 Apple Macintosh 도 다 콘솔을 지원한답니다.

 요즘은 GUI (Graphic User Interface) 라는 화려한 그래픽 화면 출력을 지원하죠.

 입력 또한 키보드, 마우스, 조이스틱 등등 다양하답니다.

<c_sigma_fx_part_1_01.c>

```
#include <stdio.h>

// external declaration

int      global_data_sum_total = 0;

int      Sum(int a, int b)
{
    global_data_sum_total += (a + b);
    return a + b;
}

void     main(void)
{
    // declarations part

    int    local_data_a = 1;
    int    local_data_b = 2;
    int    local_data_sum;

    // statements part

    printf("%d + %d = %d\\n", local_data_a, local_data_b, local_data_a + local_data_b);

    local_data_sum = Sum(local_data_a, local_data_b);

    printf("sum(%d, %d) = %d\\n", local_data_a, local_data_b, local_data_sum);

    printf("sum_total = %d\\n", global_data_sum_total);
}
```

멀더

이번 것은 이전 것보다 조금은 프로그램 같군요.

어디 봅시다. external declaration, 함수 안에서는 declarations part, statements part 라고 쓰여 있군요.

declarations part 에서는 data 가 보이는 것으로 보아 데이터를 선언하거나 숫자로 설정하는 것 같군요.

statements part 에서는 Sum(), printf() 등의 함수가 보이는 것으로 보아 데이터를 가공한다는 거군요.

어때요? 내 말이 맞죠?

스컬리

네 정확히 맞혔네요.

함수는 declarations part 와 statements part 로 이루어져 있는데요.

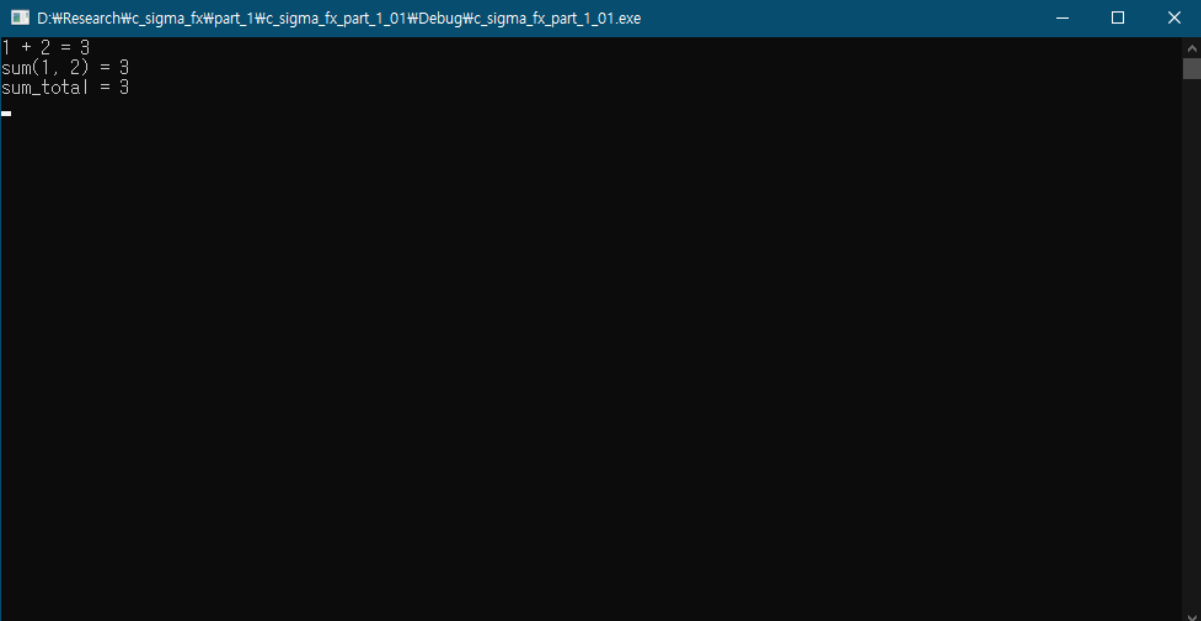
declarations part 에는 변수(데이터)를 선언하는 곳이에요.

statements part 에는 변수를 조작하거나 흐름을 조작하는 곳이고요.

변수를 조작하는 것을 식 (expression) 이라고 하고요

흐름 (flow) 을 조작하는 것을 문 (statement) 이라고 해요.

<출력 결과>



```
D:\Research\wc_sigma_fx\part_1\wc_sigma_fx_part_1_01\Debug\wc_sigma_fx_part_1_01.exe
1 + 2 = 3
sum(1, 2) = 3
sum_total = 3
```

<c_sigma_fx_part_1_02.c>

```
#include <stdio.h>

void    main()
{
    char    c = 'A';
    short   s = 0x1234;
    int     i = 0x12345678;
    __int64 i64 = 0x1234567812345678;
    float    f = 0.1234567f;
    double   d = 0.123456789012345;
    void*    p = 0;

    printf("c    = %-20c, size = %zd byte(s)\n", c, sizeof(c));
    printf("s    = %-20i, size = %zd byte(s)\n", s, sizeof(s));
    printf("i    = %-20d, size = %zd byte(s)\n", i, sizeof(i));
    printf("i64 = %-20l64d, size = %zd byte(s)\n", i64, sizeof(i64));
    printf("f    = %-20.7f, size = %zd byte(s)\n", f, sizeof(f));
    printf("d    = %-20.15f, size = %zd byte(s)\n", d, sizeof(d));
    printf("p    = %-20p, size = %zd byte(s)\n", p, sizeof(p));
}
```

멀더 데이터 유형에 대한 내용이군요.

 보기만 해도 머리가 아파지네요.

 그래도 문자, 정수 그리고 실수 데이터가 있다는 건 파악되는군요.

스컬리 네. 통찰력이 대단하네요, 멀더.

 char 변수 형은 주로 [ASCII 코드](#) 값을 담을 수 있어요.

 short 는 short int 의 약자로 짧은 정수 값을 담을 때 쓰고요.

 int 는 일반적인 정수 값을 담고요.

 float 는 일반적인 실수 값이며, double 은 좀더 정확한 실수 값을 담을 수 있네요.

멀더 __int64 하고 void* 는 뭐죠?

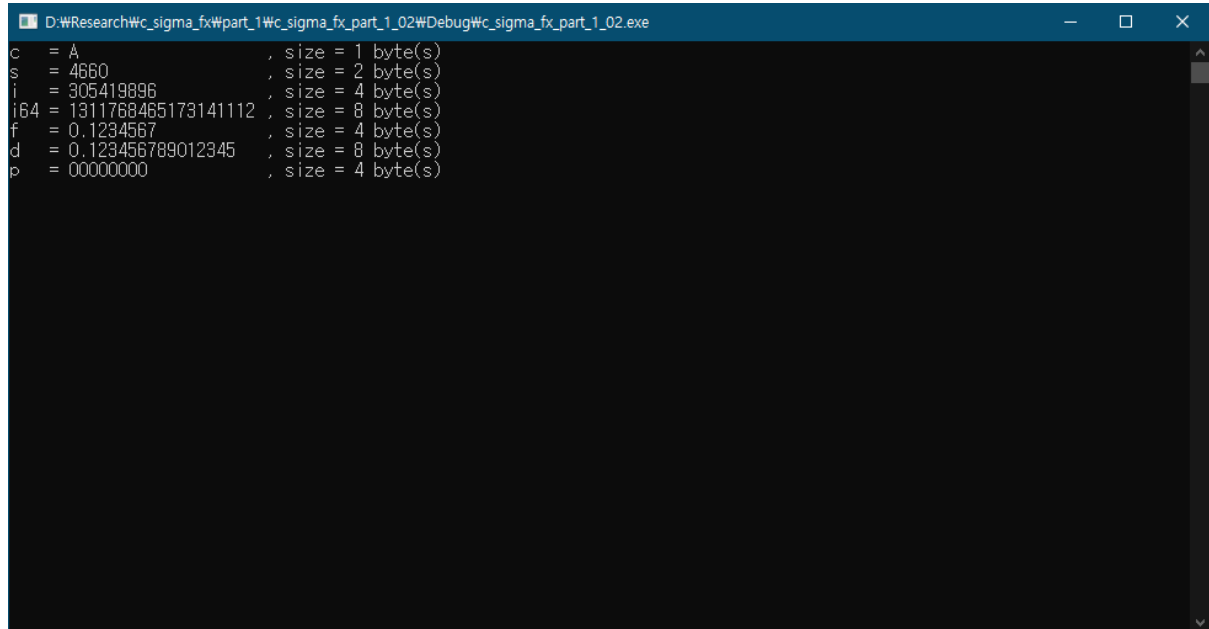
스컬리 음 ...

 __int64 는 64비트 정수형이고요, void* 는 포인터랍니다.

 자세한 건 다음에 알려 드릴게요. 저도 멀더 때문에 머리가 아파지기 시작했어요.

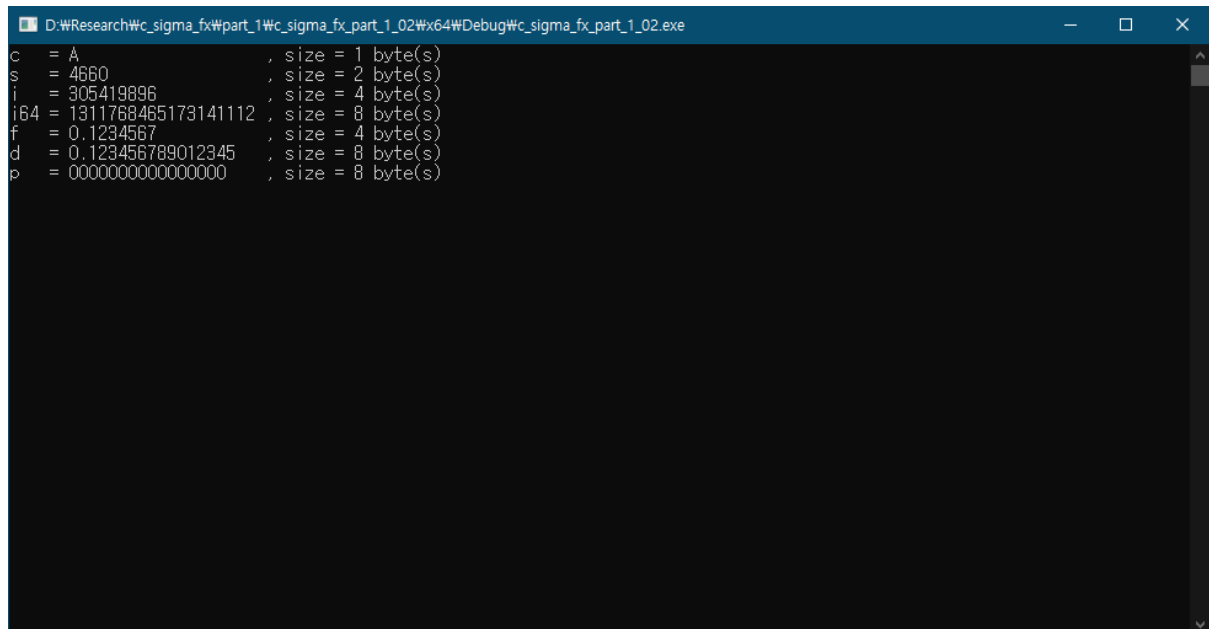
멀더 미안해요 스컬리, 하지만 호기심이 들어서요.
 다음에 꼭 그리고 자세히 알려주셔야 됩니다.

<출력 결과 - x86 (32bit)>



```
D:\Research\c_sigma_fx\part_1\c_sigma_fx_part_1_02\Debug\c_sigma_fx_part_1_02.exe
c = A , size = 1 byte(s)
s = 4660 , size = 2 byte(s)
i = 305419896 , size = 4 byte(s)
i64 = 1311768465173141112 , size = 8 byte(s)
f = 0.1234567 , size = 4 byte(s)
d = 0.123456789012345 , size = 8 byte(s)
p = 00000000 , size = 4 byte(s)
```

<출력 결과 - x64 (64bit)>



```
D:\Research\c_sigma_fx\part_1\c_sigma_fx_part_1_02\x64\Debug\c_sigma_fx_part_1_02.exe
c = A , size = 1 byte(s)
s = 4660 , size = 2 byte(s)
i = 305419896 , size = 4 byte(s)
i64 = 1311768465173141112 , size = 8 byte(s)
f = 0.1234567 , size = 4 byte(s)
d = 0.123456789012345 , size = 8 byte(s)
p = 0000000000000000 , size = 8 byte(s)
```

변수의 기본적인 종류는 다음과 같아요.

Type	Machine	x86 (32 bit)	x64 (64bit)	용도
char		1 byte (8 bit)	1 byte (8 bit)	ASCII 코드
short		2 bytes (16 bit)	2 bytes (16 bit)	정수
int		4 bytes (32 bit)	4 bytes (32 bit)	정수
_int64		8 bytes (64 bit)	8 bytes (64 bit)	정수
float		4 bytes (32 bit)	4 bytes (32 bit)	실수
double		8 bytes (64 bit)	8 bytes (64 bit)	실수
* (pointer)		4 bytes (32 bit)	8 bytes (64 bit)	주소

변수의 수치 범위는 다음과 같아요.

Bytes	singed range	unsigned range
1	-128 ~ 127	0 ~ 255
2	-32,768 ~ 32,767	0 ~ 65,535
4	-2,147,483,648 ~ 2,147,483,647	0 ~ 4,294,967,295
4 (float)	3.4E+/-38 (소수점 7자리)	X
8	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807	0 ~ 18,446,744,073,709,551,615
8 (double)	1.7E+/-308 (소수점 15자리)	X

<c_sigma_fx_part_1_03.c>

```
#include <stdio.h>

void    expression_unary()
{
    int i = 1;
    int* pi = &i;

    printf("[ unary expression ]\n");
    printf("i = %d\n", i);
    printf("&i = %p\n", &i);
    printf("*pi = %d\n", *pi);
    printf("~i = %d\n", ~i);
    printf("!i = %d\n", !i);
    i++;
    printf("i++ = %d\n", i);
    --i;
    printf("--i = %d\n", --i);
}

void    main(void)
{
    expression_unary();
}
```

멀더 expression_unary 함수가 보입니다.

해석하면 단항식 이군요.

단항식이라면 항이 하나라는 건데 맞나요?

스컬리 네~ 맞아요, 멀더.

그 연산자를 단항 연산자라고 하고요.

단항 연산자는 변수 또는 상수 하나에 같이 쓰인다고 할 수 있어요.

& (Ampersand, And) 연산자는 주소 연산자 이고요.

* (Asterisk, Star) 연산자는 간접 참조 연산자 이고요.


~ (Complement) 연산자는 [수학 보수](#) 값을 구해요.

! (Not) 은 부정 연산자로 0 이면 1, 0 이 아니면 0 로 결정이 돼요.

++ (incremental) 연산자는 정수형 변수 앞/뒤에 붙여서 1을 증가시켜요.

-- (decremental) 연산자는 정수형 변수 앞/뒤에 붙여서 1을 감소시켜요.

<출력 결과>



A screenshot of a Windows command prompt window. The title bar at the top reads "D:\Research#c_sigma_fx#part_1#c_sigma_fx_part_1_03#Debug#c_sigma_fx_part_1_03.exe". The window has standard Windows window controls (minimize, maximize, close) on the right. The command prompt shows several lines of assembly code: "[unary expression]", "i = 1", "&i = 004FF630", "*pi = 1", "~i = -2", "!i = 0", "i++ = 2", and "--i = 0". A small cursor is visible on the line following "--i = 0". The rest of the window is empty.

```
D:\Research#c_sigma_fx#part_1#c_sigma_fx_part_1_03#Debug#c_sigma_fx_part_1_03.exe
[ unary expression ]
i = 1
&i = 004FF630
*pi = 1
~i = -2
!i = 0
i++ = 2
--i = 0
-
```

<c_sigma_fx_part_1_04.c>

```
#include <stdio.h>

void    expression_arithmetic()
{
    int i = 1;
    int j = 2;

    printf("[ arithmetic expression ]\n");
    printf("i + j = %d\n", i + j);
    printf("i - j = %d\n", i - j);
    printf("i * j = %d\n", i * j);
    printf("i / j = %d\n", i / j);
    printf("i %% j = %d\n", i % j);
}

void    main(void)
{
    expression_arithmetic();
}
```

멀더 expression_arithmetic 함수가 보입니다.

해석하면 산술식 이군요.

+ 연산자는 더하기고

- 연산자는 빼기고

* 연산자는 곱하기고

/ 연산자는 나누기고

% 연산자는 나머지 연산자입니다.


내가 옛날에 사용했던 거라서 잘 알아요.

스컬리 네. 어려운걸 해냈네요. 호호

참고로 1 / 2 인 경우 모두 정수형이면 실수값 0.5 가 아니라 정수값 0 이라는 것을 알아야 해요.

또 printf 함수에서 %는 특수한 문자라서 % 자체를 표시하려면 %%를 하시면 되요.

<출력 결과>



A screenshot of a Windows command prompt window. The title bar at the top is blue and contains the text "D:\Research#c_sigma_fx#part_1#c_sigma_fx_part_1_04#Debug#c_sigma_fx_part_1_04.exe" along with standard window control buttons (minimize, maximize, close). The command prompt area has a black background with white text. The text displayed is as follows:

```
[ arithmetic expression ]  
+ j = 3  
- j = -1  
* j = 2  
/ j = 0  
% j = 1
```

<c_sigma_fx_part_1_05.c>

```
#include <stdio.h>

void    expression_bit_shift()
{
    int i = 2;


    printf("[ bit shift expression ]\n");
    printf("i << 1 = %d\n", i << 1);
    printf("i >> 1 = %d\n", i >> 1);
}

void    main(void)
{
    expression_bit_shift();
}
```

멀더 expression_bit_shift 함수가 보이는군요.
 비트 이동이라는 건데요. 스컬리 설명 부탁드립니다.

스컬리 그럴게요.
 비트 시프트는 비트 이동이 맞아요.
 쉽게 해석하면 2 를 곱하거나 나누면 되는 거예요.
 2 << 1 이라면 2 * 2 가 되겠네요. 답은 4 이고요.
 2 >> 2 이라면 2 / 2 가 되고요. 답은 1 이에요.
 참고로 2 << 2 면 2 * 2 * 2 이므로 8 이 되어요.

<출력 결과>



```
D:\Research#c_sigma_fx#part_1#c_sigma_fx_part_1_05#Debug#c_sigma_fx_part_1_05.exe
[ bit shift expression ]
i << 1 = 4
i >> 1 = 1
```

<c_sigma_fx_part_1_06.c>

```
#include <stdio.h>


void    expression_relational()
{
    int i = 1;
    int j = 2;

    printf("[ relational expression ]\n");
    printf("i = %d, j = %d\n", i, j);
    printf("i < j = %d\n", i < j);
    printf("i > j = %d\n", i > j);
    printf("i <= j = %d\n", i <= j);
    printf("i >= j = %d\n", i >= j);
    printf("i == j = %d\n", i == j);
    printf("i != j = %d\n", i != j);
}

void    main(void)
{
    expression_relational();
}
```

- 멀더 expression_relational 함수인 것을 보니 관계식이군요.
 i < j 는 i 값이 j 값보다 작은 경우 참이군요. 아니면 거짓이고요.
 i > j 는 i 값이 j 값보다 큰 경우 참이군요. 아니면 거짓이고요.
 i <= j 는 i 값이 j 값보다 작거나 같은 경우 참이군요. 아니면 거짓이고요.
 i >= j 는 i 값이 j 값보다 크거나 같은 경우 참이군요. 아니면 거짓이고요.
 i == j 는 i 값이 j 값과 같은 경우 참이군요. 아니면 거짓이고요.
 i != j 는 i 값이 j 값과 다르면 참이군요. 아니면 거짓이고요.
 머 스컬리가 설명하지 않아도 되겠군요.
- 스컬리 참고 할 만한 것은 있어요.
 참이라는 결과는 정수값 1 이 되고요.
 거짓이라는 결과는 정수값 0 이 된다는 것은 알아 두어야 해요.
- 멀더 그렇군요.

<출력 결과>



```
D:\Research#c_sigma_fx#part_1#c_sigma_fx_part_1_06#Debug#c_sigma_fx_part_1_06.exe
[ relational expression ]
= 1, j = 2
< j = 1
> j = 0
<= j = 1
>= j = 0
== j = 0
!= j = 1
```

<c_sigma_fx_part_1_07.c>

```
#include <stdio.h>

void    expression_bitwise()
{
    int i = 0x3;
    int j = 0x1;

    printf("[ bitwise expression ]\n");
    printf("i = %02X, j = %02X\n", i, j);
    printf("i & j = %02X\n", i & j);
    printf("i | j = %02X\n", i | j);
    printf("i ^ j = %02X\n", i ^ j);
}


void    main(void)
{
    expression_bitwise();
}
```

멀더 expression_bitwise 함수라면 비트식 이라는 건데요.
 이전에 비트 시프트식을 알았잖아요?
 비트에 대해 더 알아야 하나요?

스컬리 그럼요. 비트식은 더 중요하답니다.
 i , j 의 비트값을 비교해야 하네요.
 위의 예에서 처럼
 0x3 은 16진수 값으로, 2진수로 해석하면 11 이고요.
 0x1 은 16진수 값으로, 2진수로 해석하면 01 이고요.
 & (AND) 연산자는 각 비트 위치의 값이 모두 1 이면 1, 다르면 0 이 되어요.
 | (OR) 연산자는 각 비트 위치의 값이 모두 0 이면 0, 아니면 1 이 되어요.
 ^ (Exclusive OR) 연산자는 각 비트 위치의 값이 다르면 1, 같으면 0 이 되어요.
 보시다시피 16진수나 2진수는 프로그래머들에겐 반드시 알아야 할 진법이에요.
 2진수 보다는 16진수를 보다 많이 사용하니 꼭 알아 두셔야 해요.

멀더 스컬리, 대단해요!

<출력 결과>



A screenshot of a Windows command prompt window. The title bar shows the file path: D:\Research\c_sigma_fx\part_1\c_sigma_fx_part_1_07\Debug\c_sigma_fx_part_1_07.exe. The window has standard minimize, maximize, and close buttons. The command prompt displays the following text:

```
[ bitwise expression ]  
i = 03, j = 01  
i & j = 01  
i | j = 03  
i ^ j = 02
```

<c_sigma_fx_part_1_08.c>

```
#include <stdio.h>

void    expression_logical()
{
    int i = 1;
    int j = 0;


    printf("[ logical expression ]\n");
    printf("i = %d, j = %d\n", i, j);
    printf("i && j = %d\n", i && j);
    printf("i || j = %d\n", i || j);
}

void    main(void)
{
    expression_logical();
}
```

멀더 expression_logical 함수군요.
 이건 논리식 이라는 거고요.
 && (AND) 연산자는 i, j 모두 0 이 아니면 참, 나머지인 경우 거짓이군요.
 || (OR) 연산자는 i, j 중 하나라도 0 이 아니면 참, 나머지인 경우 거짓이군요.
 쉽네요. 하하하.

스컬리 네 참 잘했어요.
 논리식은 어렵지 않아요. 그리고 많이 사용하죠.
 이전에 살펴 본 관계식처럼 표현은 1 또는 0 으로 나오고요.

<출력 결과>



```
D:\Research#c_sigma_fx#part_1#c_sigma_fx_part_1_08#Debug#c_sigma_fx_part_1_08.exe
[ logical expression ]
i = 1, j = 0
i && j = 0
i || j = 1
```

<c_sigma_fx_part_1_09.c>

```
#include <stdio.h>

void    expression_conditional()
{
    int i = 1;
    int j = 2;

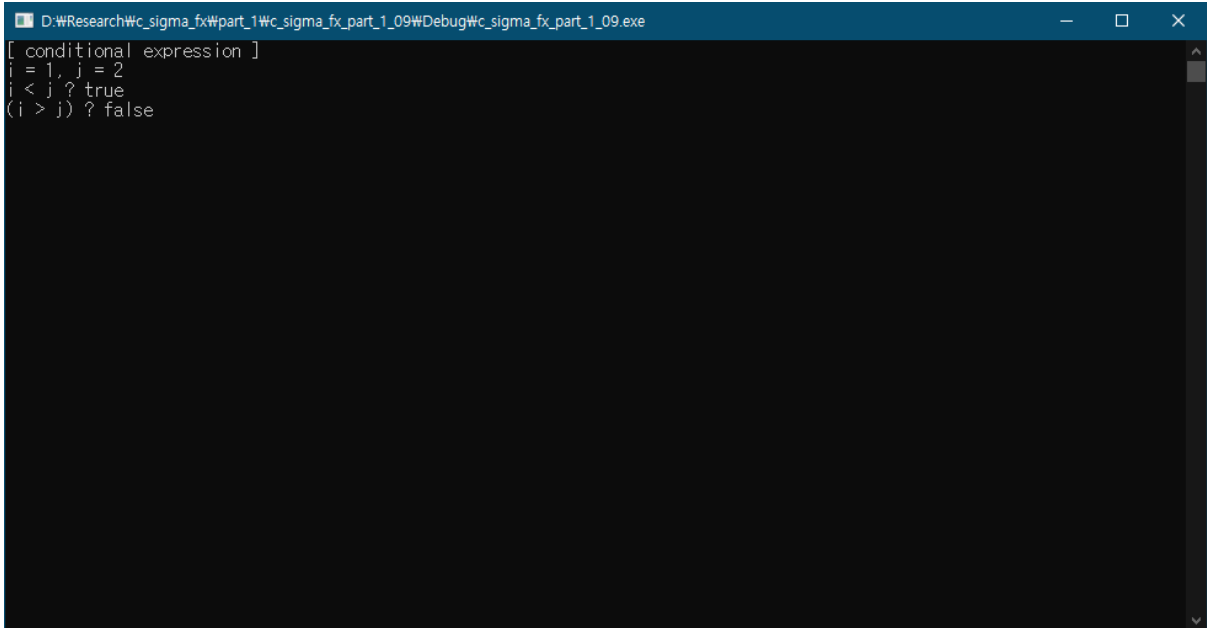
    printf("[ conditional expression ]\n");
    printf("i = %d, j = %d\n", i, j);
    printf("i < j ? %s\n", i < j ? "true" : "false");
    printf("(i > j) ? %s\n", (i > j) ? "true" : "false");
}

void    main(void)
{
    expression_conditional();
}
```

멀더 expression_conditional 함수네요.
조건식이라는 거군요.
어디 봅시다.
i < j ? "true" : "false" 이런 코드가 보이는군요.
1 이 2 보다 작은 건 당연하니까 true 가 출력되겠군요.
밑에 것은 1 이 2 보다 작지 않으니 false 가 출력되겠네요.

스컬리 네, 맞아요.
조건식은 C 언어에서 유일한 3항 연산자로 되어 있어요.
A ? B : C 이런 구조예요 A,B,C 3개의 식이 필요해요.
printf 함수에서 "true" 라는 문자열을 출력하려면 %s 를 사용하면 되고요.

<출력 결과>



```
D:\Research#c_sigma_fx#part_1#c_sigma_fx_part_1_09#Debug#c_sigma_fx_part_1_09.exe
[ conditional expression ]
i = 1, j = 2
i < j ? true
(i > j) ? false
```

<c_sigma_fx_part_1_10.c>

```
#include <stdio.h>

void    expression_assignment()
{
    int i = 1;
    int j = 2;

    printf("[ assignment expression ]\n");
    printf("i = %d, j = %d\n", i, j);
    printf("i *= j = %d\n", i *= j);
    printf("i /= j = %d\n", i /= j);
    printf("i %%= j = %d\n", i %= j);
    printf("i += j = %d\n", i += j);
    printf("i -= j = %d\n", i -= j);
    printf("i <= j = %d\n", i <= j);
    printf("i >= j = %d\n", i >= j);
    printf("i &= j = %d\n", i &= j);
    printf("i ^= j = %d\n", i ^= j);
    printf("i |= j = %d\n", i |= j);
}

void    main(void)
{
    expression_assignment();
}
```

멀더 expression_assignment 함수이므로 할당식이군요.
복잡해 보이지만 해석해봅시다.
= 은 기본 할당식이죠. 이전 설명들에서 많이 썼던 거니까요.
*=, /=, % =, +=, -= 할당식은 산술 할당식이구요.
<=, >= 할당식은 비트 시프트 할당식일테고.
&=, ^=, |= 할당식은 비트 할당식이군요.
이전에 설명했던 것들과 별반 다르지 않군요.

스컬리 그래요.
할당식은 A = B 처럼 A 에 B의 식의 결과를 저장한다는 거예요.
이제 식에 대한 설명은 대부분 된 거 같네요.

멀더. 다 이해하셨나요?
 멀더. 그럴걸요.
 하하하
 스컬리. 흠. 한 번 설명으로는 어렵겠지만 자주 사용하면 쉽게 이해하게 될꺼예요.
 힘네요. 멀더.
 그리고 순차식이라고 있는데요. 그건 나중에 설명 드리죠.
 중요하지 않지만 자주 쓰이기에 분명 이후에 나올 거예요.

<출력 결과>

```

[ assignment expression ]
= j = 2
*= j = 2
/= j = 1
%= j = 1
+= j = 3
-= j = 1
<<= j = 4
>>= j = 1
&= j = 0
^= j = 2
|= j = 2
  
```

식의 종류는 다음과 같아요.

식	연산자	설명
단항식 (unary)	& * + - ~ ! ++ --	
산술식 (arithmetic)	* / % + -	사칙 연산
비트 시프트 (bit shift)	< >	비트단위 이동
관계식 (relational)	< > <= >= == !=	식 간의 관계
비트식	& ^	
논리식 (logical)	&&	
조건식 (conditional)	? :	삼항 연산자
할당식 (assign)	= *= /= %= += -= <<= >>= &= ^= =	
순차식 (enum)	,	열거형

<c_sigma_fx_part_1_11.c>

```
#include <stdio.h>

void    statement_if()
{
    int i = 1;
    int j = 2;
    int max;

    printf("[ statement if ]\n");
    printf("i = %d, j = %d\n", i, j);

    if (i > j) max = i;
    else      max = j;

    if (0 == max)
    {
        i = 0;
        j = 0;
    }
    else if (1 == max)
    {
        i = 1;
        j = 1;
    }
    else
    {
        i = max;
        j = max;
    }

    printf("i = %d, j = %d, max = %d\n", i, j, max);
}

void    main(void)
{
    statement_if();
}
```



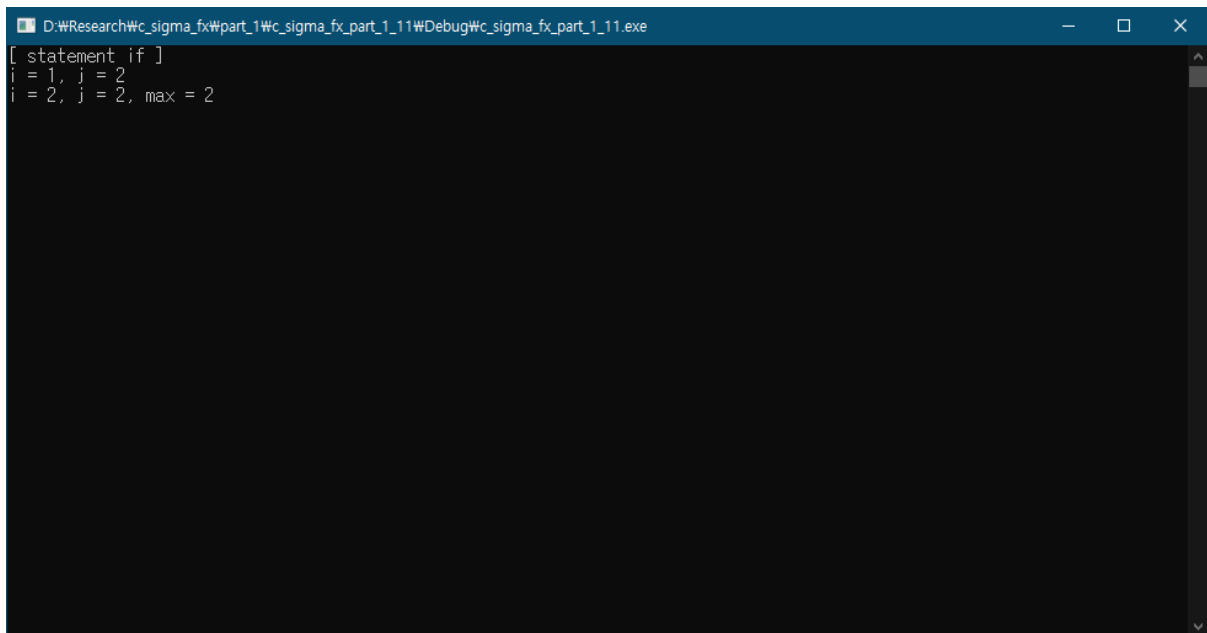
```
}
```

멀더 statement_if 함수가 보이는군요.
이제 식(expression)에서 문(statement)으로 바뀌었어요.
if 문 이라고 해석된다면 조건에 의한 선택문이군요.

스컬리 네 맞아요.
if 문은 대표적인 선택문이에요. 다른 언어에서도 볼 수 있습니다.
이전에도 말씀 드렸드시피 문은 흐름(flow)을 제어한답니다.
if 문은 (expression) 처럼 괄호 안의 식에 의해 그 식의 결과가 참인 경우 바로 밑에 {} 블록 안으로 진입하고요. 거짓인 경우 else {} 블록 안으로 진입해요.
물론 else 가 없는 경우도 있습니다.

멀더 네, 알 것 같습니다.
{ } 블록이란 한 개 이상의 식 또는 문이 있는 경우 사용하겠군요.
한 개의 식 또는 문인 경우는 {} 블록이 필요 없겠죠.

<출력 결과>



```
D:\Research\c_sigma_fx\part_1\c_sigma_fx_part_1_11\Debug\c_sigma_fx_part_1_11.exe
[ statement if ]
i = 1, j = 2
i = 2, j = 2, max = 2
```

<c_sigma_fx_part_1_12.c>

```
#include <stdio.h>

void    statement_switch()
{
    int i = 1;
    int j = 2;
    int max;

    printf("[ statement switch ]\n");
    printf("i = %d, j = %d\n", i, j);

    switch (i)
    {
        case 0: max = 0; break;
        case 1:
            max = 1;
            j = max;
            break;
        case 2:
            {
                max = 2;
                j = max;
            }
            break;
        default:
            max = -1;
            j = 0;
    }

    printf("i = %d, j = %d, max = %d\n", i, j, max);
}

void    main(void)
{
    statement_switch();
}
```

멀더 statement_switch 함수로 보아 switch 문이군요.
 if 문은 조건식이 범위 검사와 같은 관계식이 가능한 반면
 switch 문은 조건식이 일정 값을 판단하고 있군요.
 case 0: 은 조건식 결과가 0 인 경우처럼요.
 default: 는 위에 case 에 걸리지 않을 경우 처리하는 부분이군요.
 맞죠?

스컬리 네, 전에 한번 써본 느낌이 물씬 나네요 멀더.
 switch 문의 조건식은 가급적 결과가 일정 값을 갖도록 하는 경우에 많이 쓰여요.
 또한 조건식의 결과에 따라 해당 case 부분으로 바로 이동한다는 장점이 있고요.
 if 문은 if 문의 조건식을 계속 판단해야 해서 수행시간이 좀 더 걸릴 수 있어요.
 if (1 == A) Block_1
 else if (2 == A) Block_2
 else if (3 == A) Block_3
 else Block_Others
 위에서 A 식의 결과가 4 인 경우 4번의 조건식을 거쳐서 Block_Others 부분을 수행한다는 거예요.

<출력 결과>

```

D:\Research\wc_sigma_fx\part_1\wc_sigma_fx_part_1_12\Debug\wc_sigma_fx_part_1_12.exe
[ statement switch ]
i = 1, j = 2
i = 1, j = 1, max = 1
  
```

<c_sigma_fx_part_1_13.c>

```
#include <stdio.h>

void    statement_while()
{
    int count = 0;
    int sum;

    sum = 0;

    printf("[ statement while ]\n");
    printf("count = %d, sum = %d\n", count, sum);

    while (count < 10)
    {
        count++;
        sum += count;
    }


    printf("count = %d, sum = %d\n", count, sum);
}

void    main(void)
{
    statement_while();
}
```

멀더 statement_while 함수라면 while 반복문 이라는 거군요.
count++;
sum += count;
로 보아 0 부터 9 까지의 합, 아니 1 부터 10 까지의 합을 나타내겠군요.
답은 뻔하지요 55 입니다.

스컬리 네 멋지게 맞추었네요.
while 문은 조건식을 먼저 판단하고 참인 경우 {} 블록을 수행한답니다.
이런 반복문을 선 판단 후 반복 이라고 하고요.

<출력 결과>



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "D:\Research#c_sigma_fx#part_1#c_sigma_fx_part_1_13#Debug#c_sigma_fx_part_1_13.exe". The window has standard minimize, maximize, and close buttons. The main area is black with white text. The output shows three lines: "[statement while.]", "count = 0, sum = 0", and "count = 10, sum = 55". A small white cursor is visible on the line "count = 10, sum = 55". A vertical scrollbar is on the right side of the window.

```
D:\Research#c_sigma_fx#part_1#c_sigma_fx_part_1_13#Debug#c_sigma_fx_part_1_13.exe
[ statement while.]
count = 0, sum = 0
count = 10, sum = 55
```

<c_sigma_fx_part_1_14.c>

```
#include <stdio.h>

void    statement_do_while()
{
    int count = 0;
    int sum;

    sum = 0;

    printf("[ statement do while ]\n");
    printf("count = %d, sum = %d\n", count, sum);

    do {
        count++;
        sum += count;
    } while (count < 10);

    printf("count = %d, sum = %d\n", count, sum);
}

void    main(void)
{
    statement_do_while();
}
```

멀더 statement_do_while 함수라면 do while 반복문 이라는 거군요.


역시 1 부터 10 까지의 합을 나타내겠군요.

스컬리 네 맞아요.

do while 문은 { } 블록을 먼저 수행하고 조건식에 따라 다시 { } 블록을 다시 수행하고요. 즉, { } 블록은 최소 1번은 수행이 된답니다.

이런 반복문을 선 반복 후 판단 이라고 하고요.

<출력 결과>



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "D:\Research#c_sigma_fx#part_1#c_sigma_fx_part_1_14#Debug#c_sigma_fx_part_1_14.exe" along with standard window control buttons (minimize, maximize, close). The command prompt area has a black background with white text. The output displayed is:

```
[ statement do while ]  
count = 0, sum = 0  
count = 10, sum = 55
```

The text is left-aligned. A vertical scrollbar is visible on the right side of the command prompt window.

<c_sigma_fx_part_1_15.c>

```
#include <stdio.h>

void    statement_for()
{
    int sum = 0;

    printf("[ statement for ]\n");
    printf("sum = %d\n", sum);

    for (int count = 1; count <= 10; count++)
    {
        sum += count;
    }

    printf("sum = %d\n", sum);
}

void    main(void)
{
    statement_for();
}
```

멀더 statement_for 함수라면 for 반복문 이겠군요.

역시 1 부터 10 까지의 합을 나타내니다.

스컬리 네 맞고요.

for 문은 for (A; B; C) 에서 처럼 식이 A, B, C 3개가 있어요.

A 식은 주로 초기화 식이고요.

B 식은 조건식 이고요.

C 식은 주로 변화식으로 되어 있어요.


A 식의 int count = 1 이라는 것은 count 정수형 변수의 선언인데요, 밑에 수행 {} 블록 안에서만 유효한 변수랍니다.

즉, 밑에 printf("sum = %d\n", sum) 을 printf("count = %d\n", count) 라고 하면 오류가 발생해요.

멀더, 한 번 수정해서 실행해봐요, 금방 오류가 발생한답니다.

멀더 어! 그렇군요. 컴파일 수행 시에 오류가 발생합니다.

<출력 결과>



A screenshot of a Windows command prompt window. The title bar at the top is blue and contains the text "D:\Research#c_sigma_fx#part_1#c_sigma_fx_part_1_15#Debug#c_sigma_fx_part_1_15.exe" along with standard window control buttons (minimize, maximize, close). The main area of the window is black with white text. The text displayed is: "[statement for]", "sum = 0", and "sum = 55". A vertical scrollbar is visible on the right side of the window.

```
D:\Research#c_sigma_fx#part_1#c_sigma_fx_part_1_15#Debug#c_sigma_fx_part_1_15.exe  
[ statement for ]  
sum = 0  
sum = 55
```

<c_sigma_fx_part_1_16.c>

```
#include <stdio.h>

int      function_sum(int from, int to)
{
    int sum = 0;

    printf("[ function sum ]\n");
    printf("from = %d, to = %d\n", from, to);

    for (int count = from; ; )
    {
        sum += count;
        if (count++ < to) continue;

        goto exit;
    }

exit:
    printf("sum = %d\n", sum);

    return sum;
}

void     main(void)
{
    function_sum(1, 10);
}
```

멀더 function_sum 함수니까 sum 문이군요. 아차!
 이건 sum 함수군요. 느닷없이 함수명이 바뀌었어요.
 from, to 가 정수형인 것을 보면 from 값에서 to 값까지의 합을 수행하는 함수
 가 분명합니다.
 그런데 for (int count = from; ;) 에서 조건식과 변화식이 비어 있군요.
 스컬리 이게 어떻게 된거죠?

스컬리 호호호
 이것이 C 언어의 유연성이예요. for 문의 조건식이 비어 있다는 것은 항상 참이

라는 거예요. 즉, 무한 루프를 만들 때 쓰인답니다.
 변화식 또한 비어있다면 아무런 동작을 안한다는 거고요.
 {} 안에 보시면 if (count++ < to) continue; 라고 보이죠?
 변화식 및 조건식을 블록 안에서도 수행할 수 있거든요.
 또한 그 밑에 goto exit; 블록 밑에 exit: 가 보이죠?
 goto 는 exit: 부분으로 흐름을 이동하는 분기문 이랍니다.
 그런데 goto 문은 쓰지 말아야 해요. 아무 곳으로나 분기하게 되면 이해하기 힘
 들거든요.
 goto exit; 를 break; 로 바꾸고, exit: 를 삭제하고 다시 실행해 보세요. 멀더.
 네, 훨씬 간단해지는군요. 코드를 이해하기도 쉬어집니다.
 goto 문은 안쓸게요. 약속합니다.
 네. 꼭 지키셔야 해요.
 그리고, continue, break, return 과 goto 문은 분기문 이라고 해요.
 자주 쓰이는 것이니 외울 필요도 없네요.

멀더

스컬리

<출력 결과>

```

D:\Research\c_sigma_fx\part_1\c_sigma_fx_part_1_16\Debug\c_sigma_fx_part_1_16.exe
[ function sum ]
from = 1, to = 10
sum = 55
_
  
```

문의 종류는 다음과 같아요.

문	유형	
If (식) 문 If (식) 문 else 문	선택	단일 선택
switch (식) { case 상수1: break;	선택	다중 선택

case 상수2: break; default: }		
for (초기화 식; 판단 식; 변화 식) 문	반복	선 판단 후 반복
while (식) 문	반복	선 판단 후 반복
do { 문 } while (식)	반복	선 반복 후 판단
break continue return goto	분기 쓰지 말 것!	

<c_sigma_fx_part_1_17.c>

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string.h>

#define SEX_MALE      1
#define SEX_FEMALE    2

struct tagRecord
{
    char    szName[256];
    int     nAge;
    int     nSex;

    char    cData[1024];
};

struct tagRecord function_record_set(struct tagRecord aRecord)
{
    aRecord.cData[0] = 'T';
    return aRecord;
}

void    function_record_print(struct tagRecord aRecord)
{
    printf("[Record]\n");
    printf("Name : %s\n", aRecord.szName);
    printf("Age  : %d\n", aRecord.nAge);
    printf("Sex   : %s\n", SEX_MALE == aRecord.nSex ? "Male" : "Female");
    printf("Data : %c\n", aRecord.cData[0]);
}

void    main(void)
{
    struct tagRecord aRecord;
```

```

strcpy(aRecord.szName, "Tom Cruze");
aRecord.nAge = 30;
aRecord.nSex = SEX_MALE;
aRecord.cData[0] = 'M';

function_record_print(aRecord);

function_record_set(aRecord);

function_record_print(aRecord);

aRecord = function_record_set(aRecord);

function_record_print(aRecord);
}

```

- 멀더 struct tagRecord 란 것이 보이는군요.
 이것이 구조체 (Structure) 라는 것이죠?
 구조체는 여러 가지 데이터를 하나로 묶는 (Bundle) 것이고요.
 Function_record_set 을 보니 구조체 데이터를 입력 받아 그 데이터를 조작하고
 구조체 데이터를 돌려주는 것 같군요.
- 스컬리 네, 구조체는 데이터의 집합체라고 할 수 있어요.
 아주 유용하게 쓰이는 것이니 잘 알아 두어야 해요.
 그리고 function_record_set 함수처럼 함수는 기본적인 모듈 (Module) 로써 이
 또한 작은 프로그램이에요. 당연히 입력 – 조작 – 출력 형태를 이루고 있어요.
 입력 부분 (arguments) 은 각종 데이터 형식이 가능하고요.
 조작 부분은 이전에 알아보았듯이 선언부(declarations) – 서술부(statements) 로
 되어 있고요.
 출력 부분 (return value) 은 입력 부분과 마찬가지로 각종 데이터 형식이 가능해
 요. 다만 1개의 데이터 형식만 가능하답니다.
- 멀더 function_record_set(aRecord) 가 있고,
 aRecord = function_record_set(aRecord) 가 있어요.
 어떤 차이가 있을까요?
- 스컬리 function_record_set 은 aRecord.cData[0] = 'T' 처럼 입력 데이터를 조작하는데요
 입력 데이터의 원본을 조작하지는 않아요.
- 멀더 그게 무슨 소리죠? 원본 데이터를 조작하지 않는다고요?
- 스컬리 네, 이런 방식을 [Call by value] 라고 하세요.
 원본 입력 데이터를 복사하고 함수를 호출한다는 거예요.
 함수 안에서는 원본 입력 데이터가 아닌 복사된 입력 데이터를 조작하게 되요.

function_record_set(aRecord) 는 단순히 복사된 입력 데이터만 조작하니 원본 입력 데이터에는 아무런 영향이 없지만,

aRecord = function_record_set(aRecord) 는 복사된 입력 데이터를 조작하고 그 결과를 aRecord 원본 데이터에 다시 복사를 하니 조작된 데이터가 다시 할당되는 방식이에요. 입력 데이터가 구조체나 배열 등 그 사이즈가 큰 데이터라면 데이터를 복사하는데 상당한 시간이 걸릴 거예요.

멀더


내 컴퓨터는 최신 컴퓨터인데도 그럴까요?

스컬리

물론, 상대적인 것이겠죠.

그래서 C 언어에서는 이를 보완하기 위해 포인터 (pointer) 가 등장하게 된답니다.

<출력 결과>



```
D:\Research\c_sigma_fx\part_1\c_sigma_fx_part_1_17\Debug\c_sigma_fx_part_1_17.exe
[Record]
Name : Tom Cruze
Age : 30
Sex : Male
Data : M
[Record]
Name : Tom Cruze
Age : 30
Sex : Male
Data : M
[Record]
Name : Tom Cruze
Age : 30
Sex : Male
Data : T
```

<c_sigma_fx_part_1_18.c>

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string.h>
#include <time.h>

#define SEX_MALE      1
#define SEX_FEMALE    2

struct tagRecord
{
    char    szName[256];
    int     nAge;
    int     nSex;

    char    cData[1024];
};

struct tagRecord function_record_set_call_by_value(struct tagRecord aRecord)
{
    aRecord.cData[0] = 'T';
    return aRecord;
}

void          function_record_set_call_by_reference(struct tagRecord *paRecord)
{
    paRecord->cData[0] = 'T';
}

void          function_record_print(struct tagRecord *paRecord, time_t ltime)
{
    printf("[Record]\n");
    printf("Name : %s\n", paRecord->szName);
    printf("Age  : %d\n", paRecord->nAge);
    printf("Sex   : %s\n", SEX_MALE == paRecord->nSex ? "Male" : "Female");
    printf("Data : %c\n", paRecord->cData[0]);
}
```



```

        printf("Time = %l64d seconds\n", ltime);
    }

#define LOOP_COUNT    100000000

void    main(void)
{
    struct tagRecord aRecord;
    time_t ltimeBegin, ltimeEnd;

    strcpy(aRecord.szName, "Tom Cruze");
    aRecord.nAge = 30;
    aRecord.nSex = SEX_MALE;
    aRecord.cData[0] = 'M';

    function_record_print(&aRecord, 0);

    time(&ltimeBegin);
    for (int i = 0; i < LOOP_COUNT; i++)
    {
        aRecord = function_record_set_call_by_value(aRecord);
    }
    time(&ltimeEnd);

    function_record_print(&aRecord, ltimeEnd - ltimeBegin);

    time(&ltimeBegin);
    for (int i = 0; i < LOOP_COUNT; i++)
    {
        function_record_set_call_by_reference(&aRecord);
    }
    time(&ltimeEnd);

    function_record_print(&aRecord, ltimeEnd - ltimeBegin);
}

```

멀더

function_record_set_call_by_value 함수가 보이네요.

이건 스칼라가 이야기한 것처럼 입력 데이터를 복사하는 기존 형식입니다.

function_record_set_call_by_reference(struct tagRecord *paRecord) 함수가 또 보

이네요.

이것이 바로 포인터를 활용하는 건가요?

스컬리


맞네요. 멀더.

포인터는 데이터의 주소 (address) 를 가리키게 되요.

함수를 호출할 때 데이터의 주소를 넘겨줌으로써 함수 내에서도 원본 데이터를 조작할 수 있게 됩니다.

그리고, 주소만 주고 받기 때문에 처리 시간도 단축하게 만들어요.

<출력 결과>



```
D:\Research#c_sigma_fx#part_1#c_sigma_fx_part_1_18#Debug#c_sigma_fx_part_1_18.exe
[Record]
Name : Tom Cruze
Age : 30
Sex : Male
Data : M
Time = 0 seconds
[Record]
Name : Tom Cruze
Age : 30
Sex : Male
Data : T
Time = 12 seconds
[Record]
Name : Tom Cruze
Age : 30
Sex : Male
Data : T
Time = 2 seconds
```

SAMSUNG

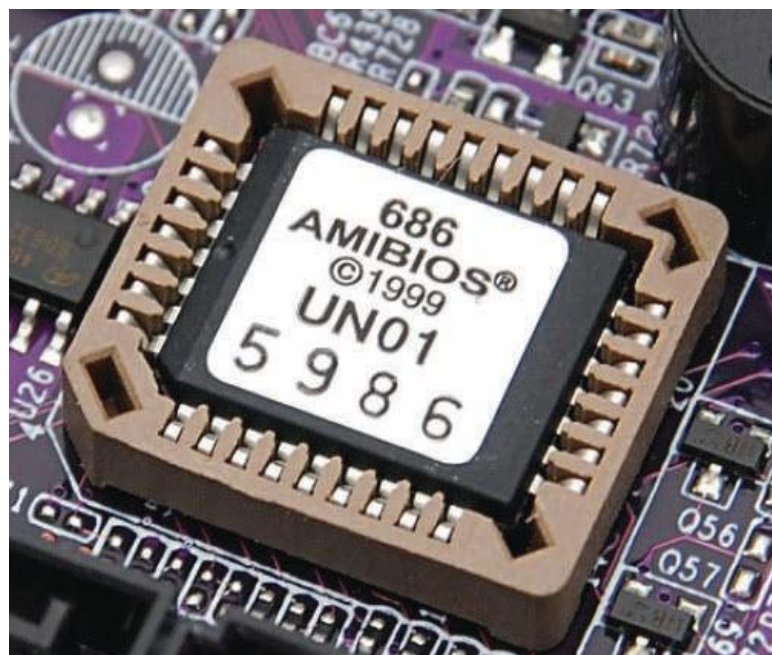
빠르고 쾌적한 PC 환경을 구축하세요.

최신 인텔® 프로세서 호환
끊김 없는 게임 플레이
쾌적한 영상 이미지 소프트웨어 사용 환경
향상된 성능과 안전성



삼성전자 DDR4 16G PC4-21300

최신 CPU와 메인보드에서 지원하는 DDR4 메모리로 빠르고 쾌적한 PC 환경을 구축하세요. DDR4 메모리는 64비트의 빠른 속도로 데이터를 전송하고 높은 대역폭과 용량으로 데이터 인코딩 오류를 최소화합니다. 게임, 디자인 작업 등 어떠한 용도로 사용하더라도 사용자가 만족하는 최적의 PC 성능을 제공하며, 시스템 리소스를 많이 소모하는 고사양 렌더링 프로그램 사용 시에도 끊김 없이 사용자가 원활하게 작업할 수 있습니다.



스컬리

포인터를 설명하려면 컴퓨터의 기억 장치를 알아야 해요.

그 중에서도 주기억 장치를 알아야 해요.

주기억 장치에는 램 (RAM; Random Access Memory) 과 롬 (ROM; Read Only Memory) 으로 구분할 수 있어요.

램은 휘발성 (volatile) 기억 장치로 전원이 끊기면 저장된 데이터가 모두 소실되요.

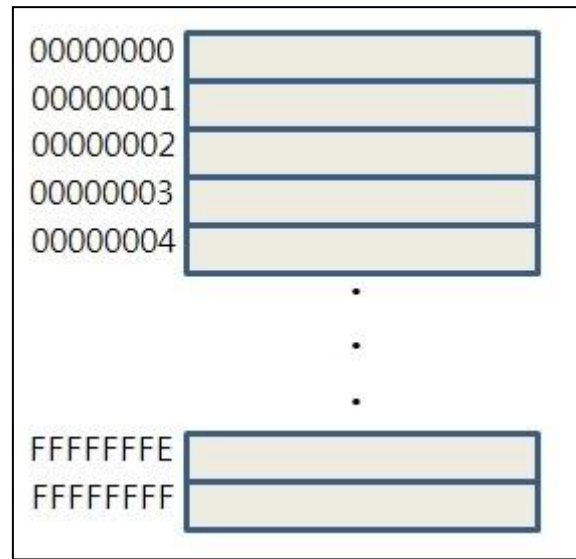
주로 시스템 또는 사용자 프로그램 및 데이터가 저장하게 된답니다.

롬은 비휘발성 (nonvolatile) 기억 장치로 전원이 끊기더라도 저장된 데이터는 남아 있게 되요.

주로 바이오스 (BIOS; Basic Input Output System) 라는 키보드, 마우스와 같은 입력 장치를 사용할 수 있게 하는 프로그램 또는 루틴, 모니터와 같은 출력 장치를 사용할 수 있게 하는 프로그램 또는 루틴이 저장되어 있어요.

[<램에 대한 자세한 정보를 보려면 여기를 클릭하세요>](#)

[<롬에 대한 자세한 정보를 보려면 여기를 클릭하세요>](#)



스컬리

메모리는 데이터를 읽거나 쓸 수 있으므로 그 위치를 알아야 해요.

그 위치를 메모리 주소라고 한답니다.

메모리 주소는 1바이트 단위로 주소가 매겨져 있습니다.

32비트 컴퓨터(x86) 기반은 32비트 주소, 즉 4바이트 주소이고요

64비트 컴퓨터(x64) 기반은 64비트 주소, 즉 8바이트 주소이네요.

그러므로 C 언어에서도 포인터의 값은 32비트 기반은 4바이트 주소를 담고요

64비트 기반은 8바이트 주소를 담는답니다.

[<메모리 주소에 대한 자세한 정보를 보려면 여기를 클릭하세요>](#)

Physical Memory Limits: Windows 10

The following table specifies the limits on physical memory for Windows 10.

Version	Limit on X86	Limit on X64
Windows 10 Enterprise	4 GB	6 TB
Windows 10 Education	4 GB	2 TB
Windows 10 Pro for Workstations	4 GB	6 TB
Windows 10 Pro	4 GB	2 TB
Windows 10 Home	4 GB	128 GB

스컬리

프로그램은 메모리에 적재 (Loading) 되어 CPU 에 의해 실행되는 데요.

적재된 프로그램을 프로세스 (Process) 라고 해요.

프로세스는 코드 메모리 (Code Segment) 와 데이터 메모리 (Data Segment) 로 나뉘게 되요.

또한 데이터 메모리에는 힙 (Heap) 메모리와 스택 (Stack) 메모리로 나뉘어요.
힙 메모리에는 C 언어에서의 함수 밖의 변수 (Global Variable), 즉 전역 변수가
저장되어 있고요.
스택 메모리에는 함수 내에서 선언된 변수 (Local Variable) 가 수시로 저장되었
다가 지워지고 한답니다.

[<프로세스 메모리에 대한 자세한 정보를 보려면 여기를 클릭하세요>](#)

[<힙 메모리에 대한 자세한 정보를 보려면 여기를 클릭하세요>](#)

[<스택 메모리에 대한 자세한 정보를 보려면 여기를 클릭하세요>](#)

<c_sigma_fx_part_1_19.c>

```
#include <stdio.h>

void    function_matrix_identify(int (*pnMatrix)[4])
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (i == j)        pnMatrix[i][j] = 1;
            else                pnMatrix[i][j] = 0;
        }
    }
}

void    main(void)
{
    // Array pointer
    char    cDigit = 'A';
    char*    pcDigit = &cDigit;

    char    szDigits[] = "0123456789";
    char*    pszDigits = szDigits;

    int      nMatrix[4][4];
    int      (*pnMatrix)[4] = nMatrix;

    // Pointer Array
    char    szTextArray0[] = "a";
    char    szTextArray1[] = "ab";
    char    szTextArray2[] = "abc";
    char    szTextArray3[] = "abcd";

    char*    pszTextArray[] = { szTextArray0, szTextArray1, szTextArray2, szTextArray3 };

    printf("[ Array Pointer ]\n");
}
```

```

printf("pcDigit = %c\n", *pcDigit);
printf("pszDigits = %s\n", pszDigits);

function_matrix_identify(pnMatrix);
for (int i = 0; i < 4; i++)
{
    for (int j = 0; j < 4; j++)
    {
        printf("pnMatrix[%d][%d] = %d", i, j, pnMatrix[i][j]);
        if (j < 4 - 1)    printf(", ");
        else                printf("\n");
    }
}

printf("[ Pointer Array ]\n");

printf("szTextArray0 = %s\n", szTextArray0);
printf("szTextArray1 = %s\n", szTextArray1);
printf("szTextArray2 = %s\n", szTextArray2);
printf("szTextArray3 = %s\n", szTextArray3);

for (int i = 0; i < 4; i++)
{
    printf("pszTextArray[%d] = %s\n", i, pszTextArray[i]);
}

}

```

멀더 포인터에 대한 설명, 고맙습니다 스컬리.

스컬리 이해하셨다니 제가 다 뿌듯하네요. 멀더.

 이제 포인터를 얼마나 활용하고 있는 지 알아보까요?

멀더 음...

 * 표시가 훨씬 많아졌군요.

 프로그램이 고급화되었다고 해야 하나 아님 어려워졌다고 해야 하나 싶군요.

스컬리 네, 그러네요. 보다 이해하기 어려워졌어요.

 포인터는 아주 유용하지만 이해하기 어렵게 만드는 단점은 있네요.

 위에 코드를 보면 배열과 포인터에 대한 내용이에요.

우선 배열 (Array) 은 같은 형태의 데이터를 나란히 배열한다는 거예요.

ㅎㅎ

나란히 배열된 것은 a[0] 처럼 [] 안에 첨자로 접근할 수 있어요.

배열의 첫번째 데이터는 [0] 처럼 0 번부터 시작된다는 건 명심해야 해요.

다음은 배열 포인터 (Array Pointer) 에요.

배열의 주소를 지정하는 건데요, 단일 포인터는 1차원 배열 포인터와 같은 의미
이고요, 2차원 이상은 다차원 배열 포인터라고 해요.

사용 방식은 (*ArrayPointer)[][] 인데요 [][] 은 꼭 개수를 명시해야 되요.

다음은 포인터 배열 (Pointer Array) 에요.

포인터 배열은 포인터를 데이터 형으로 갖는 배열이에요.

단일 포인터는 1차원 포인터 배열과 같은 의미예요.

2차원 이상은 다차원 포인터 배열이라고 하고요.

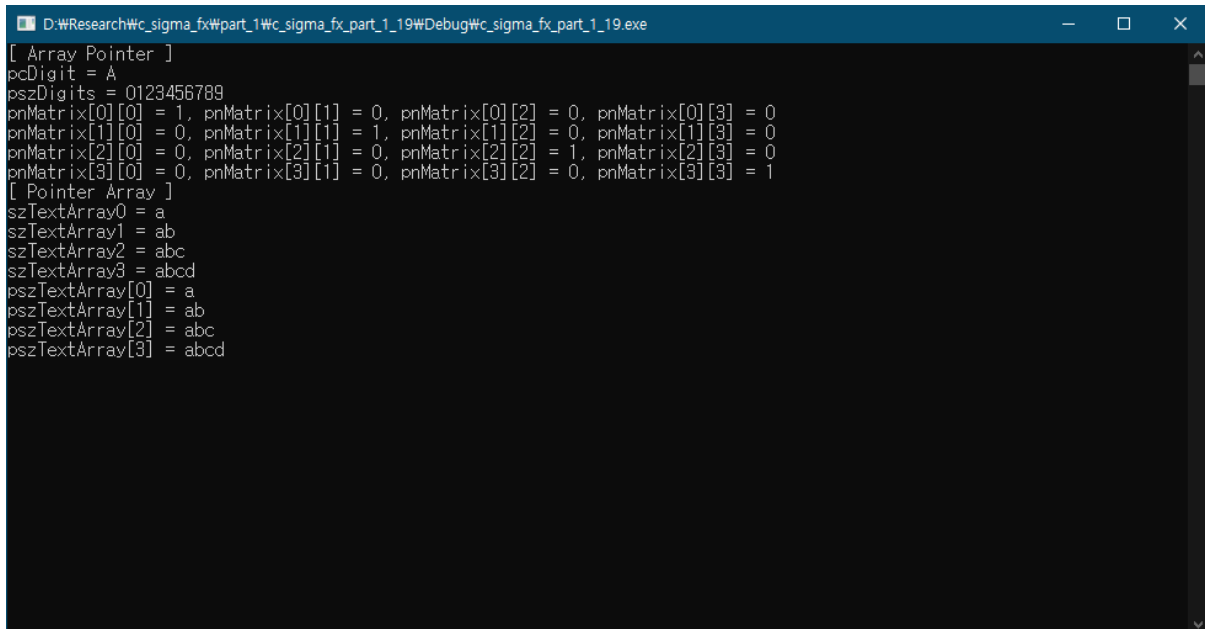
사용 방식은 *PointerArray[][] 인데요 첫번째 [] 는 개수를 명시하지 않아도 되
요.

생각보다 헷갈릴 수 있으니 충분히 이해하셔야 되요. 저도 헷갈리거든요.

배열 포인터는 자주 쓰는 것이라 잘 알아 두셔야 되구요.

포인터 배열은 꼭 필요한 경우에만 사용해야 한답니다.

<출력 결과>



```
D:\Research\c_sigma_fx\part_1\c_sigma_fx_part_1_19\Debug\c_sigma_fx_part_1_19.exe
[ Array Pointer ]
pcDigit = A
pszDigits = 0123456789
pnMatrix[0][0] = 1, pnMatrix[0][1] = 0, pnMatrix[0][2] = 0, pnMatrix[0][3] = 0
pnMatrix[1][0] = 0, pnMatrix[1][1] = 1, pnMatrix[1][2] = 0, pnMatrix[1][3] = 0
pnMatrix[2][0] = 0, pnMatrix[2][1] = 0, pnMatrix[2][2] = 1, pnMatrix[2][3] = 0
pnMatrix[3][0] = 0, pnMatrix[3][1] = 0, pnMatrix[3][2] = 0, pnMatrix[3][3] = 1
[ Pointer Array ]
szTextArray0 = a
szTextArray1 = ab
szTextArray2 = abc
szTextArray3 = abcd
pszTextArray[0] = a
pszTextArray[1] = ab
pszTextArray[2] = abc
pszTextArray[3] = abcd
```

<c_sigma_fx_part_1_20.c>

```
#ifndef __C_SIGMA_FX_PART_1_20__
#define __C_SIGMA_FX_PART_1_20__

#include <stdio.h>

#define LOOP_COUNT    10
#define X10(x)         ((x) * 10)

int    global_loop_count    = LOOP_COUNT;
int global_2x10 = X10(2);

#define FOREVER          for(;;)
#define MAX(a, b)        ((a) > (b) ? (a) : (b))
#define PRINT(expr)      printf(#expr " = %d\\n", expr)

void    main(void)
{
    int    local_loop_count = X10(global_loop_count);
    int local_max = MAX(2, 5);

    PRINT(global_loop_count);
    PRINT(global_2x10);

    PRINT(local_loop_count);
    PRINT(local_max);
}

#endif // !__C_SIGMA_FX_PART_1_20__
```

멀더 오~ 코드가 점점 있어 보여집니다.
 # 이 붙은 라인이 많이 보여집니다.
 #include <stdio.h> 는 printf 등의 라이브러리 함수의 선언을 담고 있는 걸로 알고 있습니다. ♡
 #define LOOP_COUNT 10 은 LOOP_COUNT 가 10 임을 정의한 겁니다.
 #define X10(x) ((x) * 10) 는 x * 10 을 의미하는 것 같군요. 스컬리 맞나요?
스컬리 네, 맞아요.

위에서 알려주고자 하는 것은 매크로 (MACRO) 라는 것 같아요.

매크로는 정의 (MACRO Definition) 와 확장 (MACRO Expansion) 으로 이루어 진
답니다.

#define LOOP_COUNT 10 은 매크로 정의이고요.

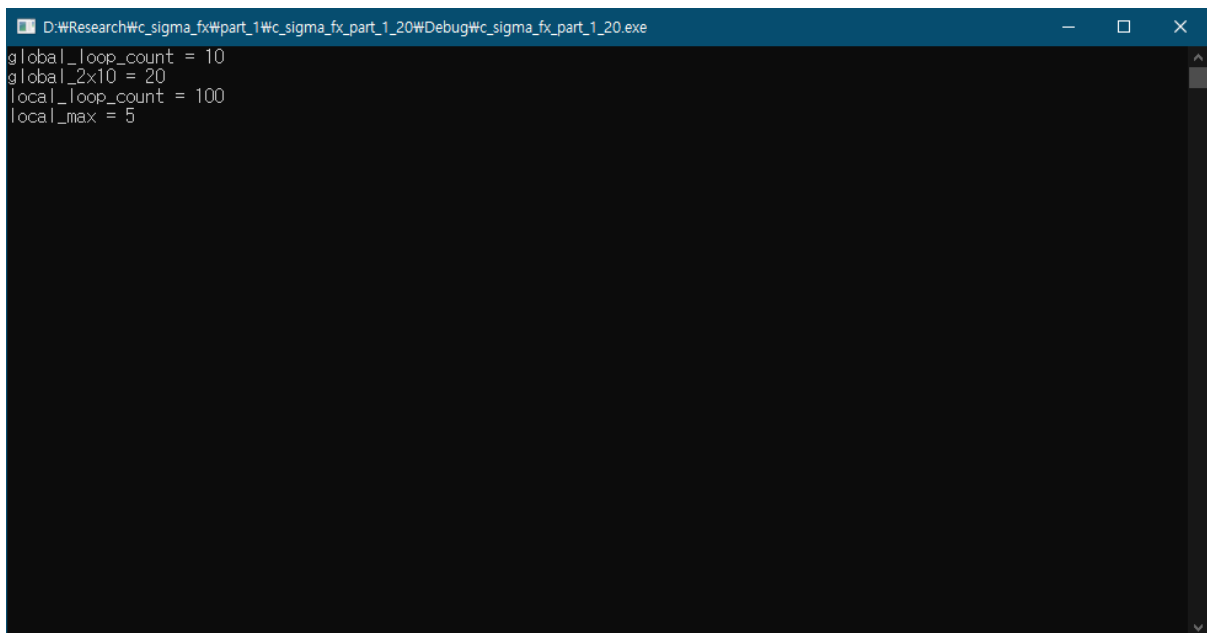
int global_loop_count = LOOP_COUNT; 은 매크로 확장이 되어

int global_loop_count = 10; 으로 대체되어요.

매크로 확장 작업은 컴파일 전에 이루어지고 컴파일은 모든 매크로가 확장된 후
이루어지네요.

으로 시작하는 것을 모두 전처리기 (Preprocessor) 하고 해요.

<출력 결과>



```
D:\Research\wc_sigma_fx\part_1\wc_sigma_fx_part_1_20\Debug\wc_sigma_fx_part_1_20.exe
global_loop_count = 10
global_2x10 = 20
local_loop_count = 100
local_max = 5
```

[<C 언어 전처리를 자세히 알려면 여기를 클릭하세요>](#)

<c_sigma_fx_part_1_21_drinks_00.c>

```
#include <conio.h>
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

////////////////////////////////////////////////////////////////

typedef unsigned int    dword;
typedef unsigned int    udword;
typedef                int    sdword;

typedef unsigned short  uword;
typedef                short  sword;

typedef unsigned char   ubyte;
typedef                char   sbyte;
typedef                char   boolean;

#define true            1
#define false           0
#define null            0

#define RAND(x)         (rand()%(x))

#define SAFE_FREE( x )      { if ( x ) { free( x ); ( x ) = null; } }
#define SAFE_RELEASE( x )   { if ( x ) { ( x )->Release( x ); ( x ) = null; } }
#define SAFE_RELEASE_VT( x ) { if ( x ) { ( x )->lpVtbl->Release( x ); ( x ) = null; } }

////////////////////////////////////////////////////////////////

#define DRINK_NAME_MAX     256

typedef struct tagDrink
{
    char    szName[DRINK_NAME_MAX];
```

```

        int            nPrice;
        int            nProfit;
    } TYPE_DRINK;

TYPE_DRINK*    Drink_New()
{
    TYPE_DRINK*    pTemp = (TYPE_DRINK*)malloc(sizeof(TYPE_DRINK));
    if (null == pTemp)        return null;

    memset(pTemp, 0, sizeof(*pTemp));

    return pTemp;
}

void            Drink_Delete(TYPE_DRINK* pDrink)
{
    SAFE_FREE(pDrink);
}

void    Drink_Init(TYPE_DRINK* pDrink)
{
    if (null == pDrink)        return;

    memset(pDrink->szName, 0, sizeof(pDrink->szName));
    pDrink->nPrice = 0;
    pDrink->nProfit = 0;
}

void    Drink_Set(TYPE_DRINK* pDrink, char* pszName, int nPrice, int nProfit)
{
    if (null == pDrink)        return;

    strcpy_s(pDrink->szName, sizeof(pDrink->szName), pszName);
    pDrink->nPrice = nPrice;
    pDrink->nProfit = nProfit;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```

#define DRINKSLOT_DRINK_MAX      10

typedef struct tagDrinkLink
{
    TYPE_DRINK*                  pDrink;
    struct tagDrinkLink*         pNext;
} TYPE_DRINKLINK;

typedef struct tagDrinkSlot
{
    TYPE_DRINKLINK*              pBottom;
    TYPE_DRINKLINK*              pTop;

    int                          nCount;
} TYPE_DRINKSLOT;

TYPE_DRINKLINK*                DrinkLink_New()
{
    TYPE_DRINKLINK* pTemp = (TYPE_DRINKLINK*)malloc(sizeof(TYPE_DRINKLINK));
    if (null == pTemp)      return null;

    memset(pTemp, 0, sizeof(*pTemp));

    return pTemp;
}

void                            DrinkLink_Delete(TYPE_DRINKLINK* pDrinkLink)
{
    SAFE_FREE(pDrinkLink->pDrink);
    SAFE_FREE(pDrinkLink);
}

void                            DrinkLink_AddDrink(TYPE_DRINKLINK*   pDrinkLink,
    TYPE_DRINK*
    pDrink)
{
    pDrinkLink->pDrink = pDrink;
}

```

```

TYPE_DRINK*      DrinkLink_GetDrink(TYPE_DRINKLINK* pDrinkLink)
{
    return pDrinkLink->pDrink;
}

boolean          DrinkSlot_Init(TYPE_DRINKSLOT* pDrinkSlot)
{
    if (null == pDrinkSlot)    return false;

    pDrinkSlot->pTop = null;
    pDrinkSlot->pBottom = null;

    pDrinkSlot->nCount = 0;

    return true;
}

char*            DrinkSlot_GetDrinkName(TYPE_DRINKSLOT* pDrinkSlot)
{
    TYPE_DRINK* pDrink;

    if (null == pDrinkSlot)    return null;

    pDrink = DrinkLink_GetDrink(pDrinkSlot->pBottom);
    if (null == pDrink)        return null;

    return pDrink->szName;
}

int              DrinkSlot_GetDrinkPrice(TYPE_DRINKSLOT* pDrinkSlot)
{
    TYPE_DRINK* pDrink;

    if (null == pDrinkSlot)    return 0;

    pDrink = DrinkLink_GetDrink(pDrinkSlot->pBottom);
    if (null == pDrink)        return 0;

```

```

        return pDrink->nPrice;
    }

    boolean DrinkSlot_IsEmpty(TYPE_DRINKSLOT* pDrinkSlot)
    {
        if (null == pDrinkSlot)    return true;

        return pDrinkSlot->nCount <= 0;
    }

    boolean DrinkSlot_IsFull(TYPE_DRINKSLOT* pDrinkSlot)
    {
        if (null == pDrinkSlot)    return true;

        return pDrinkSlot->nCount >= DRINKSLOT_DRINK_MAX;
    }

    int      DrinkSlot_GetCount(TYPE_DRINKSLOT* pDrinkSlot)
    {
        if (null == pDrinkSlot)    return 0;

        return pDrinkSlot->nCount;
    }

    void      DrinkSlot_AddDrink(TYPE_DRINKSLOT* pDrinkSlot, TYPE_DRINK* pDrink)
    {
        TYPE_DRINKLINK* pDrinkLink = DrinkLink_New();
        DrinkLink_AddDrink(pDrinkLink, pDrink);

        if (null == pDrinkSlot->pTop)
        {
            pDrinkSlot->pTop = pDrinkLink;
            pDrinkSlot->pBottom = pDrinkLink;
        }
        else
        {
            pDrinkSlot->pTop->pNext = pDrinkLink;
            pDrinkSlot->pTop = pDrinkLink;
        }
    }

```



```

        pDrinkSlot->nCount++;
    }

TYPE_DRINK*    DrinkSlot_OutputDrink(TYPE_DRINKSLOT* pDrinkSlot)
{
    TYPE_DRINKLINK* pDrinkLink = pDrinkSlot->pBottom;
    TYPE_DRINK* pDrink = pDrinkLink->pDrink;

    if (null == pDrinkLink)    return null;

    pDrinkSlot->pBottom = pDrinkLink->pNext;

    pDrinkLink->pDrink = null;
    DrinkLink_Delete(pDrinkLink);

    pDrinkSlot->nCount--;

    return pDrink;
}

/////////////////////////////////////////////////////////////////

#define DRINKSLOT_MAX 4

boolean g_bRun;

int          g_nMoney;

TYPE_DRINKSLOT g_DrinkSlot[DRINKSLOT_MAX];
int          g_nSelecctDrinkSlot;

boolean g_bSelectDrink;
boolean g_bSelectManager;

int          g_nSales;
int          g_nProfit;

int g_nMenu_Main;

```

```
////////////////////////////////////////////////////////////////
```

```
void    DrinksMachine_DrinkSlot_Print(boolean bManagement);
```

```
void    DrinksMachine_SelectDrink();
```

```
void    DrinksMachine_CheckMoney();
```

```
void    DrinksMachine_DispenseDrink();
```

```
void    DrinksMachine_Management();
```

```
void    DrinksMachineManager_Menu_Main();
```

```
void    DrinksMachineManager_Menu_DrinkAdd();
```

```
void    DrinksMachineManager_Menu_Sales();
```

```
void    DrinksMachineManager_DrinkSlot_Print(boolean bManagement);
```

```
////////////////////////////////////////////////////////////////
```

```
void    DrinksMachine_SelectDrink()
```

```
{
```

```
    boolean bSelected;
```

```
    do {
```

```
        int nCh;
```

```
        int nSelectDrinkSlot;
```

```
        printf("==== 음료 선택하세요 =====\n");
```

```
        DrinksMachine_DrinkSlot_Print(true);
```

```
        bSelected = false;
```

```
        g_bSelectDrink = false;
```

```
        g_bSelectManager = false;
```

```
        nCh = _getch();
```

```
        if ('0' == nCh)
```

```
        {
```

```
            bSelected = true;
```

```

        g_bSelectManager = true;
    }
    else
    {
        nSelectDrinkSlot = nCh - '1';

        if (0 <= nSelectDrinkSlot && nSelectDrinkSlot < DRINKSLOT_MAX)
        {
            if (false == DrinkSlot_IsEmpty(&g_DrinkSlot[nSelectDrinkSlot]))
            {
                g_bSelectDrink = true;
                g_nSelectDrinkSlot = nSelectDrinkSlot;

                bSelected = true;
            }
        }
    }

} while (false == bSelected);
}

void DrinksMachine_DrinkSlot_Print(boolean bManagement)
{
    int i;

    if (bManagement) printf("[ 관리자 ] ");

    for (i = 0; i < DRINKSLOT_MAX; i++)
    {
        TYPE_DRINKSLOT* pDrinkSlot = &g_DrinkSlot[i];
        if (DrinkSlot_IsEmpty(pDrinkSlot))
        {
            printf("%d. 음료 없음 ", i + 1);
        }
        else
        {
            printf("%d. %s (%d원) ", i + 1, DrinkSlot_GetDrinkName(pDrinkSlot),
                DrinkSlot_GetDrinkPrice(pDrinkSlot));
        }
    }
}

```

```

        }
    }

    if (bManagement)        printf("0. 관리자₩n");
    else                    printf("₩n");
}

void    DrinksMachine_Management()
{
    boolean bLoop;

    do {
        bLoop = true;

        DrinksMachineManager_Menu_Main();

        switch (g_nMenu_Main)
        {
            case 1:
                DrinksMachineManager_Menu_DrinkAdd();
                break;
            case 2:
                DrinksMachineManager_Menu_Sales();
                break;
            case 3: case 0:
                bLoop = false;
                break;
        }

    } while (bLoop);
}

void    DrinksMachine_CheckMoney()
{
    int nCh;
    boolean bSelected;

    do {
        int nMoney = 0;

```

```

        bSelected = false;

        printf("===== 다음을 선택하세요 =====\n");
        printf("1. 1000원 투입  2. 500원 투입  3. 100원 투입  0. 환불하기\n");

        nCh = _getch();

        switch (nCh)
        {
            case '1': nMoney = 1000; break;
            case '2': nMoney = 500; break;
            case '3': nMoney = 100; break;
        }

        if (nMoney > 0)
        {
            TYPE_DRINKSLOT* pDrinkSlot = &g_DrinkSlot[g_nSelecctDrinkSlot];

            g_nMoney += nMoney;
            if (g_nMoney >= DrinkSlot_GetDrinkPrice(pDrinkSlot))
            {
                bSelected = true;
            }
        }
        else if ('0' == nCh)
        {
            g_nMoney = 0;
            bSelected = true;
        }

        printf("[투입 금액] %d 원\n", g_nMoney);

    } while (false == bSelected);
}

void    DrinksMachine_DispendDrink()
{
    TYPE_DRINKSLOT* pDrinkSlot = &g_DrinkSlot[g_nSelecctDrinkSlot];

```

```

TYPE_DRINK* pDrink = DrinkSlot_OutputDrink(pDrinkSlot);

if (g_nMoney <= 0)      return;

printf("%s (%d원) 가 배출되었습니다~\n", pDrink->szName, pDrink->nPrice);

g_nMoney -= pDrink->nPrice;

g_nSales += pDrink->nPrice;
g_nProfit += pDrink->nProfit;
}

/////////////////////////////////////////////////////////////////

void    DrinksMachineManager_Menu_Main()
{
    boolean bSelected = false;

    do {
        int nCh;

        printf("[ 관리자 ] ===== 메뉴를 선택하세요 =====\n");
        printf("[ 관리자 ] 1. 음료 추가  2. 정산하기  3. 나가기  0. 전원 끄기\n");

        bSelected = false;
        g_nMenu_Main = 0;

        nCh = _getch();

        switch (nCh)
        {
            case '1': case '2': case '3':
                g_nMenu_Main = nCh - '0';
                bSelected = true;
                break;
            case '0':
                g_nMenu_Main = 0;

```

```

        g_bRun = false;
        bSelected = true;
        break;
    }

} while (false == bSelected);
}

void    DrinksMachineManager_Menu_DrinkAdd()
{
    TYPE_DRINKSLOT* pDrinkSlot;

    boolean bSelected = false;

    do {
        int nCh;

        printf("[ 관리자 ] 음료 슬롯을 선택하세요\n");
        DrinksMachineManager_DrinkSlot_Print(false);

        bSelected = false;

        nCh = _getch();

        if ('0' == nCh)
        {
            bSelected = true;
        }
        else if (0 <= nCh - '1' && nCh - '1' < DRINKSLOT_MAX)
        {
            pDrinkSlot = &g_DrinkSlot[nCh - '1'];
            if (DrinkSlot_IsFull(pDrinkSlot))
            {
                printf("슬롯이 꽉 찼습니다 !\n");
                bSelected = true;
            }
            else
            {
                TYPE_DRINK aDrink;

```

```

        Drink_Init(&aDrink);

        printf("[ 관리자 ] 음료 이름을 입력하세요\n");
        scanf_s("%s", aDrink.szName,
(unsigned)_countof(aDrink.szName));

        printf("[ 관리자 ] 음료 가격(원)을 입력하세요\n");
        scanf_s("%d", &aDrink.nPrice);

        printf("[ 관리자 ] 음료 이윤(원)을 입력하세요\n");
        scanf_s("%d", &aDrink.nProfit);

        do {
            printf("[ 관리자 ] 음료를 추가하시겠습니까?\n1.예
2.아니오\n");

            nCh = _getch();

            if ('1' == nCh)
            {
                TYPE_DRINK* pDrink = Drink_New();
                if (null != pDrink)
                {
                    Drink_Set(pDrink, aDrink.szName,
aDrink.nPrice, aDrink.nProfit);
                    DrinkSlot_AddDrink(pDrinkSlot,
pDrink);

                    printf("[ 관리자 ] 음료(총 %d개)가
추가되었습니다.\n", DrinkSlot_GetCount(pDrinkSlot));

                    if (DrinkSlot_IsFull(pDrinkSlot))
                    {
                        printf("[ 관리자 ] 더 이상
추가할 수 없습니다.\n");

                        nCh = '2';
                        bSelected = true;
                    }
                }
            }
        }
    }
}

```



```

        }
        else if ('2' == nCh)
        {
            bSelected = true;
        }

    } while ('2' != nCh);

    }

}

} while (false == bSelected);
}

void    DrinksMachineManager_Menu_Sales()
{
    printf("[ 관리자 ] 정산 정보\n");
    printf("[ 관리자 ] 판매 총액 = %d 원\n", g_nSales);
    printf("[ 관리자 ] 이윤 총액 = %d 원\n", g_nProfit);
}

void    DrinksMachineManager_DrinkSlot_Print(boolean bManagement)
{
    int i;

    if (bManagement)        printf("[ 관리자 ] ");

    for (i = 0; i < DRINKSLOT_MAX; i++)
    {
        TYPE_DRINKSLOT* pDrinkSlot = &g_DrinkSlot[i];
        if (DrinkSlot_IsEmpty(pDrinkSlot))
        {
            printf("%d. 음료 없음 ", i + 1);
        }
        else
        {
            printf("%d. %s (%d원) ", i + 1, DrinkSlot_GetDrinkName(pDrinkSlot),
                DrinkSlot_GetDrinkPrice(pDrinkSlot));
        }
    }
}

```

```

    }

    if (bManagement)        printf("0. 관리자\n");
    else                    printf("\n");
}

////////////////////////////////////

void    main(void)
{
    for (int i = 0; i < DRINKSLOT_MAX; i++)
    {
        DrinkSlot_Init(&g_DrinkSlot[i]);
    }

    printf("음료 자동판매기를 시작합니다.\n");

    g_nMoney = 0;
    g_nSales = 0;
    g_nProfit = 0;

    g_bRun = true;
    do {

        DrinksMachine_SelectDrink();

        if (g_bSelectDrink)
        {
            DrinksMachine_CheckMoney();
            DrinksMachine_DispenseDrink();
        }
        else if (g_bSelectManager)
        {
            DrinksMachine_Management();
        }

    } while (g_bRun);
}

```

```
printf("음료 자동판매기를 중지합니다.\n");
```

```
}
```

멀더

헐~~~

느닷없이 코드가 많아졌군요.

이걸 어떻게 분석하죠?

스컬리

호호호

멀더, 정신차려요

조금만 집중하면 충분히 이해할 수 있어요.

우선

#include <conio.h> 가 보이죠?

이건 콘솔에서 즉 키보드 입력을 받거나 콘솔로 즉 모니터로 출력을 할 수 있는 라이브러리 함수들이 포함되어 있어요. 우리는 포함된 함수들을 편리하게 사용하기만 하면 되요.

나머지 포함된 헤더 파일들도 찾아보면 쉽게 알 수 있어요.

이런 라이브러리 함수들에 대하여 설명한 것을 라이브러리 레퍼런스 가이드 라고 해요. 컴파일러에서 모두 지원한답니다.

다음은

```
typedef unsigned int dword;
```

라고 되어 있는데요. 이건 사용자 데이터 형을 새로 만드는 거예요.

dword 는 기존 unsigned int 형을 재정의 한거라고요.

typedef 는 주로 데이터 형이 길거나 아니면 편리하게 사용하기 위해 쓴답니다.

그리고

```
#define RAND(x) (rand()%(x))
```

인데요, 이건 매크로 함수이예요 매크로 역시 편리하게 사용하기 위해 쓰고요.

위의 내용은 랜덤함수를 편리하게 사용하기 위해 정의 한 것 같아요.

또

```
typedef struct tagDrink
```

```
{
```

```
    char    szName[DRINK_NAME_MAX];
```

```
    int     nPrice;
```

```
    int     nProfit;
```

```
} TYPE_DRINK;
```

역시 TYPE_DRINK 데이터 형을 정의했는데요 위에서처럼 구조체를 정의할 수 도 있어요.

드링크니까 음료라는 데이터 형이네요.

```
TYPE_DRINK*    Drink_New()
{
    TYPE_DRINK*    pTemp = (TYPE_DRINK*)malloc(sizeof(TYPE_DRINK));
    if (null == pTemp)        return null;

    memset(pTemp, 0, sizeof(*pTemp));

    return pTemp;
}
```

malloc 이라는 함수가 보이네요.

이 함수는 동적 할당 (Dynamic Allocation) 하는 함수예요.

메모리 중에 해당 프로세스의 힙 메모리에 실행 중에 할당한다는 거예요.

정말 많이 사용한답니다. 꼭 기억해야 해요.

memset 함수는 데이터를 모두 0으로 초기화 (Initialization) 한답니다.

아래는

```
typedef struct tagDrinkLink
{
    TYPE_DRINK*        pDrink;
    struct tagDrinkLink*    pNext;

} TYPE_DRINKLINK;
```

드링크링크 인데요, 이런 것을 싱글 링크드 리스트 (Single Linked List) 라고 한답니다.

Struct tagDrinkLink* pNext;

에서처럼 현재 데이터에 pNext 는 다음 데이터를 연결하거든요.

참고로 배열은 미리 개수를 정해 놓아야 해서 실행 중에 개수를 늘리거나 줄일 수 없습니다. 하지만 링크드 리스트를 사용하면 실행 중에 개수를 늘리거나 줄일 수 있거든요. 이런 장점으로 인해 많이 사용하는 구조 이네요.

꼭 알아두셔야 해요.

이것만 알아도 위의 코드를 전부 이해할 수 있을 거예요.

힘내세요! 멀더.

그렇군요.

스킬리 설명하느라 수고했어요. 정말 고맙군요.

일단, 컴파일을 하고 실행해 보면 좀더 이해하기 쉬겠군요.

멀더

<출력 결과>

```

선택 D:\Research\c_sigma_fx\part_1\c_sigma_fx_part_1_21_drinks_00\#x64\Debug\c_sigma_fx_part_1_21_drinks_00.exe
음료 자동판매기를 시작합니다.
===== 음료를 선택하세요 =====
[ 관리자 ] 1. 음료 없음 2. 음료 없음 3. 음료 없음 4. 음료 없음 0. 관리자
[ 관리자 ] ===== 메뉴를 선택하세요 =====
[ 관리자 ] 1. 음료 추가 2. 정산하기 3. 나가기 0. 전원 끄기
[ 관리자 ] 음료 종류를 선택하세요
1. 음료 없음 2. 음료 없음 3. 음료 없음 4. 음료 없음
[ 관리자 ] 음료 이름을 입력하세요
Colar
[ 관리자 ] 음료 가격(원)을 입력하세요
1000
[ 관리자 ] 음료 이윤(원)을 입력하세요
100
[ 관리자 ] 음료를 추가하시겠습니까?
1. 예 2. 아니오
[ 관리자 ] 음료(총 1개)가 추가되었습니다.
[ 관리자 ] 음료를 추가하시겠습니까?
1. 예 2. 아니오
[ 관리자 ] 음료(총 2개)가 추가되었습니다.
[ 관리자 ] 음료를 추가하시겠습니까?
1. 예 2. 아니오
[ 관리자 ] 음료(총 3개)가 추가되었습니다.
[ 관리자 ] 음료를 추가하시겠습니까?
1. 예 2. 아니오
[ 관리자 ] ===== 메뉴를 선택하세요 =====
[ 관리자 ] 1. 음료 추가 2. 정산하기 3. 나가기 0. 전원 끄기
===== 음료를 선택하세요 =====
[ 관리자 ] 1. Colar (1000원) 2. 음료 없음 3. 음료 없음 4. 음료 없음 0. 관리자
===== 다음을 선택하세요 =====
1. 1000원 투입 2. 500원 투입 3. 100원 투입 0. 환불하기
[ 투입 금액] 1000 원
Colar (1000원)가 배출되었습니다~
===== 음료를 선택하세요 =====
[ 관리자 ] 1. Colar (1000원) 2. 음료 없음 3. 음료 없음 4. 음료 없음 0. 관리자
[ 관리자 ] ===== 메뉴를 선택하세요 =====
[ 관리자 ] 1. 음료 추가 2. 정산하기 3. 나가기 0. 전원 끄기
음료 자동판매기를 중지합니다.

```

멀더

오~~~

음료 자동 판매기 프로그램 같군요.

관리자로 들어가 음료를 설정한 다음 추가했어요.

음료를 선택하고 금액을 투입하니 음료가 나왔네요.

하하하.

스컬리

네, 그런 것 같네요.

코드가 길어서 그렇지 실행해 보면 어렵지 않아 보이네요.

멀더

코드가 길면 정말 이해하기 어렵군요.

코드를 좀 짧게 하는 방법은 없나요, 스컬리?

스컬리

물론 방법이 있어요.

소스 파일을 여러 개로 나누는 방법이 있어요.

그러면 헤더 파일과 소스 파일 2개씩 만들어야 하는 수고를 해야 되지만요.

그래도 소스 파일 하나에 모든 소스를 넣는 것보다 효율적이라고 볼 수 있어요.

아마도 다음 내용은 소스를 여러 개로 나눈 것일 가능성이 있네요.

기대해 보아요 멀더.

<c_sigma_fx_part_1_21_drinks_01.c>

```
#include <stdio.h>
#include "DrinksMachine.h"
#include "DrinkSlot.h"

////////////////////////////////////

boolean g_bRun;

int          g_nMoney;

TYPE_DRINKSLOT g_DrinkSlot[DRINKSLOT_MAX];
int           g_nSelecctDrinkSlot;

boolean g_bSelectDrink;
boolean g_bSelectManager;

int       g_nSales;
int       g_nProfit;

int g_nMenu_Main;

void    main(void)
{
    for (int i = 0; i < DRINKSLOT_MAX; i++)
    {
        DrinkSlot_Init(&g_DrinkSlot[i]);
    }

    printf("음료 자동판매기를 시작합니다.\n");

    g_nMoney = 0;
    g_nSales = 0;
    g_nProfit = 0;

    g_bRun = true;
    do {
```

```

        DrinksMachine_SelectDrink();

        if (g_bSelectDrink)
        {
            DrinksMachine_CheckMoney();
            DrinksMachine_DispendeDrink();
        }
        else if (g_bSelectManager)
        {
            DrinksMachine_Management();
        }

    } while (g_bRun);

    printf("음료 자동판매기를 중지합니다.\n");
}

```

<Define.h>

```

#ifndef __DEFINE_H__
#define __DEFINE_H__

typedef unsigned int    dword;
typedef unsigned int    udword;
typedef                int    sdword;

typedef unsigned short  uword;
typedef                short  sword;

typedef unsigned char   ubyte;
typedef                char   sbyte;
typedef                char   boolean;

#define true            1
#define false          0
#define null            0

#define RAND(x)        (rand()%(x))

```

```

#define SAFE_FREE( x )                { if ( x ) { free( x ); ( x ) = null; } }
#define SAFE_RELEASE( x )             { if ( x ) { ( x )->Release( x ); ( x ) = null; } }
#define SAFE_RELEASE_VT( x )         { if ( x ) { ( x )->lpVtbl->Release( x ); ( x ) = null; } }

#endif // !_DEFINE_H_

```

<Drink.h>

```

#ifndef __DRINK_H__
#define __DRINK_H__

#include "Define.h"

#define DRINK_NAME_MAX      256

typedef struct tagDrink
{
    char    szName[DRINK_NAME_MAX];
    int     nPrice;
    int     nProfit;
} TYPE_DRINK;

TYPE_DRINK* Drink_New();
void    Drink_Init(TYPE_DRINK* pDrink);
void    Drink_Set(TYPE_DRINK* pDrink, char* pszName, int nPrice, int nProfit);
void    Drink_Delete(TYPE_DRINK* pDrink);

#endif // !_DRINK_H_

```

<Drink.c>

```

#include <malloc.h>
#include <memory.h>
#include <string.h>
#include "Drink.h"

TYPE_DRINK* Drink_New()
{
    TYPE_DRINK* pTemp = (TYPE_DRINK*)malloc(sizeof(TYPE_DRINK));
    if (null == pTemp)    return null;
}

```



```

        memset(pTemp, 0, sizeof(*pTemp));

        return pTemp;
    }

void    Drink_Delete(TYPE_DRINK* pDrink)
{
    SAFE_FREE(pDrink);
}

void    Drink_Init(TYPE_DRINK* pDrink)
{
    if (null == pDrink)        return;

    memset(pDrink->szName, 0, sizeof(pDrink->szName));
    pDrink->nPrice = 0;
    pDrink->nProfit = 0;
}

void    Drink_Set(TYPE_DRINK* pDrink, char* pszName, int nPrice, int nProfit)
{
    if (null == pDrink)        return;

    strcpy_s(pDrink->szName, sizeof(pDrink->szName), pszName);
    pDrink->nPrice = nPrice;
    pDrink->nProfit = nProfit;
}

```

<DrinkLink.h>

```

#ifndef __DRINKLINK_H__
#define __DRINKLINK_H__

#include "Drink.h"

typedef struct tagDrinkLink
{
    TYPE_DRINK* pDrink;
    struct tagDrinkLink* pNext;
}

```

```

} TYPE_DRINKLINK;

TYPE_DRINKLINK* DrinkLink_New();
void                DrinkLink_Delete(TYPE_DRINKLINK* pDrinkLink);
void                DrinkLink_AddDrink(TYPE_DRINKLINK*    pDrinkLink,    TYPE_DRINK*
pDrink);
TYPE_DRINK* DrinkLink_GetDrink(TYPE_DRINKLINK* pDrinkLink);

#endif // !_DRINKLINK_H__

```

<DrinkLink.c>

```

#include <malloc.h>
#include <memory.h>
#include "DrinkLink.h"

TYPE_DRINKLINK* DrinkLink_New()
{
    TYPE_DRINKLINK* pTemp = (TYPE_DRINKLINK*)malloc(sizeof(TYPE_DRINKLINK));
    if (null == pTemp)      return null;

    memset(pTemp, 0, sizeof(*pTemp));

    return pTemp;
}

void                DrinkLink_Delete(TYPE_DRINKLINK* pDrinkLink)
{
    SAFE_FREE(pDrinkLink->pDrink);
    SAFE_FREE(pDrinkLink);
}

void                DrinkLink_AddDrink(TYPE_DRINKLINK*    pDrinkLink,    TYPE_DRINK*
pDrink)
{
    pDrinkLink->pDrink = pDrink;
}

TYPE_DRINK* DrinkLink_GetDrink(TYPE_DRINKLINK* pDrinkLink)

```

```
{
    return pDrinkLink->pDrink;
}
```

<DrinkSlot.h>

```
#ifndef __DRINKSLOT_H__
#define __DRINKSLOT_H__

#include "DrinkLink.h"

#define DRINKSLOT_DRINK_MAX      10

typedef struct tagDrinkSlot
{
    TYPE_DRINKLINK* pBottom;
    TYPE_DRINKLINK* pTop;

    int nCount;
} TYPE_DRINKSLOT;

boolean DrinkSlot_Init(TYPE_DRINKSLOT* pDrinkSlot);
char* DrinkSlot_GetDrinkName(TYPE_DRINKSLOT* pDrinkSlot);
int DrinkSlot_GetDrinkPrice(TYPE_DRINKSLOT* pDrinkSlot);
boolean DrinkSlot_IsEmpty(TYPE_DRINKSLOT* pDrinkSlot);
boolean DrinkSlot_IsFull(TYPE_DRINKSLOT* pDrinkSlot);
int DrinkSlot_GetCount(TYPE_DRINKSLOT* pDrinkSlot);
void DrinkSlot_AddDrink(TYPE_DRINKSLOT* pDrinkSlot, TYPE_DRINK* pDrink);
TYPE_DRINK* DrinkSlot_OutputDrink(TYPE_DRINKSLOT* pDrinkSlot);

#endif // !__DRINKSLOT_H__
```

<DrinkSlot.c>

```
#include <malloc.h>
#include <memory.h>
#include <string.h>
#include "DrinkSlot.h"

boolean DrinkSlot_Init(TYPE_DRINKSLOT* pDrinkSlot)
```

```

{
    if (null == pDrinkSlot)    return false;

    pDrinkSlot->pTop = null;
    pDrinkSlot->pBottom = null;

    pDrinkSlot->nCount = 0;

    return true;
}

char* DrinkSlot_GetDrinkName(TYPE_DRINKSLOT* pDrinkSlot)
{
    TYPE_DRINK* pDrink;

    if (null == pDrinkSlot)    return null;

    pDrink = DrinkLink_GetDrink(pDrinkSlot->pBottom);
    if (null == pDrink)        return null;

    return pDrink->szName;
}

int DrinkSlot_GetDrinkPrice(TYPE_DRINKSLOT* pDrinkSlot)
{
    TYPE_DRINK* pDrink;

    if (null == pDrinkSlot)    return 0;

    pDrink = DrinkLink_GetDrink(pDrinkSlot->pBottom);
    if (null == pDrink)        return 0;

    return pDrink->nPrice;
}

boolean DrinkSlot_IsEmpty(TYPE_DRINKSLOT* pDrinkSlot)
{
    if (null == pDrinkSlot)    return true;

```

```

        return pDrinkSlot->nCount <= 0;
    }

    boolean DrinkSlot_IsFull(TYPE_DRINKSLOT* pDrinkSlot)
    {
        if (null == pDrinkSlot)    return true;

        return pDrinkSlot->nCount >= DRINKSLOT_DRINK_MAX;
    }

    int      DrinkSlot_GetCount(TYPE_DRINKSLOT* pDrinkSlot)
    {
        if (null == pDrinkSlot)    return 0;

        return pDrinkSlot->nCount;
    }

    void      DrinkSlot_AddDrink(TYPE_DRINKSLOT* pDrinkSlot, TYPE_DRINK* pDrink)
    {
        TYPE_DRINKLINK* pDrinkLink = DrinkLink_New();
        DrinkLink_AddDrink(pDrinkLink, pDrink);

        if (null == pDrinkSlot->pTop)
        {
            pDrinkSlot->pTop = pDrinkLink;
            pDrinkSlot->pBottom = pDrinkLink;
        }
        else
        {
            pDrinkSlot->pTop->pNext = pDrinkLink;
            pDrinkSlot->pTop = pDrinkLink;
        }

        pDrinkSlot->nCount++;
    }

    TYPE_DRINK* DrinkSlot_OutputDrink(TYPE_DRINKSLOT* pDrinkSlot)
    {
        TYPE_DRINKLINK* pDrinkLink = pDrinkSlot->pBottom;
    }

```

```

    TYPE_DRINK* pDrink = pDrinkLink->pDrink;

    if (null == pDrinkLink)    return null;

    pDrinkSlot->pBottom = pDrinkLink->pNext;

    pDrinkLink->pDrink = null;
    DrinkLink_Delete(pDrinkLink);

    pDrinkSlot->nCount--;

    return pDrink;
}

```

<DrinksMachine.h>

```

#ifndef __DRINKSMACHINE_H__
#define __DRINKSMACHINE_H__

#include "Define.h"

#define DRINKSLOT_MAX 4

void    DrinksMachine_DrinkSlot_Print(boolean bManagement);

void    DrinksMachine_SelectDrink();
void    DrinksMachine_CheckMoney();
void    DrinksMachine_DispenseDrink();

void    DrinksMachine_Management();

#endif // !__DRINKSMACHINEMANAGER_H__

```

<DrinksMachine.c>

```

#include <conio.h>
#include <malloc.h>
#include <memory.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

```

#include "Drink.h"
#include "DrinkSlot.h"
#include "DrinksMachineManager.h"
#include "DrinksMachine.h"

extern boolean   g_bRun;

extern int       g_nMoney;

extern TYPE_DRINKSLOT  g_DrinkSlot[DRINKSLOT_MAX];
extern int             g_nSelecctDrinkSlot;

extern boolean   g_bSelectDrink;
extern boolean g_bSelectManager;

extern int       g_nSales;
extern int       g_nProfit;

extern int g_nMenu_Main;

void   DrinksMachine_SelectDrink()
{
    boolean bSelected;

    do {
        int nCh;
        int nSelectDrinkSlot;

        printf("==== 음료를 선택하세요 =====\n");

        DrinksMachine_DrinkSlot_Print(true);

        bSelected = false;
        g_bSelectDrink = false;
        g_bSelectManager = false;

        nCh = _getch();

        if ('0' == nCh)

```

```

        {
            bSelected = true;
            g_bSelectManager = true;
        }
        else
        {
            nSelectDrinkSlot = nCh - '1';

            if (0 <= nSelectDrinkSlot && nSelectDrinkSlot < DRINKSLOT_MAX)
            {
                if (false == DrinkSlot_IsEmpty(&g_DrinkSlot[nSelectDrinkSlot]))
                {
                    g_bSelectDrink = true;
                    g_nSeleccctDrinkSlot = nSelectDrinkSlot;

                    bSelected = true;
                }
            }
        }

    } while (false == bSelected);
}

void DrinksMachine_DrinkSlot_Print(boolean bManagement)
{
    int i;

    if (bManagement) printf("[ 관리자 ] ");

    for (i = 0; i < DRINKSLOT_MAX; i++)
    {
        TYPE_DRINKSLOT* pDrinkSlot = &g_DrinkSlot[i];
        if (DrinkSlot_IsEmpty(pDrinkSlot))
        {
            printf("%d. 음료 없음 ", i + 1);
        }
        else
        {

```



```

        printf("%d. %s (%d원)  ", i + 1, DrinkSlot_GetDrinkName(pDrinkSlot),
DrinkSlot_GetDrinkPrice(pDrinkSlot));
    }
}

if (bManagement)        printf("0. 관리자₩n");
else                    printf("₩n");
}

void    DrinksMachine_Management()
{
    boolean bLoop;

    do {
        bLoop = true;

        DrinksMachineManager_Menu_Main();

        switch (g_nMenu_Main)
        {
            case 1:
                DrinksMachineManager_Menu_DrinkAdd();
                break;
            case 2:
                DrinksMachineManager_Menu_Sales();
                break;
            case 3: case 0:
                bLoop = false;
                break;
        }

    } while (bLoop);
}

void    DrinksMachine_CheckMoney()
{
    int nCh;
    boolean bSelected;

```

```

do {
    int nMoney = 0;

    bSelected = false;

    printf("===== 다음을 선택하세요 =====\n");
    printf("1. 1000원 투입  2. 500원 투입  3. 100원 투입  0. 환불하기\n");

    nCh = _getch();

    switch (nCh)
    {
        case '1': nMoney = 1000; break;
        case '2': nMoney = 500; break;
        case '3': nMoney = 100; break;
    }

    if (nMoney > 0)
    {
        TYPE_DRINKSLOT* pDrinkSlot = &g_DrinkSlot[g_nSelecctDrinkSlot];

        g_nMoney += nMoney;
        if (g_nMoney >= DrinkSlot_GetDrinkPrice(pDrinkSlot))
        {
            bSelected = true;
        }
    }
    else if ('0' == nCh)
    {
        g_nMoney = 0;
        bSelected = true;
    }

    printf("[투입 금액] %d 원\n", g_nMoney);

} while (false == bSelected);
}

void DrinksMachine_DispenseDrink()

```

```

{
    TYPE_DRINKSLOT* pDrinkSlot = &g_DrinkSlot[g_nSelecctDrinkSlot];
    TYPE_DRINK* pDrink = DrinkSlot_OutputDrink(pDrinkSlot);

    if (g_nMoney <= 0)        return;

    printf("%s (%d원) 가 배출되었습니다~\n", pDrink->szName, pDrink->nPrice);

    g_nMoney -= pDrink->nPrice;

    g_nSales += pDrink->nPrice;
    g_nProfit += pDrink->nProfit;
}

```

<DrinksMahchineManager.h>

```

#ifndef __DRINKSMACHINEMANAGER_H__
#define __DRINKSMACHINEMANAGER_H__

#include "Define.h"

void    DrinksMachineManager_Menu_Main();
void    DrinksMachineManager_Menu_DrinkAdd();
void    DrinksMachineManager_Menu_Sales();
void    DrinksMachineManager_DrinkSlot_Print(boolean bManagement);

#endif // !__DRINKSMACHINEMANAGER_H__

```

<DrinksMahchineManager.c>

```

#include <conio.h>
#include <malloc.h>
#include <memory.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Drink.h"
#include "DrinkSlot.h"
#include "DrinksMachine.h"
#include "DrinksMachineManager.h"

```

```

extern boolean  g_bRun;

extern int      g_nMoney;

extern TYPE_DRINKSLOT  g_DrinkSlot[DRINKSLOT_MAX];
extern int          g_nSelecctDrinkSlot;

extern boolean  g_bSelectDrink;
extern boolean g_bSelectManager;

extern int      g_nSales;
extern int      g_nProfit;

extern int g_nMenu_Main;

void  DrinksMachineManager_Menu_Main()
{
    boolean bSelected = false;

    do {
        int nCh;

        printf("[ 관리자 ] ===== 메뉴를 선택하세요 =====\n");
        printf("[ 관리자 ] 1. 음료 추가  2. 정산하기  3. 나가기  0. 전원 끄기\n");

        bSelected = false;
        g_nMenu_Main = 0;

        nCh = _getch();

        switch (nCh)
        {
            case '1': case '2': case '3':
                g_nMenu_Main = nCh - '0';
                bSelected = true;
                break;
            case '0':
                g_nMenu_Main = 0;

```

```

        g_bRun = false;
        bSelected = true;
        break;
    }

} while (false == bSelected);
}

void    DrinksMachineManager_Menu_DrinkAdd()
{
    TYPE_DRINKSLOT* pDrinkSlot;

    boolean bSelected = false;

    do {
        int nCh;

        printf("[ 관리자 ] 음료 슬롯을 선택하세요\n");
        DrinksMachineManager_DrinkSlot_Print(false);

        bSelected = false;

        nCh = _getch();

        if ('0' == nCh)
        {
            bSelected = true;
        }
        else if (0 <= nCh - '1' && nCh - '1' < DRINKSLOT_MAX)
        {
            pDrinkSlot = &g_DrinkSlot[nCh - '1'];
            if (DrinkSlot_IsFull(pDrinkSlot))
            {
                printf("슬롯이 꽉 찼습니다 !\n");
                bSelected = true;
            }
            else
            {
                TYPE_DRINK aDrink;

```

```

        Drink_Init(&aDrink);

        printf("[ 관리자 ] 음료 이름을 입력하세요\n");
        scanf_s("%s", aDrink.szName,
(unsigned)_countof(aDrink.szName));

        printf("[ 관리자 ] 음료 가격(원)을 입력하세요\n");
        scanf_s("%d", &aDrink.nPrice);

        printf("[ 관리자 ] 음료 이윤(원)을 입력하세요\n");
        scanf_s("%d", &aDrink.nProfit);

        do {
            printf("[ 관리자 ] 음료를 추가하시겠습니까?\n1.예
2.아니오\n");

            nCh = _getch();

            if ('1' == nCh)
            {
                TYPE_DRINK* pDrink = Drink_New();
                if (null != pDrink)
                {
                    Drink_Set(pDrink, aDrink.szName,
aDrink.nPrice, aDrink.nProfit);
                    DrinkSlot_AddDrink(pDrinkSlot,
pDrink);

                    printf("[ 관리자 ] 음료(총 %d개)가
추가되었습니다.\n", DrinkSlot_GetCount(pDrinkSlot));

                    if (DrinkSlot_IsFull(pDrinkSlot))
                    {
                        printf("[ 관리자 ] 더 이상
추가할 수 없습니다.\n");

                        nCh = '2';
                        bSelected = true;
                    }
                }
            }
        }
    }
}

```

```

        }
        else if ('2' == nCh)
        {
            bSelected = true;
        }

    } while ('2' != nCh);

    }

}

} while (false == bSelected);
}

void    DrinksMachineManager_Menu_Sales()
{
    printf("[ 관리자 ] 정산 정보\n");
    printf("[ 관리자 ] 판매 총액 = %d 원\n", g_nSales);
    printf("[ 관리자 ] 이윤 총액 = %d 원\n", g_nProfit);
}

void    DrinksMachineManager_DrinkSlot_Print(boolean bManagement)
{
    int i;

    if (bManagement)        printf("[ 관리자 ] ");

    for (i = 0; i < DRINKSLOT_MAX; i++)
    {
        TYPE_DRINKSLOT* pDrinkSlot = &g_DrinkSlot[i];
        if (DrinkSlot_IsEmpty(pDrinkSlot))
        {
            printf("%d. 음료 없음 ", i + 1);
        }
        else
        {
            printf("%d. %s (%d원) ", i + 1, DrinkSlot_GetDrinkName(pDrinkSlot),
                DrinkSlot_GetDrinkPrice(pDrinkSlot));
        }
    }
}

```

```

    }

    if (bManagement)        printf("0. 관리자\n");
    else                    printf("\n");
}

```

멀더 스컬리, 당신 말대로 파일이 나누어져 있군요.

파일 별로 보면 그렇게 어렵지는 않아 보이는군요.

스컬리 네, 멀더, 파일이 나누어져 있으니 한결 좋아 보여요.

이런 것 또한 모듈화 (Modularity) 라고 할 수 있어요.

한 번에 큰 건 무엇이든 누구든 처음엔 어렵거든요.

이런 식으로 모듈화되어 있다면 보다 쉽게 분석할 수 있어요.

한 번 볼까요?

Define.h 가 있어요. Typedef 이나 #define 에 의한 정의 들로 이루어져 있네요.

다음은

Drink.h, Drink.c 가 보여요.

헤더 파일에서는 데이터 구조체 형과 함수의 선언 들로 이루어져 있네요.

이건 다른 모듈의 헤더 파일이나 다른 모듈의 소스 파일에서 사용할 수 있도록 하는 거예요.

기존의 #include <stdio.h> 처럼 다른 모듈에서 사용할 수 있어요.

소스 파일에서는 자기 자신의 헤더 파일을 포함하고 헤더 파일에서 선언된 함수 들을 정의 즉, 구현되어 있어요.

다음은

DrinkLink.h, DrinkLink.c 역시 마찬가지고요.

DrinkSlot.h, DrinkSlot.c 또한 마찬가지고요.

DrinksManchine.h, DrinksMachine.c 도 마찬가지고요.

DrinksMachineManager.h, DrinksMachineManager.c 가 마지막이네요.

멀더 파일명에 의미를 두니 보다 쉽게 이해가 되겠군요.

작명 센스가 있어야 되겠어요. 하하하.

스컬리 네, 맞네요. 호호호

<출력 결과>

```

D:\Research\c_sigma_fx\part_1\c_sigma_fx_part_1_21_drinks_01#Debug#c_sigma_fx_part_1_21_drinks_01.exe
음료 자동판매기를 시작합니다.
===== 음료를 선택하세요 =====
[ 관리자 ] 1. 음료 없음 2. 음료 없음 3. 음료 없음 4. 음료 없음 0. 관리자
[ 관리자 ] ===== 메뉴를 선택하세요 =====
[ 관리자 ] 1. 음료 추가 2. 정산하기 3. 나가기 0. 전원 끄기
[ 관리자 ] 음료 슬롯을 선택하세요
1. 음료 없음 2. 음료 없음 3. 음료 없음 4. 음료 없음
[ 관리자 ] 음료 이름을 입력하세요
Cider
[ 관리자 ] 음료 가격(원)을 입력하세요
1000
[ 관리자 ] 음료 이윤(원)을 입력하세요
100
[ 관리자 ] 음료를 추가하시겠습니까?
1. 예 2. 아니오
[ 관리자 ] 음료(총 1개)가 추가되었습니다.
[ 관리자 ] 음료를 추가하시겠습니까?
1. 예 2. 아니오
[ 관리자 ] 음료(총 2개)가 추가되었습니다.
[ 관리자 ] 음료를 추가하시겠습니까?
1. 예 2. 아니오
[ 관리자 ] 음료(총 3개)가 추가되었습니다.
[ 관리자 ] 음료를 추가하시겠습니까?
1. 예 2. 아니오
[ 관리자 ] 음료(총 4개)가 추가되었습니다.
[ 관리자 ] 음료를 추가하시겠습니까?
1. 예 2. 아니오
[ 관리자 ] ===== 메뉴를 선택하세요 =====
[ 관리자 ] 1. 음료 추가 2. 정산하기 3. 나가기 0. 전원 끄기
===== 음료를 선택하세요 =====
[ 관리자 ] 1. Cider (1000원) 2. 음료 없음 3. 음료 없음 4. 음료 없음 0. 관리자
===== 다음을 선택하세요 =====
1. 1000원 투입 2. 500원 투입 3. 100원 투입 0. 환불하기
[ 투입 금액 ] 1000 원
Cider (1000원) 가 배출되었습니다~
===== 음료를 선택하세요 =====
[ 관리자 ] 1. Cider (1000원) 2. 음료 없음 3. 음료 없음 4. 음료 없음 0. 관리자

```

<inflate.c>

```

int ZEXPORT inflate(strm, flush)
z_streamp strm;
int flush;
{
    struct inflate_state FAR *state;
    z_const unsigned char FAR *next;    /* next input */
    unsigned char FAR *put;    /* next output */
    unsigned have, left;        /* available input and output */
    unsigned long hold;         /* bit buffer */
    unsigned bits;              /* bits in bit buffer */
    unsigned in, out;           /* save starting available input and output */
    unsigned copy;              /* number of stored or match bytes to copy */
    unsigned char FAR *from;    /* where to copy match bytes from */
    code here;                  /* current decoding table entry */
    code last;                  /* parent table entry */
    unsigned len;               /* length to copy for repeats, bits to drop */
    int ret;                    /* return code */
#ifdef GUNZIP
    unsigned char hbuf[4];      /* buffer for gzip header crc calculation */
#endif
    static const unsigned short order[19] = /* permutation of code lengths */
        {16, 17, 18, 0, 8, 7, 9, 6, 10, 5, 11, 4, 12, 3, 13, 2, 14, 1, 15};

    if (inflateStateCheck(strm) || strm->next_out == Z_NULL ||
        (strm->next_in == Z_NULL && strm->avail_in != 0))
        return Z_STREAM_ERROR;

    state = (struct inflate_state FAR *)strm->state;
    if (state->mode == TYPE) state->mode = TYPEDO;    /* skip check */
    LOAD();
    in = have;
    out = left;
    ret = Z_OK;
    for (;;)
        switch (state->mode) {
            case HEAD:

```

```

        if (state->wrap == 0) {
            state->mode = TYPEDO;
            break;
        }
        NEEDBITS(16);
#ifdef GUNZIP
        if ((state->wrap & 2) && hold == 0x8b1f) { /* gzip header */
            if (state->wbits == 0)
                state->wbits = 15;
            state->check = crc32(0L, Z_NULL, 0);
            CRC2(state->check, hold);
            INITBITS();
            state->mode = FLAGS;
            break;
        }
        state->flags = 0; /* expect zlib header */
        if (state->head != Z_NULL)
            state->head->done = -1;
        if (!(state->wrap & 1) || /* check if zlib header allowed */
#else
        if (
#endif
            ((BITS(8) << 8) + (hold >> 8)) % 31) {
            strm->msg = (char *)"incorrect header check";
            state->mode = BAD;
            break;
        }
        if (BITS(4) != Z_DEFLATED) {
            strm->msg = (char *)"unknown compression method";
            state->mode = BAD;
            break;
        }
        DROPBITS(4);
        len = BITS(4) + 8;
        if (state->wbits == 0)
            state->wbits = len;
        if (len > 15 || len > state->wbits) {
            strm->msg = (char *)"invalid window size";
            state->mode = BAD;

```

```

        break;
    }
    state->dmax = 1U << len;
    Tracev((stderr, "inflate:   zlib header ok\n"));
    strm->adler = state->check = Adler32(0L, Z_NULL, 0);
    state->mode = hold & 0x200 ? DICTID : TYPE;
    INITBITS();
    break;
#ifdef GUNZIP
    case FLAGS:
        NEEDBITS(16);
        state->flags = (int)(hold);
        if ((state->flags & 0xff) != Z_DEFLATED) {
            strm->msg = (char *)"unknown compression method";
            state->mode = BAD;
            break;
        }
        if (state->flags & 0xe000) {
            strm->msg = (char *)"unknown header flags set";
            state->mode = BAD;
            break;
        }
        if (state->head != Z_NULL)
            state->head->text = (int)((hold >> 8) & 1);
        if ((state->flags & 0x0200) && (state->wrap & 4))
            CRC2(state->check, hold);
        INITBITS();
        state->mode = TIME;
    case TIME:
        NEEDBITS(32);
        if (state->head != Z_NULL)
            state->head->time = hold;
        if ((state->flags & 0x0200) && (state->wrap & 4))
            CRC4(state->check, hold);
        INITBITS();
        state->mode = OS;
    case OS:
        NEEDBITS(16);
        if (state->head != Z_NULL) {

```

```

        state->head->xflags = (int)(hold & 0xff);
        state->head->os = (int)(hold >> 8);
    }
    if ((state->flags & 0x0200) && (state->wrap & 4))
        CRC2(state->check, hold);
    INITBITS();
    state->mode = EXLEN;
case EXLEN:
    if (state->flags & 0x0400) {
        NEEDBITS(16);
        state->length = (unsigned)(hold);
        if (state->head != Z_NULL)
            state->head->extra_len = (unsigned)hold;
        if ((state->flags & 0x0200) && (state->wrap & 4))
            CRC2(state->check, hold);
        INITBITS();
    }
    else if (state->head != Z_NULL)
        state->head->extra = Z_NULL;
    state->mode = EXTRA;
case EXTRA:
    if (state->flags & 0x0400) {
        copy = state->length;
        if (copy > have) copy = have;
        if (copy) {
            if (state->head != Z_NULL &&
                state->head->extra != Z_NULL) {
                len = state->head->extra_len - state->length;
                zmemcpy(state->head->extra + len, next,
                    len + copy > state->head->extra_max ?
                        state->head->extra_max - len : copy);
            }
            if ((state->flags & 0x0200) && (state->wrap & 4))
                state->check = crc32(state->check, next, copy);
            have -= copy;
            next += copy;
            state->length -= copy;
        }
    }
    if (state->length) goto inf_leave;

```

```

    }
    state->length = 0;
    state->mode = NAME;
case NAME:
    if (state->flags & 0x0800) {
        if (have == 0) goto inf_leave;
        copy = 0;
        do {
            len = (unsigned)(next[copy++]);
            if (state->head != Z_NULL &&
                state->head->name != Z_NULL &&
                state->length < state->head->name_max)
                state->head->name[state->length++] = (Bytef)len;
        } while (len && copy < have);
        if ((state->flags & 0x0200) && (state->wrap & 4))
            state->check = crc32(state->check, next, copy);
        have -= copy;
        next += copy;
        if (len) goto inf_leave;
    }
    else if (state->head != Z_NULL)
        state->head->name = Z_NULL;
    state->length = 0;
    state->mode = COMMENT;
case COMMENT:
    if (state->flags & 0x1000) {
        if (have == 0) goto inf_leave;
        copy = 0;
        do {
            len = (unsigned)(next[copy++]);
            if (state->head != Z_NULL &&
                state->head->comment != Z_NULL &&
                state->length < state->head->comm_max)
                state->head->comment[state->length++] = (Bytef)len;
        } while (len && copy < have);
        if ((state->flags & 0x0200) && (state->wrap & 4))
            state->check = crc32(state->check, next, copy);
        have -= copy;
        next += copy;
    }

```

```

        if (len) goto inf_leave;
    }
    else if (state->head != Z_NULL)
        state->head->comment = Z_NULL;
    state->mode = HCRC;
case HCRC:
    if (state->flags & 0x0200) {
        NEEDBITS(16);
        if ((state->wrap & 4) && hold != (state->check & 0xffff)) {
            strm->msg = (char *)"header crc mismatch";
            state->mode = BAD;
            break;
        }
        INITBITS();
    }
    if (state->head != Z_NULL) {
        state->head->hcrc = (int)((state->flags >> 9) & 1);
        state->head->done = 1;
    }
    strm->adler = state->check = crc32(0L, Z_NULL, 0);
    state->mode = TYPE;
    break;
#endif

case DICTID:
    NEEDBITS(32);
    strm->adler = state->check = ZSWAP32(hold);
    INITBITS();
    state->mode = DICT;
case DICT:
    if (state->havedict == 0) {
        RESTORE();
        return Z_NEED_DICT;
    }
    strm->adler = state->check = Adler32(0L, Z_NULL, 0);
    state->mode = TYPE;
case TYPE:
    if (flush == Z_BLOCK || flush == Z_TREES) goto inf_leave;
case TYPEDO:
    if (state->last) {

```

```

        BYTEBITS();
        state->mode = CHECK;
        break;
    }
    NEEDBITS(3);
    state->last = BITS(1);
    DROPBITS(1);
    switch (BITS(2)) {
    case 0:                                /* stored block */
        Tracev((stderr, "inflate:    stored block%s\n",
                state->last ? " (last)" : ""));
        state->mode = STORED;
        break;
    case 1:                                /* fixed block */
        fixedtables(state);
        Tracev((stderr, "inflate:    fixed codes block%s\n",
                state->last ? " (last)" : ""));
        state->mode = LEN_;                 /* decode codes */
        if (flush == Z_TREES) {
            DROPBITS(2);
            goto inf_leave;
        }
        break;
    case 2:                                /* dynamic block */
        Tracev((stderr, "inflate:    dynamic codes block%s\n",
                state->last ? " (last)" : ""));
        state->mode = TABLE;
        break;
    case 3:
        strm->msg = (char *)"invalid block type";
        state->mode = BAD;
    }
    DROPBITS(2);
    break;
case STORED:
    BYTEBITS();                            /* go to byte boundary */
    NEEDBITS(32);
    if ((hold & 0xffff) != ((hold >> 16) ^ 0xffff)) {
        strm->msg = (char *)"invalid stored block lengths";

```



```

        state->mode = BAD;
        break;
    }
    state->length = (unsigned)hold & 0xffff;
    Tracev((stderr, "inflate:        stored length %u\\n",
            state->length));
    INITBITS();
    state->mode = COPY_;
    if (flush == Z_TREES) goto inf_leave;
case COPY_:
    state->mode = COPY;
case COPY:
    copy = state->length;
    if (copy) {
        if (copy > have) copy = have;
        if (copy > left) copy = left;
        if (copy == 0) goto inf_leave;
        zmemcpy(put, next, copy);
        have -= copy;
        next += copy;
        left -= copy;
        put += copy;
        state->length -= copy;
        break;
    }
    Tracev((stderr, "inflate:        stored end\\n"));
    state->mode = TYPE;
    break;
case TABLE:
    NEEDBITS(14);
    state->nlen = BITS(5) + 257;
    DROPBITS(5);
    state->ndist = BITS(5) + 1;
    DROPBITS(5);
    state->ncode = BITS(4) + 4;
    DROPBITS(4);
#ifdef PKZIP_BUG_WORKAROUND
    if (state->nlen > 286 || state->ndist > 30) {
        strm->msg = (char *)"too many length or distance symbols";

```

```

        state->mode = BAD;
        break;
    }
#endif

    Tracev((stderr, "inflate:         table sizes ok\n"));
    state->have = 0;
    state->mode = LENLENS;
case LENLENS:
    while (state->have < state->ncode) {
        NEEDBITS(3);
        state->lens[order[state->have++]] = (unsigned short)BITS(3);
        DROPBITS(3);
    }
    while (state->have < 19)
        state->lens[order[state->have++]] = 0;
    state->next = state->codes;
    state->lencode = (const code FAR *)(state->next);
    state->lenbits = 7;
    ret = inflate_table(CODES, state->lens, 19, &(state->next),
                        &(state->lenbits), state->work);
    if (ret) {
        strm->msg = (char *)"invalid code lengths set";
        state->mode = BAD;
        break;
    }
    Tracev((stderr, "inflate:         code lengths ok\n"));
    state->have = 0;
    state->mode = CODELENS;
case CODELENS:
    while (state->have < state->nlen + state->ndist) {
        for (;;) {
            here = state->lencode[BITS(state->lenbits)];
            if ((unsigned)(here.bits) <= bits) break;
            PULLBYTE();
        }
        if (here.val < 16) {
            DROPBITS(here.bits);
            state->lens[state->have++] = here.val;
        }
    }

```

```

else {
    if (here.val == 16) {
        NEEDBITS(here.bits + 2);
        DROPBITS(here.bits);
        if (state->have == 0) {
            strm->msg = (char *)"invalid bit length repeat";
            state->mode = BAD;
            break;
        }
        len = state->lens[state->have - 1];
        copy = 3 + BITS(2);
        DROPBITS(2);
    }
    else if (here.val == 17) {
        NEEDBITS(here.bits + 3);
        DROPBITS(here.bits);
        len = 0;
        copy = 3 + BITS(3);
        DROPBITS(3);
    }
    else {
        NEEDBITS(here.bits + 7);
        DROPBITS(here.bits);
        len = 0;
        copy = 11 + BITS(7);
        DROPBITS(7);
    }
    if (state->have + copy > state->nlen + state->ndist) {
        strm->msg = (char *)"invalid bit length repeat";
        state->mode = BAD;
        break;
    }
    while (copy--)
        state->lens[state->have++] = (unsigned short)len;
}

/* handle error breaks in while */
if (state->mode == BAD) break;

```

```

/* check for end-of-block code (better have one) */
if (state->lens[256] == 0) {
    strm->msg = (char *)"invalid code -- missing end-of-block";
    state->mode = BAD;
    break;
}

/* build code tables -- note: do not change the lenbits or distbits
   values here (9 and 6) without reading the comments in inftrees.h
   concerning the ENOUGH constants, which depend on those values */
state->next = state->codes;
state->lencode = (const code FAR *)(state->next);
state->lenbits = 9;
ret = inflate_table(LENS, state->lens, state->nlen, &(state->next),
                    &(state->lenbits), state->work);
if (ret) {
    strm->msg = (char *)"invalid literal/lengths set";
    state->mode = BAD;
    break;
}
state->distcode = (const code FAR *)(state->next);
state->distbits = 6;
ret = inflate_table(DISTS, state->lens + state->nlen, state->ndist,
                    &(state->next), &(state->distbits), state->work);
if (ret) {
    strm->msg = (char *)"invalid distances set";
    state->mode = BAD;
    break;
}
Tracev((stderr, "inflate:         codes ok\n"));
state->mode = LEN_;
if (flush == Z_TREES) goto inf_leave;
case LEN_:
    state->mode = LEN;
case LEN:
    if (have >= 6 && left >= 258) {
        RESTORE();
        inflate_fast(strm, out);
    }

```

```

        LOAD();
        if (state->mode == TYPE)
            state->back = -1;
        break;
    }
    state->back = 0;
    for (;;) {
        here = state->lencode[BITS(state->lenbits)];
        if ((unsigned)(here.bits) <= bits) break;
        PULLBYTE();
    }
    if (here.op && (here.op & 0xf0) == 0) {
        last = here;
        for (;;) {
            here = state->lencode[last.val +
                                (BITS(last.bits + last.op) >> last.bits)];
            if ((unsigned)(last.bits + here.bits) <= bits) break;
            PULLBYTE();
        }
        DROPBITS(last.bits);
        state->back += last.bits;
    }
    DROPBITS(here.bits);
    state->back += here.bits;
    state->length = (unsigned)here.val;
    if ((int)(here.op) == 0) {
        Tracevv((stderr, here.val >= 0x20 && here.val < 0x7f ?
                "inflate:         literal '%c'\n" :
                "inflate:         literal 0x%02x\n", here.val));
        state->mode = LIT;
        break;
    }
    if (here.op & 32) {
        Tracevv((stderr, "inflate:         end of block\n"));
        state->back = -1;
        state->mode = TYPE;
        break;
    }
    if (here.op & 64) {

```

```

        strm->msg = (char *)"invalid literal/length code";
        state->mode = BAD;
        break;
    }
    state->extra = (unsigned)(here.op) & 15;
    state->mode = LENEXT;
case LENEXT:
    if (state->extra) {
        NEEDBITS(state->extra);
        state->length += BITS(state->extra);
        DROPBITS(state->extra);
        state->back += state->extra;
    }
    Tracevv((stderr, "inflate:         length %u\n", state->length));
    state->was = state->length;
    state->mode = DIST;
case DIST:
    for (;;) {
        here = state->distcode[BITS(state->distbits)];
        if ((unsigned)(here.bits) <= bits) break;
        PULLBYTE();
    }
    if ((here.op & 0xf0) == 0) {
        last = here;
        for (;;) {
            here = state->distcode[last.val +
                                (BITS(last.bits + last.op) >> last.bits)];
            if ((unsigned)(last.bits + here.bits) <= bits) break;
            PULLBYTE();
        }
        DROPBITS(last.bits);
        state->back += last.bits;
    }
    DROPBITS(here.bits);
    state->back += here.bits;
    if (here.op & 64) {
        strm->msg = (char *)"invalid distance code";
        state->mode = BAD;
        break;
    }

```

```

    }
    state->offset = (unsigned)here.val;
    state->extra = (unsigned)(here.op) & 15;
    state->mode = DISTEXT;
case DISTEXT:
    if (state->extra) {
        NEEDBITS(state->extra);
        state->offset += BITS(state->extra);
        DROPBITS(state->extra);
        state->back += state->extra;
    }
#ifdef INFLATE_STRICT
    if (state->offset > state->dmax) {
        strm->msg = (char *)"invalid distance too far back";
        state->mode = BAD;
        break;
    }
#endif

    Tracevv((stderr, "inflate:         distance %u\n", state->offset));
    state->mode = MATCH;
case MATCH:
    if (left == 0) goto inf_leave;
    copy = out - left;
    if (state->offset > copy) {          /* copy from window */
        copy = state->offset - copy;
        if (copy > state->whave) {
            if (state->sane) {
                strm->msg = (char *)"invalid distance too far back";
                state->mode = BAD;
                break;
            }
        }
    }
#ifdef INFLATE_ALLOW_INVALID_DISTANCE_TOOFAR_ARRR
    Trace((stderr, "inflate.c too far\n"));
    copy -= state->whave;
    if (copy > state->length) copy = state->length;
    if (copy > left) copy = left;
    left -= copy;
    state->length -= copy;
    do {

```

```

        *put++ = 0;
    } while (--copy);
    if (state->length == 0) state->mode = LEN;
    break;
#endif

    }
    if (copy > state->wnext) {
        copy -= state->wnext;
        from = state->window + (state->wsize - copy);
    }
    else
        from = state->window + (state->wnext - copy);
    if (copy > state->length) copy = state->length;
}
else { /* copy from output */
    from = put - state->offset;
    copy = state->length;
}
if (copy > left) copy = left;
left -= copy;
state->length -= copy;
do {
    *put++ = *from++;
} while (--copy);
if (state->length == 0) state->mode = LEN;
break;
case LIT:
    if (left == 0) goto inf_leave;
    *put++ = (unsigned char)(state->length);
    left--;
    state->mode = LEN;
    break;
case CHECK:
    if (state->wrap) {
        NEEDBITS(32);
        out -= left;
        strm->total_out += out;
        state->total += out;
        if ((state->wrap & 4) && out)

```



```

        strm->adler = state->check =
            UPDATE(state->check, put - out, out);
        out = left;
        if ((state->wrap & 4) && (
#ifdef GUNZIP
            state->flags ? hold :
#endif
            ZSWAP32(hold)) != state->check) {
            strm->msg = (char *)"incorrect data check";
            state->mode = BAD;
            break;
        }
        INITBITS();
        Tracev((stderr, "inflate:   check matches trailer\n"));
    }
#ifdef GUNZIP
    state->mode = LENGTH;
case LENGTH:
    if (state->wrap && state->flags) {
        NEEDBITS(32);
        if (hold != (state->total & 0xffffffffUL)) {
            strm->msg = (char *)"incorrect length check";
            state->mode = BAD;
            break;
        }
        INITBITS();
        Tracev((stderr, "inflate:   length matches trailer\n"));
    }
#endif
    state->mode = DONE;
case DONE:
    ret = Z_STREAM_END;
    goto inf_leave;
case BAD:
    ret = Z_DATA_ERROR;
    goto inf_leave;
case MEM:
    return Z_MEM_ERROR;
case SYNC:

```

```

        default:
            return Z_STREAM_ERROR;
    }

    /*
     * Return from inflate(), updating the total counts and the check value.
     * If there was no progress during the inflate() call, return a buffer
     * error.  Call updatewindow() to create and/or update the window state.
     * Note: a memory error from inflate() is non-recoverable.
     */
    inf_leave:
        RESTORE();
        if (state->wsize || (out != strm->avail_out && state->mode < BAD &&
            (state->mode < CHECK || flush != Z_FINISH)))
            if (updatewindow(strm, strm->next_out, out - strm->avail_out)) {
                state->mode = MEM;
                return Z_MEM_ERROR;
            }
        in -= strm->avail_in;
        out -= strm->avail_out;
        strm->total_in += in;
        strm->total_out += out;
        state->total += out;
        if ((state->wrap & 4) && out)
            strm->adler = state->check =
                UPDATE(state->check, strm->next_out - out, out);
        strm->data_type = (int)state->bits + (state->last ? 64 : 0) +
            (state->mode == TYPE ? 128 : 0) +
            (state->mode == LEN_ || state->mode == COPY_ ? 256 : 0);
        if (((in == 0 && out == 0) || flush == Z_FINISH) && ret == Z_OK)
            ret = Z_BUF_ERROR;
        return ret;
    }

```

멀더

어~ 이 코드는 뭔가요?

제목에는 gzip 이라고 쓰여 있군요.

gzip 은 가장 보편화된 압축 프로그램인데요

맞죠 스컬리?

스컬리

네, gzip 은 가장 효율적인 압축 알고리즘을 사용하고 있는데요.

그 소스의 일부인거 같네요.
 그것도 inflate 라는 함수 하나의 코드이예요.

멀더 함수 하나의 소스 코드라고요?
 그럼 전체 코드는 어떻다는 거죠?
 분석 자체가 안되겠군요. 저는 여기서 포기하고 싶군요.

스컬리 멀더, 그렇다고 포기하면 안되요. 여기까지 왔잖아요.
 위의 코드가 복잡하고 분석하기 힘들어 보이는 건 사실이예요.
 이런 코드를 **스파게티 코드 (Spaghetti code)** 라고 할 수 있어요.
 스파게티 코드란 쉽게는 알아보기 힘든 코드라고 보면 되요.
 그 밖에 이전에서도 가끔 나온 잘 사용하지 않는 방식 또는 사용하지 말아야 하는 방식을 사용하는 경우를 말하기도 하죠.
 위 코드는 모듈화가 잘 되어 있지 않은 방식이죠.
 또, 매크로 함수를 많이 써서 쉽게 이해되지 않는 부분도 있고요.
 그리고, ...

멀더 잠깐만요, 스컬리.
 그럼 스파게티 코드를 그냥 보고 분석해야 한다는 건가요?
 이런 코드를 해결할 수 있는 방법은 없나요?

스컬리 음 ...
 C 언어에서는 사실 해결하기가 쉽지는 않아요.
 그래서 C++ 이나 Java 같은 객체 지향 언어가 태어나게 되었네요.

멀더 그럼 C 언어는 더 이상 사용할 이유가 없는 거군요.
 C++ 이나 Java 언어를 배우는게 낫겠군요.

스컬리 객체 지향 언어 대부분이 C 언어에 기반을 두고 있어요.
 그래서 C 언어를 먼저 배우면 객체 지향 언어를 배우는 데 도움이 되거든요.

멀더 아~ 머리가 아파 옵니다. 조금은 쉬었다 가야겠군요.
 스컬리도 쉬어야 할 거 같군요.

스컬리 그래요. 조금은 쉬는 것도 나쁜지 않은 생각이예요.

<출력 결과>

<gzip을 컴파일하기 위해서는 DDK 등의 플랫폼 라이브러리를 설치해야 합니다>
 <[gzip](#) 을 클릭하여 컴파일 가이드를 참고하기 바랍니다>

#Scene 3 스컬리 & 멀더의 탐정 사무소

<스컬리와 멀더가 사무소를 나가려고 하는데 사무소 전화기가 울린다>

멀더 나가려는데 전화가 오는군.
 여보세요? 스컬리 & 멀더 탐정 사무소입니다.

부국장 날세 멀더.
 잘 지내고 있나?

멀더 부국장님. 오랜만입니다.
 저는 잘 지내고 있습니다.

부국장 그런데 말일세, 요즘 골치 아픈 일을 겪고 있지 않나?
 무슨 USB 안에 있는 내용을 분석하고 있다는 것 같은데.

멀더 그걸 어떻게 알고 계십니까?
 소문이 그렇게 빠를 것 같지는 않은 것 같은데요.

부국장 사실은 담배 피는 남자가 자네를 보자고 연락이 왔네만.
 무슨 골치 아픈 일을 해결하고 싶다면 자신을 만나 보는게 좋을거라고 하는구만.
 USB 에 관해서 말이지.

멀더 그렇군요. 일이 어떻게 돌아가는지 알 것 같습니다.
 어디로 가야 그 자를 만날 수 있을 까요?

부국장 글쎄, 장소는 이야기 하지 않았네만, 충분히 자네는 알거라고만 했네.
 문제의 시발점이라고 들은 것 같네.

멀더 음 ...
 알 것 같습니다. 부국장님. 전화 주셔서 감사합니다.
 이만 나가봐야 할 것 같습니다.

부국장 멀더, 다음에 또 연락하지.

<멀더는 전화를 끊고 잠깐 고민에 잠긴다>

스컬리 부국장님이 뭐라고 하던가요?

멀더 지금 닥친 문제를 해결할 수 있을 것 같군요.
 당장 담배 피는 남자를 만나러 가야 합니다.

스컬리 어디로 가는데요?

멀더 AT & T 벨 연구소입니다.

<스컬리와 멀더는 탐정 사무소를 나와 자동차로 이동한다>

<End of Part 1>