

혼자 공부하며 함께 만드는

혼공 용어 노트

목차

가나다순

| | | | |
|-----------------------------------|----|-----------------------------|----|
| 2의 보수 2's complement | 9 | 배열명 | 22 |
| 구조체 structure | 28 | 배타적 논리합 exclusive or | 13 |
| 기억 부류 storage class | 24 | 버퍼 buffer | 23 |
| 다차원 배열 multi-dimensional array | 25 | 변수 variable | 9 |
| 동적 할당 dynamic allocation | 27 | 변환 문자 | 7 |
| 디버깅 debugging | 7 | 분할 정복 기법 divide and conquer | 17 |
| 레지스터 변수 register variable | 25 | 분할 컴파일 | 30 |
| 레지스터 register | 14 | 블록 block | 15 |
| 로드 load | 14 | 비주얼 스튜디오 Visual Studio | 6 |
| 매개변수 parameter | 19 | 비트 bit | 8 |
| 매달린 else 문제 Dangling else Problem | 16 | 상수 constant | 8 |
| 매크로 함수 | 30 | 선택문 conditional statement | 15 |
| 메모리 누수 memory leak | 27 | 소스 코드 source code | 6 |
| 메모리 주소 address | 22 | 소스 파일 source file | 6 |
| 명령행 인수 command line argument | 27 | 숏 서킷 룰 short circuit rule | 13 |
| 모듈 module | 30 | 스토어 store | 14 |
| 무한 반복문 infinite loop | 18 | 스트림 stream | 29 |
| 바이너리 파일 binary file | 29 | 식별자 identifier | 12 |
| 바이트 어라인먼트 byte alignment | 28 | 쓰레기 값 garbage (value) | 10 |
| 바이트 byte | 9 | 아스키 코드 ASCII code | 8 |
| 반복문 loop | 17 | 연결 리스트 linked list | 29 |
| 배열 array | 20 | 연산자 operator | 12 |

| | | | |
|---------------------------------|----|----------------------|----|
| 열거형 enumeration | 29 | 토큰 token | 30 |
| 예약어 reserved word | 12 | 패딩 바이트 padding byte | 28 |
| 이식성 portability | 31 | 포인터 pointer | 21 |
| 이중 포인터 double pointer | 26 | 포인터 배열 pointer array | 26 |
| 자료형 data type | 11 | 피연산자 operand | 12 |
| 재귀호출 함수 recursive call function | 19 | 함수 function | 19 |
| 전역 변수 global variable | 24 | 헤더 파일 header file | 11 |
| 전위 표기 prefix | 13 | 형 변환 | 13 |
| 전처리 preprocessing | 30 | 형 재정의 typedef | 29 |
| 정규화 normalization | 7 | 화이트 스페이스 white space | 23 |
| 정적 지역 변수 static local variable | 24 | 후위 표기 postfix | 13 |
| 정적 함수 static function | 30 | | |
| 제어문 | 15 | | |
| 제어 문자 control character | 8 | | |
| 주석문 comment | 8 | | |
| 주소 연산자 | 21 | | |
| 중복 포함 문제 | 31 | | |
| 지역 변수 local variable | 24 | | |
| 첨자 index | 21 | | |
| 초기화 initialize | 10 | | |
| 컴파일 compile | 6 | | |
| 텍스트 파일 text file | 29 | | |

ABC 순

| | | | |
|-----------------------------------|----|--------------------------------|----|
| 2's complement 2의 보수 | 9 | divide and conquer 분할 정복 기법 | 17 |
| address 메모리 주소 | 22 | double pointer 이중 포인터 | 26 |
| Arithmetic Logic Unit ALU | 14 | dynamic allocation 동적 할당 | 27 |
| array 배열 | 20 | enumeration 열거형 | 29 |
| ASCII code 아스키 코드 | 8 | exclusive or 배타적 논리합 | 13 |
| binary file 바이너리 파일 | 29 | function 함수 | 19 |
| bit 비트 | 8 | garbage (value) 쓰레기 값 | 10 |
| block 블록 | 15 | gets 함수 | 23 |
| buffer 버퍼 | 23 | global variable 전역 변수 | 24 |
| byte 바이트 | 9 | header file 헤더 파일 | 11 |
| byte alignment 바이트 얼라인먼트 | 28 | identifier 식별자 | 12 |
| command line argument 명령행 인수 | 27 | index 첨자 | 21 |
| comment 주석문 | 8 | infinite loop 무한 반복문 | 18 |
| compile 컴파일 | 6 | initialize 초기화 | 10 |
| conditional statement 조건문 | 15 | linked list 연결 리스트 | 29 |
| const 변수 | 12 | load 로드 | 14 |
| constant 상수 | 8 | local variable 지역 변수 | 24 |
| continue | 18 | loop 반복문 | 17 |
| control character 제어 문자 | 8 | main 함수 | 7 |
| Dangling else Problem 매달린 else 문제 | 16 | memory leak 메모리 누수 | 27 |
| data type 자료형 | 11 | module 모듈 | 30 |
| debugging 디버깅 | 7 | multi-dimensional array 다차원 배열 | 25 |

| | | | |
|---------------------------------|----|--------------------------------|----|
| normalization 정규화 | 7 | static local variable 정적 지역 변수 | 24 |
| operand 피연산자 | 12 | stdlib.h 헤더 파일 | 27 |
| operator 연산자 | 12 | storage class 기억 부류 | 24 |
| padding byte 패딩 바이트 | 28 | store 스토어 | 14 |
| parameter 매개변수 | 19 | strcpy 함수 | 24 |
| pointer 포인터 | 21 | stream 스트림 | 29 |
| pointer array 포인터 배열 | 26 | string.h 헤더 파일 | 11 |
| portability 이식성 | 31 | structure 구조체 | 28 |
| postfix 후위 표기 | 13 | switch~case문 | 16 |
| prefix 전위 표기 | 13 | text file 텍스트 파일 | 29 |
| preprocessing 전처리 | 30 | token 토큰 | 30 |
| printf 함수 | 8 | typedef 형 재정의 | 29 |
| recursive call function 재귀호출 함수 | 19 | variable 변수 | 9 |
| register 레지스터 | 14 | Visual Studio 비주얼 스튜디오 | 6 |
| register variable 레지스터 변수 | 25 | white space 화이트 스페이스 | 23 |
| reserved word 예약어 | 12 | | |
| scanf 함수 | 23 | | |
| short circuit rule 숏 서킷 룰 | 13 | | |
| sizeof 연산자 | 21 | | |
| source code 소스 코드 | 6 | | |
| source file 소스 파일 | 6 | | |
| static function 정적 함수 | 30 | | |

01장 [✓]

□ 소스 파일 source file [1장 33쪽]

프로그래밍 언어의 문법에 맞게 작성한 문서 파일. 여기서는 C 언어의 문법에 맞게 작성한 문서 파일을 말한다.

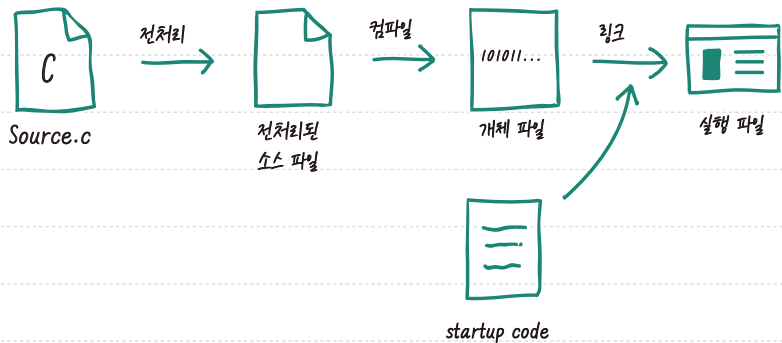
□ 소스 코드 source code [1장 41쪽]

= 코드. 프로그래밍 언어의 문법에 맞게 작성한 글

□ 컴파일 compile [1장 33쪽]

소스 파일을 컴퓨터가 이해하는 기계어로 바꾸는 과정

컴파일 3단계



□ 비주얼 Visual Studio [1장 34쪽]

스튜디오

소스 코드를 편집하고 컴파일할 수 있게 도와주는 프로그램. C 언어를 컴파일할 때는 VC++ 컴파일러를 사용하는데, 반드시 파일의 확장자를 .c로 적어야 C 언어의 문법에 맞게 컴파일한다. 비주얼 스튜디오는 자동완성 기능과 문법 검사 기능, VC++ 컴파일러를 이용해 컴파일하거나 디버깅하는 기능을 제공한다.

에러를 수정하는 것

02장

코드 실행이 제일 처음 시작되고 제일 마지막으로 끝나는 곳. 줄여서 'main' 또는 '메인'이라고도 한다.

```
int main(void)
```

```
{
```

```
// 실행 코드 블록
```

```
}
```

이 사이의 코드들을 블록이라고 한다.

출력 시 데이터의 형태에 따라 변숫값을 텍스트로 바꾸어 주거나, 입력 시 텍스트를 알맞은 변숫값으로 바꾸어 주는 문자

* 자주 사용하는 변환 문자! (알기하자)

`int => %d` (정수)

`double => %lf` (실수)

`char => %c` (문자)

`char배열 => %s` (문자열)

소수점 앞에 0이 아닌 한 자리만을 사용해 지수 형태로 바꾸는 것

* 실수를 정규화하는 과정

`0.0000314 => 3.14e-5` 10^{-5} 라는 의미!

| | | |
|-------------|---|----------|
| □ 아스키 코드 | ASCII code | [2장 71쪽] |
| | 사람이 사용하는 기호를 컴퓨터 안에서 표현하는 방법에 대해 약속한 것 | |
| □ 주석문 | comment | [2장 54쪽] |
| | 실행 결과에 영향을 미치지 않는 문장. 소스 코드를 설명하는 내용을 담게 된다. | |
| | <ul style="list-style-type: none"> • <code>// : //</code>부터 해당 줄의 끝까지 주석 처리 • <code>/* ~ */ : /*과 */</code> 사이의 모든 내용 주석 처리 | |
| | <div> <pre>//이 책을 다 보고 나서 무엇을 공부하면 될까요? /*예제가 C 언어로 제공되는 <이것이 자료구조+알고리즘이다>를 추천드리고요, 또 다른 언어를 공부하고 싶다면 <혼자 공부하는 파이썬> / <혼자 공부하는 자바>를 추천합니다! 그리고 <혼공단> 참여도 추천합니다! */</pre> <div> 혼공단 아이돌 '혼공양' 드림  </div> </div> | |
| □ printf 함수 | | [2장 55쪽] |
| | 화면에 데이터를 출력하는 함수. print formatted의 약어. 변환 문자를 사용해서 변수에 맞는 자료형을 알려 주다. | |
| □ 제어 문자 | control character | [2장 56쪽] |
| | 문자는 아니지만, 출력 방식에 영향을 주는 문자. <code>\n</code> , <code>\t</code> 등등 출바꿈(개행) 탭만큼 띄어쓰기 | |
| □ 상수 | constant | [2장 64쪽] |
| | 한 번 값을 정하면 프로그램이 끝날 때까지 변하지 않는 변수. 원주율 값처럼 값이 정해져 있고 변하면 안 되는 경우에 주로 사용한다. | |
| □ 비트 | bit | [2장 71쪽] |
| | 컴퓨터 저장장치의 가장 작은 단위. 스위치와 같이 2가지 상태를 나타낸 것. 1(혹은 true)과 0(혹은 false)으로 나타낸다. | |

□ 바이트

byte

[2장 71쪽]

컴퓨터 저장장치의 단위. 1바이트는 8비트. 16진수를 사용하는 이유는 4비트 단위로 이진수 값을 표현하기 위해서다.

* 컴퓨터의 저장 단위

$$1\text{byte} = 8\text{bit}$$

$$1\text{KB}(\text{킬로바이트}) = 2^{10}\text{byte} (2^{10} = 1024)$$

$$1\text{MB}(\text{메가바이트}) = 2^{10}\text{KB} = 2^{20}\text{byte}$$

$$1\text{GB}(\text{기가바이트}) = 2^{10}\text{MB}$$

$$1\text{TB}(\text{테라바이트}) = 2^{10}\text{GB}$$

□ 2의 보수

2's complement

[2장 74쪽]

절대값인 2진수의 0과 1을 반전시킨 상태(이 상태를 1의 보수라 함)에서 1을 더한 값

* -8을 2의 보수로 표현하기

$$\textcircled{1} 8\text{을 } 2\text{진수 변환} \rightarrow 1000_{(2)}$$

$$\textcircled{2} \text{비트 반전} \rightarrow 0111$$

$$\textcircled{3} 1\text{을 더한다} \rightarrow 1000$$

-8을 컴퓨터가 저장하는 방식!!

03장

□ 변수

variable 참고 용어 지역 변수, 전역 변수

[3장 80쪽]

데이터를 저장하는 메모리의 공간

• 변수 선언: 변수를 어떤 형태와 이름으로 사용할 것이라고 알리는 것

• 할당: 변수에 값을 넣거나 저장하는 것을 '할당한다'고 한다.

• 참조: 변수에 접근하는 것을 '참조한다'고 한다.

메모리에 변수의 값이 들어갈 공간을 할당해야 하므로.

그것이 알고싶다 signed 변수와 unsigned 변수

signed 변수는 부호를 가진 변수며 선언 시 이를 따로 표기하지 않는다. unsigned 변수는 부호가 없는 변수를 사용하고 싶을 때 사용하는 예약어며, 변수 선언 시 unsigned int와 같이 사용하면 된다. int를 2바이트라고 하고 예시로 들면 signed int는 -32768 ~ 32767, unsigned int는 0~65535의 범위를 갖는다. 즉, 음수를 표현할 수 있는 범위를 양수를 표현하는 데에 사용한다.

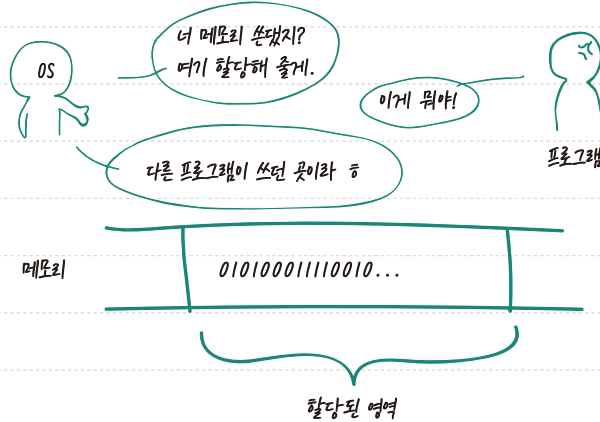
□ 쓰레기 값

garbage (value)

[3장 81쪽]

변수로 지정했던 공간에는 다른 프로그램이 사용한 흔적이 남아 있을 수 있는데, 남아 있는 이 값이 쓰레기 값!

* 초기화되지 않은 변수에 접근해도 오류가 뜨지 않는 언어들에서는 종종 초기화하지 않은 변수 때문에 버그가 생긴다.



□ 초기화

initialize

[3장 81쪽]

변수, 배열, 구조체에 값을 처음으로 대입하는 것

```
// 변수 초기화의 유형
/* 1. 선언과 동시에 초기화 */
int num1 = 3;
/* 2. 사용자 입력에 의한 초기화 */
int num2;
scanf("%d", &num2);
/* 3. 코드 중간에 초기화 */
int num3;
num3 = 3;
```

```
//배열 초기화의 유형
/* 1. 선언과 동시에 초기화 */
int ary1[5] = {1, 2, 3, 4, 5};
int ary2[5] = {1, 2, 3};
int ary3[] = {1, 2, 3};
/* 2. 개별 원소를 초기화 */
int ary1[4] = 3; //이 경우 주로 반복문을 이용해 초기화
for(int i = 0; i < 5; i++)
{
    ary1[i] = 0;
}
/* 3. memset을 이용한 초기화 */
memset(ary1, 0, sizeof(ary1)); //모든 원소를 0으로 초기화
```

□ 자료형

data type

[3장 85쪽]

자료의 형태, 자료형에 따라 컴퓨터가 데이터를 처리하는 방식이 달라진다.

- 정수형: int(4bytes), short(2bytes), long(4bytes)
- 문자형: char(1byte)
- 실수형: double(8bytes)

| 종류 | 표현 방법 | 사용 예 |
|-----|--------------------------|---|
| 정수 | 0~9, +, - 기호 사용 | 10, -5, +20, 0 |
| 실수 | 0~9, +, -, . (소수점) 기호 사용 | 3.4, -1.7, 5, 10.0 → 0.5와 같다. 10을 실수 레이어로 처리하거나, 실수와 연산을 하고 싶을 때 이렇게 표현한다 |
| 문자 | 문자를 작은 따옴표로 묶음 | 'A', 'b', '0', '*' |
| 문자열 | 하나 이상의 문자를 큰 따옴표로 묶음 | "A", "banana" |

□ 헤더 파일

header file 참고 용어 전처리 지시자

[3장 93쪽]

표준 라이브러리 함수의 원형 등을 포함하고 있는 파일. '헤더'라고도 한다. 라이브러리는 쉽게 가져다 쓰라고 미리 구현해 둔 함수. #include라는 전처리 지시자로 불러들인다.

전처리 지시자가 무엇인지 몰라도 된다. 지금은 #include가
전처리 지시자라는 것만 알고 뒤에서 자세히 배우자.

□ string.h

[3장 93쪽]

헤더 파일

문자열 처리와 관련된 함수가 선언된 헤더 파일

□ const 변수

[3장 94쪽]

한 번 초기화하면 값을 바꿀 수 없는 변수. 코드가 실행되는 동안은 값을 바꿀 필요가 없을 때 사용한다.

ex) 원주율, 황금 비율, 게임에서 공격력 수치가 데미지로 환산되는 비율 등

```
const int buflen = 1024; //정수형, const를 사용한 변수
#define KEY 'A' //문자형, #define을 사용한 매크로 상수
#define ERROR_MESSAGE "Error Occurred!" //문자열형
```

□ 식별자

identifier

[3장 95쪽]

필요에 따라 사용자가 만들어 쓰는 단어. 보통 함수나 변수의 용도에 맞게 의미 있는 이름을 사용한다.

□ 예약어

reserved word

[3장 95쪽]

언어에 의해 예약된 단어. 식별자로 사용할 수 없다.

04장 ✓

□ 연산자

operator

[4장 112쪽]

연산에 사용되는 표시나 기호



□ 피연산자

operand

[4장 112쪽]

연산의 대상이 되는 데이터



□ 전위 표기

prefix

[4장 117쪽]

연산자가 피연산자 앞에 놓이는 수식 표기

* 전위 표기 : $+ab$

* 후위 표기 : abt

* 중위 표기 : $a + b$

□ 후위 표기

postfix

[4장 117쪽]

연산자가 피연산자 뒤에 놓이는 수식 표기

* 우리가 흔히 사용하는 수식 표기는 '중위 표기'이다.

□ 형 변환

[4장 129쪽]

자료형을 다른 자료형으로 변환하는 것

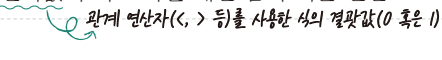
- 자동 형 변환(묵시적, 암시적 형 변환 / promotion): 자료형이 일치하지 않는 값이 들어왔을 때, 변환이 가능한 타입에 한해 컴파일러가 알맞은 자료형으로 해당 값을 바꾸어 주는 것
- 강제 형 변환(명시적 형 변환 / casting): 값을 일시적으로 원하는 형태로 바꾸는 것

□ 배타적 논리합

exclusive or

[4장 139쪽]

두 피연산자의 진리값이 서로 다를 때만 참이 되는 연산

* 진리표 

| a | b | $a \wedge b$ |
|-----|-----|--------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

□ 숏 서킷 룰

short circuit rule

[4장 121쪽]

좌항만으로 $\&\&$ 와 $\|\$ 연산 결과를 판별하는 기능

```

a = 0;
b = 1
if (a && b) // a만 검사하고 빠져나온다
if (a || b) // b까지 검사해야 논리 연산의 결과값을 알 수 있다

```

□ ALU

Arithmetic Logic Unit

[4장 123쪽]

CPU의 산술논리 연산장치

□ 레지스터

register 참고 용어 레지스터 변수

[4장 123쪽]

CPU의 메모리. 연산할 데이터와 연산 후의 결과를 임시로 저장한다.

CPU 메모리는 메인 메모리에 있는 자료처럼 복사하는 과정이 없으니 당연히 빠름!

□ 로드

load

[4장 123쪽]

메인 메모리에서 레지스터로 값을 복사하는 과정

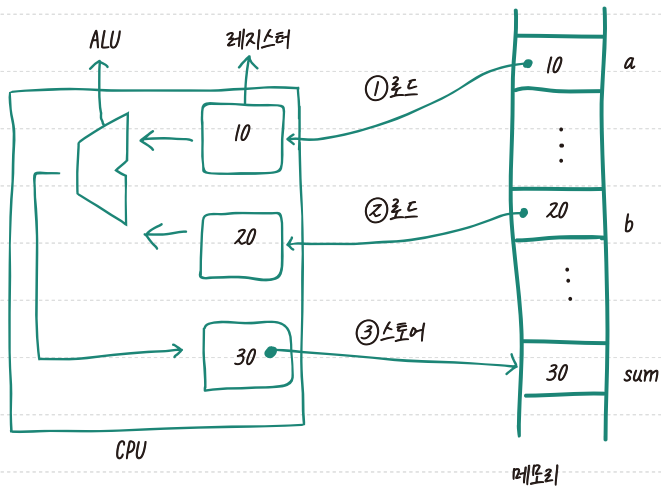
□ 스토어

store

[4장 123쪽]

연산이 완료된 값을 레지스터에서 메인 메모리로 복사하는 과정

$sum = a + b$ 가 실행되는 과정



연산을 할 때 CPU는 메모리에 있는 변수의 값을 가져와 연산을 한다. 이 때 CPU는 메모리에서 연산 작업을 수행하지 않고 CPU의 내부에 있는 레지스터에 메모리의 내용을 복사해서 연산해야 하는데, 이 복사하는 과정을 '로드'라고 한다. 데이터가 레지스터에 저장되면 연산장치인 ALU에 의해 덧셈 연산이 수행되고 그 결과값은 일단 레지스터에 저장된다. 이후 대입 연산을 수행하면 메모리 공간인 sum에 복사되어 수식의 모든 과정이 완료된다. 연산할 때는 메모리에 있는 변수의 값을 CPU로 복사해서 사용하므로 아무리 많은 연산을 수행해도 피연산자 a, b의 값은 변하지 않고, 대입 연산을 수행한 sum은 연산장치 ALU에서 어떤 연산이 수행되느냐에 따라 값이 변할 수 있다.

05장

□ 제어문

[5장 150쪽]

특정 조건에 따라 실행하거나 실행하지 않아야 할 때 사용하는 문장

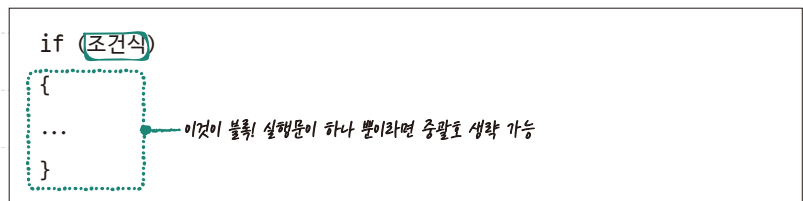
- 선택문: if문, switch~case문
- 반복문: for문, while문, do~while문
- 분기문: break, continue, return

□ 블록

block **참고 용어** 지역 변수

[5장 173쪽]

함수, 반복문, 선택문 등의 중괄호로 이루어진 단위를 말한다.



□ 선택문

conditional statement

[5장 150쪽]

(조건문)

특정 조건을 만족할 때 코드를 실행하는 문법

- if문: 괄호 내의 조건식이 참이면 블록 내의 문장을 실행한다.

- else문: if문의 조건식이 거짓이면 블록 내의 문장을 실행한다. 필요 없으면
없어도 된다.
- else if문: if문의 조건식이 거짓일 때 실행시킬 코드에 추가 조건을 걸고 싶
을 때 사용한다. 마찬가지로 필요 없으면 else if를 사용하지 않아도 된다.

```
if (a > 0)
    printf("a는 양수입니다.\n "); //중괄호를 사용하지 않으면 조건식이
    참일 때 한 줄만 실행
else if (a == 0)
{
    printf("a는 0입니다.\n");
}
else
{
    printf("a는 음수입니다.\n");
}
```

□ 매달린 **Dangling else Problem** [5장 172쪽]

else 문제 if문을 중첩해서 사용할 때 뒤따르는 else의 위치가 모호해져 생기는 문제

□ switch~case [5장 172쪽]

문 여러 선택지 중 조건을 만족하는 선택지의 코드를 실행하는 문법

- switch문: 괄호에 비교 대상을 넣어 블록 내의 각 case문을 검사한다.
- case: ‘case (해당하는 값):’과 같이 적는다. 해당하는 값은 비교 대상 변수
에 맞는 자료형의 데이터를 적는다. 정수, 문자, 열거 상수 등이 될 수 있다.
- default: 어떤 케이스도 비교 대상과 맞지 않을 때 이 문장을 실행할 수 있
다. case문을 모두 적은 후 마지막에 적을 수 있으므로 위의 case문에 해당
하지 않을 때 실행시키는 용도로 사용할 수 있다.
- break: 반복문, 선택문 블록을 빠져나오게 하는 예약어


```
switch(ch)
{
case 'a': //이 케이스에서 break가 없기 때문에 블록을 빠져나가지 않고
다음 case문으로 넘어간다
case 'b':
    printf("ch는 a 혹은 b입니다\n"); // a일 경우에도 출력된다.
    break;
case 'c':
    printf("ch는 c입니다\n");
    break;
default:
    printf("a도 b도 c도 아닙니다\n");
    break; //이 break가 없어도 이 블록을 빠져나갈 수 있다.
}
```

□ 분할 정복 기법

divide and conquer

[5장 169쪽]

재귀에 기반해 큰 문제를 작게 쪼개 하나씩 처리하며 결과를 취합하는 문제 해결 기법

06장

□ 반복문

loop

[6장 182쪽]

특정 조건을 만족하는 동안 반복해서 실행하는 문법

for문: 반복 횟수가 정해진 경우에 주로 사용

```
//for문 예시
for(int i = 0; i < n; i++)
//0부터 n-1까지 i를 증가시키며 코드 실행 (n회 반복)
...
```

while문: 반복 횟수를 모를 때 주로 사용

```
//while문 예시
while(1)
{
    result *= 3;
    if(result >= 100) //이 조건을 만족시켜야 반복문 종료
        break;
}
```

do-while문: 조건 만족 여부와 상관없이 코드를 먼저 실행하고 다음 루프부터 조건을 검사

```
//do-while문 예시
do
{
    printf("어머 이걸 꼭 출력되어야 해...! ");
} while(0); //1회만 출력
```

□ 무한 반복문

infinite loop

[6장 201쪽]

무한히 반복을 하고 싶거나 정해진 횟수 없이 일정한 조건을 충족하면 빠져나오게 하고 싶을 때 사용. 무한 루프라고도 한다.

```
//for문을 사용한 무한 루프
for(;;)
{
    //반복하고 싶은 코드
}

//while문을 사용한 무한 루프
while(1)
{
    //반복하고 싶은 코드
}
```

□ continue

[6장 202쪽]

continue 아래의 코드를 실행하지 않고, 반복문의 조건을 검사한 후 다시 루프를 시작하게 하는 예약어

07장

□ 함수

function

[7장 208쪽]

기능을 수행하는 코드 단위

- 함수 정의: 함수를 실제 코드로 만드는 것
- 함수 호출: 필요한 곳에서 함수를 사용하는 것
- 함수 선언: 프로그램의 상단에서 어떤 함수를 만들어서 쓸 것이라고 컴파일러에 알려 주는 것
- 함수 원형: 함수명, 매개변수, 반환형을 적은 것

그것이 알고싶다 함수 원형을 선언하는 이유

컴파일러가 함수를 호출하는 코드를 만나기 전에, 매개변수가 어떤 자료형이고 몇 개인지 미리 알리면 잘못된 매개변수를 전달하면서 생기는 에러를 막을 수 있기 때문이다.

□ 재귀호출 함수

recursive call function

[7장 226쪽]

함수 안에서 자신을 호출하는 함수

* 재귀함수에 들어가야 할 코드

1. 종료 조건을 검사하는 코드
2. 재귀하며 수행할 코드(출력 등)
3. 자기 자신을 호출하는 코드

그것이 알고싶다 재귀를 사용하는 상황

재귀 함수는 반복문을 쓰는 경우보다 간결한 코드로 구현할 수 있다는 장점이 있다. 혹은 반복문보다 많은 메모리를 쓰는 대신 비교적 빠른 작업을 할 수 있는 문제들이 있는데, 일부 정렬 알고리즘이 그런 경우에 해당한다.

□ 매개변수

parameter

[7장 221쪽]

함수가 처리할 데이터를 저장하는 변수

그것이 알고싶다 인수(argument)와 매개변수(parameter)

```
int mySum(int arr[], int length)
{
    int result = 0;
    for(int i = 0; i < length; i++)
    {
        result += arr[i];
    }
    return result;
}
...
int myVar = mySum(myArray, 6);
...
```

매개변수
인수

08장

□ 배열

array

[8장 236쪽]

기본 자료형을 여러 개 묶어서 사용하는 것. 인덱스를 가지고 순차적으로 순회하며 변수에 접근할 수 있다.

- 배열 선언: 어떤 이름을 가지고 어떤 형태의 변수가 몇 개인지를 컴파일러에 알리는 것

```
int ary[5] = {1, 2, 3, 4, 5};
for(int i = 0; i < 5; i++)
{
    printf("%d" ary[i]);
}
```

이것이 철자!

□ 첨자

index

[8장 238쪽]

배열에서 순차적으로 나열된 요소에 매겨진 번호. 요소에 접근할 때 사용한다. 영
번호는 0부터 시작한다. 0, 1, 2, 3,
문명 그대로 인덱스라고도 한다.

□ sizeof 연산자

[8장 244쪽]

변수가 메모리에 할당된 크기를 바이트 단위로 반환하는 함수

```
int ary[5] = {1, 2, 3, 4, 5};  
printf("%d", sizeof(ary)); //출력: 20  
  
(int *)malloc(sizeof(int) * n);
```

09장

□ 포인터

pointer

[9장 266쪽]

주소를 저장하는 변수로 일반 변수와 마찬가지로 선언 후에 사용한다.

- 간접 참조 연산자(*): 포인터가 가리키는 변수를 사용할 때 포인터에 사용하는 특별한 연산자
- 주소 연산자(&): 변수의 주소를 구할 때 사용하는 연산자

□ 주소 연산자

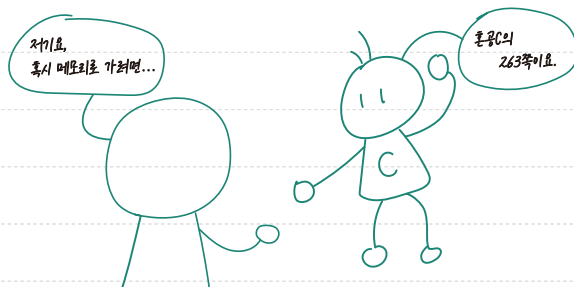
[9장 264쪽]

주소를 구할 때 사용하는 연산자. 포인터에 이 주소값을 저장해 해당 메모리에 접근할 수도 있고, scanf에서 사용할 때처럼 함수에 메모리 주소를 전달하는 용도로 사용할 수도 있다. 비트 연산자 &와는 다르므로 헷갈리지 말자

주소, 주소 값이라고도 한다. 메모리상 데이터의 위치를 식별할 수 있는 값. 보통은 해당 데이터의 시작 위치를 주소 값으로 가짐.

그것이 알고싶다 주소와 포인터의 차이

주소는 변수에 할당된 메모리 저장 공간의 시작 주소값 자체이고 포인터는 그 값을 저장하는 또 다른 메모리의 공간이다. 또한 모든 주소와 포인터는 가리키는 자료형과 관계없이 크기가 같다.



10장 ✓

배열명은 배열의 첫 번째 요소의 주소이기 때문에 배열의 첫 번째 요소를 가리킨다. 포인터처럼 사용할 수 있으나, 포인터와 다르게 배열명은 상수이므로 그 값을 바꿀 수 없다.

* 배열명의 특징

배열명의 덧셈이나 뺄셈을 하면 0에서 더한 숫자만큼 인덱스를 움직여 요소에 접근한다. 그러나 실질적으로 접근하는 메모리 주소는 더한 숫자에 자료형의 크기만큼 움직인 곳이다.

```
int ary[5];
*(ary + 1); //ary[1]의 값에 접근
```

11장

□ scanf 함수

[11장 325쪽]

stdin 스트림 파일로부터 데이터를 형식에 따라 변환해 입력하는 함수

□ 화이트

white space

[11장 327쪽]

스페이스

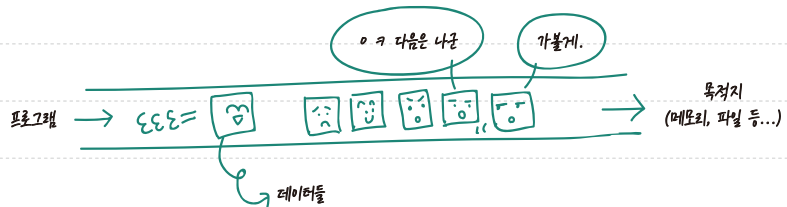
space bar, tab, enter 키를 눌렀을 때 입력되는 문자를 묶어 부르는 말

□ 버퍼

buffer

[11장 332쪽]

프로그램이 출력한 데이터를 모아서 한꺼번에 출력 장치로 보내거나 입력 장치에서 한 번에 많은 데이터를 읽어 저장해 놓고 프로그램이 필요한 데이터를 바로 꺼내 쓸 수 있도록 준비하는 공간



12장

□ gets 함수

[12장 352쪽]

stdin 스트림 파일로부터 한 줄의 데이터를 읽는 함수

```
//gets()의 함수 원형
char *gets(char *str);
```

문자열을 복사한다.

//strcpy()의 함수 원형

```
char *strcpy(char *dest, const char* src);
```

13장

블록 내에 선언되어 해당 블록 내에서만 사용할 수 있는 변수. 따라서 다른 블록 내에서는 사용할 수 없다. auto 예약어와 함께 함수 안에서 지역 변수를 선언한다. auto가 없어도 블록 안에 선언된 변수는 자동으로 지역 변수가 된다.

그것이 알고싶다 지역 변수와 자동 변수

둘은 같은 용어다. 지역 변수는 기억 부류 중 하나. 따라서 지역 변수는 변수의 특성 중에 사용 범위를 강조한 것이며, 자동 변수는 지역 변수가 auto 예약어를 사용하므로 붙여진 것이다.

사용 범위와 메모리에서의 존재 기간에 따라 변수를 나눈 것

함수 밖에 변수를 선언하면 전역 변수가 된다. 전역 변수는 특정 함수의 블록에 포함되지 않으므로 사용 범위가 함수나 블록으로 제한되지 않는다

선언된 함수가 반환되더라도 그 저장 공간을 계속 유지하는 변수


```
void f() {
    static int count = 0;
    count++;
    printf("%d", count);
}

int main(){
    f(); //출력: 1
    f(); //출력: 2
    f(); //출력: 3
    return 0;
}
```

□ 레지스터 변수

register variable

[13장 394쪽]

블록 혹은 함수 내에 변수를 선언할 때 사용할 수 있는 변수다. CPU 내의 공간인 레지스터에 저장되기 때문에 일반적인 지역 변수와는 다르다.

그것이 알고싶다

레지스터 변수로 지정했는데, 레지스터 변수가 아닐 수도 있다?

레지스터는 CPU의 연산장치가 사용하는 비싸고 중요한 저장 공간이므로 당장 연산할 필요가 없는 데이터를 레지스터에 보관하면 레지스터의 활용성이 떨어진다. 따라서 컴파일러는 사용자가 레지스터 변수를 선언하더라도 레지스터와 메모리 중 어디에 할당하는 것이 더 이득인지 판단해 적당한 저장 공간을 선택한다.

14장

□ 다차원 배열

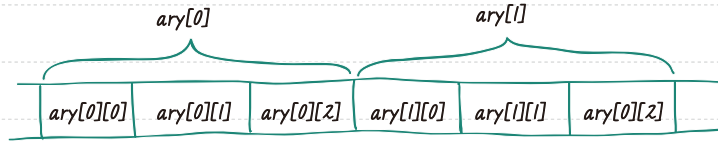
multi-dimensional array

[14장 410쪽]

배열을 요소로 갖는 배열. 2차원 배열에서는 일반 배열(1차원 배열)을 요소로 갖고, 3차원 배열에서는 2차원 배열을 요소로 갖는 형식이다.

```
int ary[2][3] = { {1, 2, 3}, {4, 5, 6} };
```

* 다차원 배열이 메모리에 저장된 모양



결국, 저장 방식은 일차원 배열과 같다

ary[0], ary[1]도 배열이기 때문에 배열명처럼 요소에 접근할 수 있다.

* 다차원 배열을 논리적으로 표현한 형태들



행(row)이 두 번째 괄호, 열(column)이 첫 번째 괄호라고 생각하면 된다

→ 행렬, 좌표 평면 등을 표현할 때 응용된다.

□ 포인터 배열

pointer array

[14장 432쪽]

포인터를 모아 만든 배열. 여러 개의 문자열을 다루거나 2차원 배열처럼 사용할 수 있다.

15장 ✓

□ 이중 포인터

double pointer

[15장 445쪽]

주소값 자체를 처리하는 포인터. 즉, 주소를 저장한 포인터도 하나의 변수이고 따라서 그 주소를 구할 수 있으며 또 다른 포인터에 저장하고 가리키는 형태다.

```
int * * ppi; //이중 포인터
가리키는 변수의   자신의 자료형을 의미함
자료형
```

16장

□ `stdlib.h`

[16장 479쪽]

헤더 파일

메모리 동적 할당이나 문자열을 정수로 변환하는 함수 등 다양한 기능의 범용 함수가 정의된 헤더 파일.

□ 동적 할당

dynamic allocation

[16장 479쪽]

프로그램 실행 중에 저장 공간을 할당하는 것. 주로 코드 실행 중에 배열에 저장할 값의 개수가 결정되는 경우에 사용된다.

```
int *arr;
...
arr = (int *)malloc(sizeof(int) * n);
```

□ 메모리 누수

memory leak

[16장 482쪽]

메모리 공간을 사용하고 나서 반환하지 않았을 때 할당된 채로 사용하지 않는 공간이 생기는 것.

누수 → 새어 나감 → 내용물 부족해짐

=> 메모리 공간이 새어나가 빈 공간이 부족한 상태.

□ 명령행 인수

command line argument

[16장 497쪽]

명령행에서 프로그램을 실행시킬 때 프로그램의 이름 외에 함께 주는 프로그램에 필요한 정보

```
$ ./HongongExample arg1 arg2
C:\> HongongExample arg1 arg2
//프로그램 내부
```

리눅스 계열
윈도우

```
int main(int argc, char** argv)
```

```
...
```

병렬형 인수의 개수. 포로
그림의 이름을 포함하므로
항상 1 이상이다.

병렬형 인수를 가진 이차원 배열. 문자열을 가진
배열이라고 생각하면 된다.

argv[0] : HongongExample
argv[1] : arg1
argv[2] : arg2

17장

□ 구조체

structure

[17장 505쪽]

형태가 서로 다른 변수를 묶는 자료형. 한 번 형태가 정의되면 그 이후부터는 구조체 변수, 구조체 배열, 구조체 포인터 등으로 활용할 수 있다.

- 자기 참조 구조체: 자기 자신을 가리키는 포인터를 갖는 구조체

□ 패딩 바이트

padding byte

[17장 507쪽]

구조체 멤버의 크기가 들쭉날쭉한 경우 정렬할 용도로 멤버 사이에 넣는 것

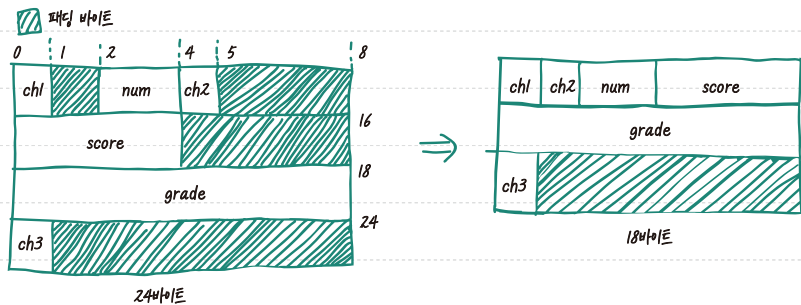
□ 바이트

byte alignment

[17장 507쪽]

얼라인먼트

패딩 바이트를 넣어 멤버를 정렬하는 것



*구조체 멤버 변수 선언 순서에 따라 구조체의 크기가 달라질 수 있다. (패딩 바이트가 들어가기 때문!)

□ 열거형

enumeration

[17장 532쪽]

변수에 저장할 수 있는 정수 값을 기호로 정의해 나열하는 자료형

```
enum state{ FEVER, COUGH, RUNNY_NOSE }; //멤버의 이름을 대문자로  
짓는 것이 관례!  
  
enum season{ SPRING, SUMMER, FALL, WINTER };
```

□ 연결 리스트

linked list

[17장 529쪽]

자기 참조 구조체 변수를 포인터로 연결한 것

□ 형 재정의

typedef

[17장 533쪽]

자신이 필요로 하는 자료형을 만들어 쉽게 선언하도록 하는 문법

18장

□ 스트림

stream

[18장 545쪽]

프로그램이 외부 파일, 외부 네트워크 등과 통신할 때 데이터가 흐르는 길

□ 텍스트 파일

text file

[18장 557쪽]

데이터를 아스키 코드 값에 따라 저장한 것

□ 바이너리 파일

binary file

[18장 557쪽]

텍스트 인코딩 이외의 방식으로 저장된 파일

ex) ASCII, Unicode 등

19장

| | | |
|----------|---|------------|
| □ 전처리 | preprocessing | [19장 584쪽] |
| | 전처리 지시자에 따라 소스파일을 가공하는 과정  #include, #define 등 | |
| □ 모듈 | module | [19장 587쪽] |
| | 분할 컴파일 시 컴파일 가능한 파일 단위. 파일을 모듈별로 나누는 목적은 코드를 재사용하거나, 거대한 프로그램에서 일부 수정사항이 있을 때 변경된 부분만 컴파일할 수 있도록 함에 있다. | |
| □ 분할 컴파일 | | [19장 605쪽] |
| | 하나의 프로그램을 여러 개의 소스 파일로 나누어 각각 독립적으로 컴파일하는 것 | |
| □ 매크로 함수 | | [19장 589쪽] |
| | 인수에 따라 서로 다른 결괏값을 갖도록 치환되므로 비록 함수는 아니지만 인수를 주고 함수처럼 쓸 수 있는 것 | |
| | <pre>#define SWAP(x, y, t) (t) = (x); (x) = (y); (y) = (t); ... SWAP(a, b, temp); //x와 y에 저장된 값을 서로 바꾼다</pre> | |
| □ 토큰 | token | [19장 595쪽] |
| | 프로그램에서 독립된 의미를 갖는 하나의 단위 | |
| □ 정적 함수 | static function | [19장 613쪽] |
| | 같은 소스 파일 내에서만 호출할 수 있는 함수 | |

□ 이식성

portability

[19장 584쪽]

기종이 다른 컴퓨터에서도 사용할 수 있도록 만들어진 프로그램의 성질

□ 중복 포함 문제

[19장 614쪽]

헤더 파일을 재활용하는 과정에서 구조체 등이 중복 선언되는 문제

