시스템프로그래밍기초 정리

1강 - 어휘요소, 연산자, C시스템
2강 - 데이터 타입의 기본
3강 - flow of control
4강 - 함수
5강 - 포인터
6강 - 전처리기
7강 - 공용체와 구조체

기본이 되는 어휘 요소, 연산자, 시스템에 대해 살펴보자.

스스로가 컴파일러의 눈으로 지켜봐보자.
코딩(소스코드 인간이해가능) > 컴파일(번역, 매칭 과정) > binary(컴퓨터 이해할 수 있는)
번역, 매칭이 가능하려면 룰이 필요할 것이다. 룰을 이해하자.

컴파일러
▶ c프로그램이 **구문**에 맞는지 검사
▶ **오류**가 있다면, 오류 메시지를 출력(c언어는 엄격하게 체크)
▶ 오류가 없다면, 목적 코드 생성

ANSI C 토큰(띄어쓰기 : 문법적으로 의미를 갖는 최소단위) 종류 **시험**
이 6개를 벗어나면 신택스 에러.

▶ 키워드 : 예약어 - 기능요소
▶ 식별자(ID) : 이름이다./ 키워드 외의 모르는 애들이라면 식별자의 가능성이 크다. 엄격한 선언규칙을 갖고 있다.

데이터
▶ 상수 : 보존
▶ 문자열 상수 : 특별대우 없다.

처리(기능적 요소)
▶ 연산자
▶ 구두점

프로그램에 사용되는 문자 : 엄격하게 정의

소문자, 대문자, 숫자, 특수문자(쓸수 있는놈이 따로 있다.), 공백, 개행

예제 분석
'#'이 나오면 컴파일러 입장에서 전처리기와 연관, (컴파일은 공백으로 처리) 이 되겠구나 하고 생각하면 된다.
주석 하나는 공백과 대치
int, 함수이름은 예약어 중 하나.

printf는 예약어인가? : 아니다. **함수의 이름**은 키워드/예약어가 아니다. **식별자/id** 이다.

구문(bnf)법칙

italics : 구문
::= 정의하자면
| or

아주 엄격한 c컴파일러를 만나면 //라는 c++스타일 주석 때문에 에러가 날 수도 있다.

식별자의 첫 번째 문자는 문자, 밑줄문자만 가능하다.

상수
- 수치상수, 문자상수, 문자열 상수

수치상수 표기법
8진수 : 0을 맨 앞에 붙임
16진수: 0x 또는 0X를 맨앞에 붙인다.
지수 : e또는 E를 붙인다.
소수점 : 소수점을 사용한다.

문자 상수
- 0~255와 문자 대응시켜둔 숫자로 표현. 1바이트

문자열 상수
-문자 상수의 배열이다.

열거상수
-enum에 의해 선언된 상수

(주의) -49는 상수 수식임. 연산자 + 상수로 이루어진 수식.

작은 따옴표 : 문자상수 1byte
큰따옴표 : 문자상수가 모여 이루어진 배열. 문자열.

확장문자열(암기 - 시험)
\a
\n 개행
\t 탭
\xhh 16진수
\0 문자코드 0

연산자
산술연산자 (사칙연산, 이항연산자임.)
+ - * / %(몫을 리턴)

공백 없는 이항 연산자. 공백없이도 특수문자 자체가 구분자 역할을 해서 토큰을 따로 인식.

연산의 우선순위와 결합법칙(무조건 외우자.)

우선순위 : (*/,(+-)

결합법칙 : 좌에서 우로.

우선순위,결합법칙 (시험에 무조건 나온다.)

괄호 > 증감연산 (전위, 후위)> 단항 기호+ - > 사칙연산(수학) > 이 단계 까지 거치면 '값'이 도출된다. false or true 도출> 관계연산자(크기 비교) > 등가 연산자 > 논리 연산자 > if문 통째로 > assign수식 > statement 결합

assign 조차도 값을 리턴한다. 우측에 있는 값으로 리턴됨. 따라서
a= b= c= 0이 가능하다. call by value

배정연산자 예제 이해, 암기, 시험.

데이터타입의 기본

렉시컬 요약

구분할 줄 알아야 한다.
1. 키워드(32개)
2. 아이덴티파이어 > 확장 - 함수 , 변수 등등 : 이름
3. 상수> 데이터
4. 문자열 상수 >데이터
5. 연산자 > 기능적 요소
6. 구두점 > 기능적 요소

연산의 우선순위 암기하자.

조금 더 편안하게 들어보자. 내부는 어떻게 움직인다.

c언어라는 것은 시스템을 제어하기 위함이 가장 큰 존재 이유. 그러므로 하드웨어 제어를 효과적으로 해야한다. 그러면 어떤 비밀이 있는거냐?

선언
모든 변수 = identifier 모든 변수는 사용전에 선언
목적은 메모리 공간을 확보하기 위함. > 고급언어지만 하드웨어에 가장 가까운 고급언어이다.
변수 = 메모리공간 이다.
타입에 따라 +연산이 다른 **올바른 연산**을 수행한다.
int+int = int> 정수연산 float + int > 실수연산
이러한 것들이 선언에서부터 시작을 한다.

수식(데이터 처리)
메모리 공간에 있는 값들을 이용해서 일을 하는 것
상수, 변수, 연산자, 함수 호줄 등의 의미있는 결합
상수, 변수, 함수 호출은 그 자체가 수식.

배정(결과 갈무리, 저장)
= 이라는 연산은 assign으로 읽자. 배정연산자임. 우측 값을 좌측변수에 배정을 한다.
=연산자는 배정을 끝낸 뒤 우측 값을 리턴한다.

3,777;
a + b;
▶ 문법은 문제 없으나 용도가 없음. 갈무리는 안해도 문법에 문제 없으나, 용도가 없음.

정수형
char : 0~255숫자를 배정해서 문자를 표현(1byte)
signed char : -128~127
unsigned char : 0~255

short(signed)
int(signed)
long(signed)
+ 각 unsigned도 존재

실수형
float
double

산술타입
정수형 + 실수형

확장문자열

큰따옴표 : \"
탭 : \t
개행 : \n
null : \0
작은따옴표 : \'
물음표 : \?
공백은 어떻게 표현? ' ' 이거 맞는지 확인 필

int 자료형 (4byte 32비트 - 일반적, 머신마다 다르다.)

정수적형
short : 2바이트(가급적 사용 x)
long ; 4바이트

부동형 (소수점) float , double

기본적으로 접미사 없는 부동형은 double로 간주
float : 10^38~ 10-^38 ,**유효숫자 6자리(시험에 나옴)**
double : 10*308 ~ 10-^308 , **유효숫자 15자리 (시험에 나옴)**

부동형 상수의 예
0e0이 0.0과 같은 원리를 알자.
314,159e-2F는 어떻게 표현?

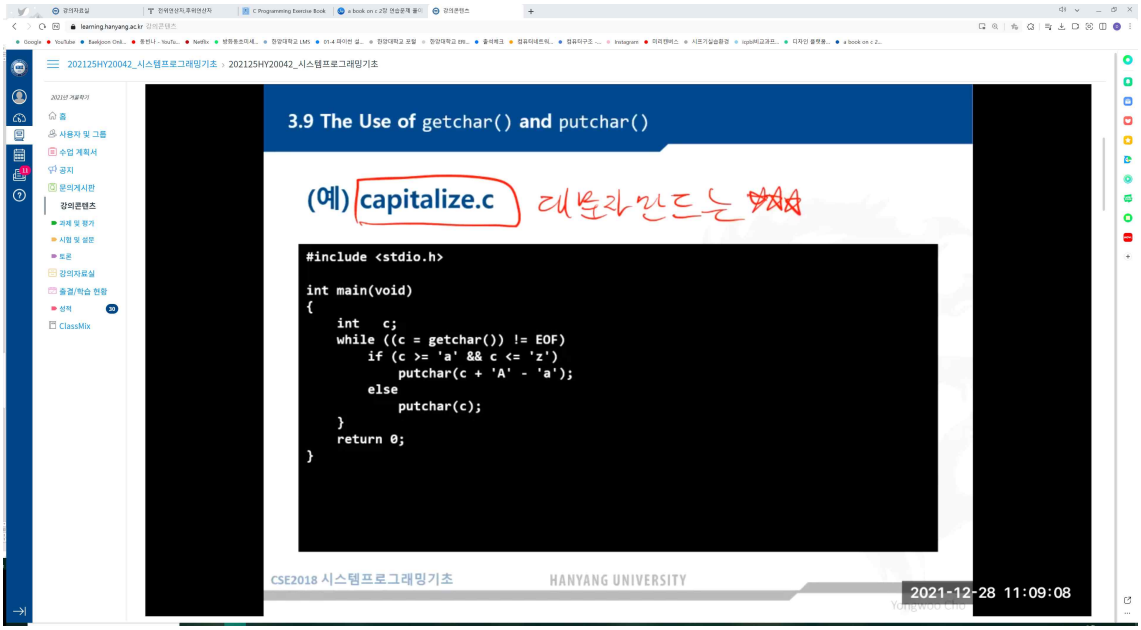
typedef의 사용 : identifier를 이용하여 새로운 데이터 타입을 정의할 수 있다.

sizeof연산자(함수(identifier)가 아님, 키워드임.)

객체 저장시 메모리 할당 바이트 수를 알기 위해 사용.

char를 어떻게 읽어오거나 출력할 지에 대한 매크로.
getchar() / putchar()

이거 비슷한 사례 시험에 나온다. 직접 코딩해보자.



수학함수

c에는 연산자(사칙연산)는 있지만 내장 수학함수가없다. 연산자를 이용해 확장해서 써라 라는 것이 c언어의 기본 철학이다. 수학 library에서 제공 <math.h>

타입캐스트

일반적 자동변환 : 수식에서 범위가 제일 큰 타입으로 변환한다.
long double >double > float
un long > long > unsigned > int 의 순서

ppt 예제 모두 이해하자.

명시적 변환
(원하는 타입)변수

맞는 예 : (double)(x=77)
틀린 예 : (double)x=77

flow of control(알고리즘이 형성하는 최소단위)

c는 겉으로 고급언어지만 기계와 상당히 가까운 언어이다.

c언어는 structured한 언어이다.
왜? 물론 마이크로 레벨에서의 컨트롤이 분명있다. c는 매크로 레벨에서 또한 컨트롤이 가능하면 스트럭쳐드 랭귀지라고한다. 기본단위는 함수다.
func + struct + pointer
>
library
>
open-source

이 전체가 c언어의 철학이다.

판단의 기준이 되는 것(분기가 되어 경로가 바뀌는 기준)
관계연산자 / 등가 연산자 / 논리연산자 : t/f가 도출된다.

관계연산자 (이항 크기 비교 > t/f 도출 / 뺄샘으로 비교 수행 / L > R)
< > <=(=<과 다른 의미다.) >=

a > b 는 (a-b) <0으로 구현
a < b < c 는 문법적으로는 맞지만 혼동할 수 있다. 왜냐하면 관계연산자는 수식을 평가하고 결과가 참이면 1(true)를 리턴하고 거짓이면 0(false)를 리턴하기 때문이다. > 이거 직접 코딩하기.

합체하는 연산

등가연산자 (이항연산자/ 뺄샘으로 비교 수행)
같다 ==(compare)
같지않다 !=

atomic sentence
논리연산자(0 또는 ~~1~무한?~~ 도출)
!(단항) && ||
논리부정연산자 : 식이 1의 값을 갖고있다면 그의 부정인 0값을 도출, 그게 아니면 역. 0이면 1. (**예외 ! 후위증감, 기호와 더불어 우선순위를 갖는다.**)

연산결과(sementics)

false : 0
true : non0

우선순위,결합법칙 (시험에 무조건 나온다.)

괄호 > 증감연산 (전위, 후위)> 단항 기호+ - > 사칙연산(수학) > 이 단계 까지 거치면 '값'이 도출된다. false or true 도출> 관계연산자(크기 비교) > 등가 연산자 > 논리 연산자 > if문 통째로 > assign수식 > statement 결합

declarations and initializations 코딩예제 부분 (시험에 나올 수 있다.)
괄호없는 상태에서 괄호쳐서 우선순위 따져보자.

```
Declarations and initializations
char      c = 'w';
int       i = 1, j = 2, k = -7;
double    x = 7e+33, y = 0.001;

Expression            Equivalent expression          Value
'a' + 1 < c           ('a' + 1) < c                  1
- i - 5 * j >= k +1   ((-i) - (5 *j)) >= (k + 1)     0
3 < j < 5             (3 < j) < 5                    1
x - 3.333 <= x + y    (x - 3.333) <= (x + y)         1
x < x + y             x < (x + y)                    0
```


단축평가
결과의 참, 거짓이 이미 판명되면 더 이상 다음 수식을 평가하지 않는다.

복합문(통째로 논리적 실행단위가 된다.)
- 중괄호로 묶여진 선언문과 실행문
- 범위(scope)를 결정하는

flow of control 분기

if 와 if-else (예약어인 if, else)

while
if+ 도돌이표

for문
while문 변형

for(expr1; expr2 ; expr3) **초기화; 판단; 카운트 증가**
        statement

expr1; **초기화**
while(expr2){ **판단**
        statement
expr3;   **카운트증가**

모든 for문은 while문으로 변형시킬 수 있다.
모든 while 문을 for문으로 변형시킬 수는 없다.

콤마연산자
expr1, expr2
expr1이 먼저 평가 후 그다음 expr2가 평가됨
프로그램상 대부분의 컴마는 연산자컴마가 아니다.

do while문

do
        statement
while(expr);
- 먼저 statement를 실행한 후, expr가 0이 아니면, do문의 시작부분으로 돌아감.
- while문은 루프 최상단에서 조건판단, do문은 한번은 실행하고 최하단에서 조건판단

판단할 때 등가수식보다는 관계수식으로 조건 평가하자. 그게 편하고 안전하다.
float형이나 double형의 식에 대한 등가검사는 컴퓨터에서 수 표현의 정확도 때문에 의도대로 작동되지 않을 수 있다.

break문
break문과 continue는 정상적인 제어의 흐름을 중단시킨다. 루프의 내부나 switch문으로부터 빠져나옴.

continue문
for, while , do 루프의 현재 반복 동작을 멈추고 즉시 다음 반복을 하게함.

switch문
모든 케이스를 조건에 맞을때까지 순차적으로 검사함. 따라서 빈번한 케이스를 처음에 놓자. 희귀한 케이스는 뒤로 갈수록 놓자.
switch 다음에 오는 괄호 안에 사용되는 제어식은 반드시 정수적형을 써라.

조건부 연산자 (시험나옴)
-삼항연산자
-작은 if문

expr1 ? expr2 : expr3
판단 ? t일 때 실행 : f일 때 실행

```
if (y < z)
        x = y;
else
        x = z;
```
▶ 아래와 같은 문장임(최소값 저장)

x = (y < z) ? y : z

function

▶ 코드의 재사용성
▶ 데이터들과 무관하게 블록처럼 조합을 통해 사용한다.

지금까지의 정리
1)어휘의 요소 - 6 token
identifier로 변수, 함수, 형 등등 확장 가능함.

2) 데이터 - 선언 > 수식 (연산 우선순위) > 배정
3) flow - 분기(연산의 우선 순위 확장)

함수는 1),2),3)을 wrap up해서 쌓는 방법에 대해 배우는 것이다.
함수호출 시 call stack에 쌓임.

함수형이 정의되지 않은 경우 :
default - int형으로 간주함

지역변수와 전역변수
함수블록 안에 선언된 변수 - callstack에서 그 함수블록 내에 정의
함수 외부에 선언된 변수 - 처음 시스템을 실행하는 첫 순간에 bss라는 공간에 할당해서 모두가 참조할수 있으며 프로그램전체가 끝나서 main함수가 리턴하는 순간까지도 살아있다.

리턴
-return_statement ::= return; | return expression(수식);
-float형 리턴 함수에서 int선언 변수 리턴시 함수 반환값으로 변환이된다.
-받은 리턴값을 사용하지 않아도된다.
-return문에 도달하면 해당 함수 종료

함수원형(선언에 필요)
tpye function name(parameter); - 바디없이.
정의만 하고 아직까지 없는 '가상화' - 함수원형을 이용하는 방법이다.(찾아보기)
식별자는 원형에 영향 없음(생략 가능)
void f(char, int);
컴파일러에게 인자type, 인자 수 , return갑의 형을 알림
컴파일러 관점에서의 함수 선언

함수정의 순서의 다른 형태

-하향식 함수 작성 : **함수의 정의가 원형역할을 수행** 할 수 있도록 하는 작성방법.
-피호출 함수 정의를 먼저 작성.

c는 값에 의한 호출이다.
c로 call by reference효과를 만들 수 있다! 라는 표현을 쓴다. > 이 표현이 헷갈리게 만든다
.
함수가 스택에 쌓이고 결국은 값만 돌려준다. 함수상자로 생각하면 쉽다. value를 넣고 value를 뱉는다.

call-by-reference라는 건 함수에 포인터를 파라미터로 전달을 했을 때 손가락으로 가르키는 내용물을 함수 내에서 값을 변경하면 실제로 포인터로 가르키는 내용물의 값이 변경값으로 적용이되는 것을 말한다.
▶ 원칙적으로 call by value로 밖에 지원을 안한다.

포인터는 주소를 가르키는 손가락이다. 주소가 아니다. 얼마든 내용물이 가변적일 수 있다.

call back함수란?

==========

대형프로그램의 개발

대형프로그램을 서로 나누어 개발할 때 pgm.h에 프로그램에 쓰일 모든 함수의 원형을 담아둔다. 그리고 각 부분프로그램은 #include “pgm.h”를 하여 정의 되어있는 인터페이스 때문에 함수 호출 가능하다.

다음과 같이 컴파일 할 수 있음
gcc -o 실행가능한 프로그램 명 c파일1.c c파일2.c c파일3.c **(링킹)**
▶ 컴파일러는 세게의 .c파일을 컴파일하고 세 개의 .o파일로 구성된 하나의 실행 가능한 pgm파일을 생성

단정
대형 프로그램은 안정성이 필요함. 안정성을 담보해야함.
assetion(사전) / exception handle(사후)

#include <assert.h>
assert(조건) - **조건에 부합하지 않으면 프로그램 종료.**

변수의 유효범위 규칙

유효범위 : 변수나 상수가 유효하게 정의되는 범위
범위 효력이 정해지는 시점 기준
1) 정적유효범위 : 컴파일 시간에 결정(프로그램 구조에 의해) > **c에서는 바뀌지 않는다.**
2) ~~동적유효범위 : 실행시간에 결정(실행순서에따라)~~

유효범위의 규칙
-기본적 규칙은 변수가 선언된 블록 안에서만 그 변수를 이용 가능.
-외부 블록 변수는 내부에서 다시 정의하지 않는 한 여전히 유효함
-만약 내부블록에서 해당 식별자가 다시정의되면 외부변수는 내부 블록으로부터 숨겨진다.
(메모리가 새로 할당)

scope rule(예제코드 봐두기, 메모리블럭을 그려보면서 이해하자) 무조건 시험에 나온다.
a=b 부분은 a를 다시정의가 된게 아닌 배정연산이다.

```
{
    int a = 1, b = 2, c = 3;
    printf("%3d%3d%3d", a, b, c);          /* 1 2 3 */
    {
        int b = 4;
        float c = 5.0;
        printf("%3d%3d%5.1f", a, b, c);    /* 1 4 5.0 */
        a = b;
        {
            int c;
            c = b;
            printf("%3d%3d%3d", a, b, c);  /* 4 4 4 */
        }
        printf("%3d%3d%5.1f", a, b, c);    /* 4 4 5.0 */
    }
    printf("%3d%3d%3d", a, b, c);          /* 4 2 3 */
}
```

변수 속성
1. 타입
2. 기억영역 클래스(기계적 하드웨어부분이다.) **auto extern register static**

기억영역클래스 auto

함수의 몸체부분에서 선언된 변수는 auto다. 이는 콜스택에 생겼다가 없어진다. 자동으로 관리가 된다. 굳이 auto라고 안적어도 된다. 리턴해도 남겨놔야하는 애들이 있긴하다.
지역변수들은 블록을 빠져나가게되면 모두 메모리에서 제거. 따라서 변수에 저장하고 있던 값 잃음.

기억영역클래스 extern
전역변수와 비슷한 개념, 전역변수로 취급이 된다. 인터페이스사용처럼 자기가 안쓰는 변수 끌어다 쓸 때 쓰인다.

함수간 정보전달 두가지 방법
1, 파라미터
2. 외부변수(extern)

외부변수 부작용 : 사용이 용이하지만 누가 언제 바꿔서 지금 어떤 상태인지 왜 그런지 아무도 모른다. **(가급적 최소화하라.) >> 좋은방법: 구조체 , 포인터사용**

매개변수 매커니즘
-코드의 모듈성 향상
-원치 않은 부작용 가능성 줄임

기억영역클래스 register

변수를 가능하다면 고속메모리인 레지스터에 저장되도록 지시(cpu machine cycle에 개입)
한정된 자원으로 인해 가급적 잠시쓰고 치고 빠져야한다.

기억영역클래스 static

auto가 아니라는 얘기. (오토는 리턴때 반납한다.)
static은 반납안함. 콜스택이 아니라 다른영역에 alloc이 된다.
기본용도 : 그 블록으로 다시 들어갈 때 지역변수로 선언된 변수가 **이전의 값을 유지하고 있게 하는 것(콜스택에서 리턴시 반환하지 않고 다른 영역에 그대로 그 이전의 값을 유지한채로 남아 있기 때문이다.)**

정적외부변수(static + extern)

모듈화에 있어서 매우 중요한 개념인 ***비공개를 제공한다.***
가시화 또는 유효범위를 제한할 수 있다.
제 3자가 못건들인다!
다른 애들은 참조는 할 지언정 값을 건드리진 못한다.

-정적 외부 구조물은 프로그램 모듈화에 있어서 매우 중요한 개념인 비공개를 제공함
-비공개란 변수나 함수의 가시화 또는 유효범위의 제한을 의미함
-정적 외부 변수의 유효 범위는 자신이 선언되어있는 원시 파일의 나머지 부분임
-다른 파일에서 선언된 함수가 키워드 기억영역 클래스 extern을 사용하여 그 변수를 사용하고자해도 사용할 수 없음

초기화

외부변수, 정적변수는 초기화하지않아도 시스템에 의해 0으로 초기화된다.
~~**이와 같은 방식으로 초기화 되는 것에는 배열, 문자열, 포인터, 구조체, 공용체가 있다.(무슨의미?)**~~
반면 *오토, 레지스터변수*는 시스템에 의해 초기화 *되지않는다.*(쓰레기값 가질 수도 있다.)

재귀함수
함수가 자기 자신을 호출.

가장 중요한 것은 stop조건(base조건을 설정- return)
두 번째로는 일반적인 경우 함수 디자인

1. 변수를 검사하여 기본적인 경우인지 일반적인 경우인지를 결정
2. 기본적인 경우일 때에는 더 이상 재귀 호출을 하지 않고 필요한 값을 리턴
3. 일반적인 경우일 때에는 그 변수의 값이 결국에 기본적인 경우의 값이 될 수 있게 하여 재귀 호출

배열,포인터, 문자열

배열은 기본자료형이 아니다.

배열, 포인터 그리고 문자열

~~지금까지 배웠던 내용 정리해보자.~~
~~어휘원소 - 토큰 (키워드 32개, identifier 변수, 함수 구조체 확장가능, 상수, 문자열 상수, 연산자 ,구두점)~~

~~괄호 > 전위증감연산자 > 단항기호 >후위> 부정논리 > 사칙연산 > 이러면 결과값도출 > 관계연산자 > 등가연산자 > 논리연산자 > 판단 > if통째로 > 대입연산자 >~~

~~함수 : call by value . library도 만들 수 있다.~~

배열
같은 자료형의 자료를 여려개 생성.
자료형 변수명[원소개수]
배열원소의 첨자는 0부터 시작.

메모리에 연속된 공간을 잡아 해당하는 자료들을 묶어서 나열.
시작점부터 몇칸 떨어져있냐?를 따져보면된다.
첫 번째원소는 0칸 n번째 원소는 n칸 떨어져있기 때문에 첫원소는 a[0]이다.

lower bound =0
upper bound = SIZE-1
SIZE = upper bound +1

배열초기화
자료형 배열이름[원소개수] = {a1. a2. a3. a4. a5 };

초기화 리스트가 배열원소개수보다 적으면 나머지는 0으로 초기화.
ex)
int a[100]= {10};
a[0]만 10으로 , 나머지 1~99는 0으로 채움.

배열명 = &배열명[0].

동적할당? 숫자를 가변적으로 할당하고싶을 때.
int a[] = { 2, 3, 4, 5} ;
> 원소 4개 할당.

문자배열에도 적용된다.
char s[] =“abc”;
char s[] = {‘a’,‘b’,‘c’,‘\0’};
▶ **s배열의 크기는 3이 아니라 4다! null값으로 문자열의 끝을 표시.**
▶ 스페이스바는 1바이트에 포함되나?

첨자
a가 배열이면, a의 원소를 접근할 때 a[expr]로 쓴다.
선언한 범위는 프로그래머가 책임져야한다. (boundary check)
범위를 벗어났을 때의 효과는 시스템마다 다르다. : 대체로 런타임오류 발생

포인터

포인터는 주소가 아니라, 저장주소를 가르키는 **손가락이다.**

실제로 주소를 알기위한 방법은 무엇이있나?
▶ 주소연산자(&) - address of 변수명 이라고 읽자.
▶ 주소표현을 위한 수단일뿐, 실제 주소값을 신경쓸 필요 없다.

포인터변수
-주소를 값으로 가지는 변수다.
-본질은 - 대상을 손가락으로 가리킴. 손가락으로는 대상을 옮겨 가리킬 수 있다.

선언방법
int *p;　　정수형 포인터 p 혹은 *p 자체가 정수형. 후자로 해석하면 나중에 편하다.
-int형 변수의 주소를 가지는 포인터 변수.

포인터 범위
- 특수주소 0(null)과 주소공간에 있는 양의 정수.

포인터 변수 배정 예제

int *p;
p=0;
p= NULL; /* p=0과 같다. */
p=&i ; I의 주소를 갖는 포인터 변수.
p = (int*) 1776; /메모리 내의 절대적 주소값/

역참조 연산자

*
-값을 다루는 연산자.
-&의 반대 연산자
-value of 변수명 이라고 읽자.

포인트 메커니즘

int a=1, b=2, *p;
주소 = 값; 이라고 이해하자.
a라는 메모리가 생기고 거기에 1을 할당한다. (주소에 값을 할당한다.)

p = &a;
▶ p : a의 주소를 가리킴
▶ *p : a의 주소에 있는 변수값

b = *p;
▶ b=a; == b =1;

시험에 나옴/ 코딩해보자

지정할 수 없는 구조

상수 , 일반식, register변수 에는 주소값을 참조할 수 없다.

```
Declarations and initializations
int     i = 3, j = 5, *p = &i, *q = &j, *r;
double  x;

Expression              Equivalent expression          Value
p == & i                p == (& i)                     1
* * & p                 * (* (& p))                    3
r = & x                 r = (& x)                      /* illegal */
7 *  * p / * q + 7      (((7 * (* p))) / (* q)) + 7    11
* (r = & j) *= * p      (* (r = (& j))) *= (* p)       15
```

*int j = 5, i = 3, * p = &i, * r;*

**(r = &j) *= *p;*

*printf("%d₩n%d₩n%d", *r, j, *p); // 15, 15, 3*

size of pointers 코딩해보자.

```
#include <stdio.h>

int main(void)
{
    printf("size of char pointer : %d\n", sizeof(char *));
    printf("size of short pointer : %d\n", sizeof(short *));
    printf("size of int pointer : %d\n", sizeof(int *));
    printf("size of long pointer : %d\n", sizeof(long *));
    printf("size of signed pointer : %d\n", sizeof(signed *));
    printf("size of unsigned pointer : %d\n", sizeof(unsigned *));
    printf("size of float pointer : %d\n", sizeof(float *));
    printf("size of double pointer : %d\n", sizeof(double *));
    printf("size of longdouble pointer : %d\n", sizeof(long double *));
    return 0;
}
```

==========
요약
포인터 : 뭔가를 가르키는 손가락.

실제 사례를 알아보자.
call-by-value
함수에 대해 배웠었다. in >out 하는 것. 이는 둘 모두 value로 이루어졌었다.
value-정수값(문자), 실수값, array, **포인터도 value**로서 줄 수 있다.

포인터가 가르킬 수 있는 것
-정수변수(4바이트)
-실수변수
-또다른 포인터
-함수 그 자체
-다른 시스템들 내에 있는 것들
콜백? 인터룹트핸들러?

참조에 의한 호출 : 함수 파라미터에 포인터를 전달.
(함수파라미터도)포인터도 콜스택메모리에 할당되나?

call by reference의 효과를 얻는 방법 (value: 주소값 > 찾아가서 참조는 실제 값을 **참조**)
1. 함수 매개변수를 포인터형으로 선언
2. 함수 호출 시 주소를 인자로 전달
3. 함수 구현에서 역참조 포인터 사용

배열과 포인터의 관계
배열이름은 포인터의값, 주소이다. 차이는 **배열 이름은 옮겨 다니면 안된다. 고정되어 있어야** 한다. **배열과 포인터에는 둘 다 첨자를 사용할 수 있음.**
배열이름은 고정된 주소 또는 (상수)포인터임.
배열명 = &배열명[0]

배열이름이 a 일 때, 포인터 p,q할당시

p = a; // p = &a[0] , a[0]은 변수이다. a와 &a[0]랑 같은 녀석

q = a+3; // *몇칸 떨어져있냐? 라는 연산자. 포인터 연산이다. 덧셈이 아니다!!!!!*

q = &a[3]

a와 p는 포인터이고 둘 다 첨자를 붙일 수도 있다.

a[i] == *(a+i)

p[i] == *(p+i) : 포인터도 배열 형태로 사용가능하다.

p= a+i; (가능한 선언.)

a =q; (잘못된 선언. a는 고정된 주소이다. 변경 불가능)

포인터 연산과 원소 크기

+가 결코 덧셈이 아니다!! 포인터를 옮기기 위한 연산자이다!!

p+n // n칸 이동.

p와 q가 모두 한 배열의 원소들을 포인트하고 있다면, p-q는 p와 q사이에 있는 배열 원소의 개수를 나타내는 int값 생성.

포인터 수식과 산술 수식은 형태는 유사하지만, 완전히 다름!!!!

연산자도 변수마다 올바른 연산형태가 다르댜. int용 연산, double형 연산 등등 포인터용 연산도 완전히 다르다.

함수 인자로서의 배열

함수의 인자로 배열이 전달되면, 배열의 기본 주소가 **call by value로 전달**됨.

배열 원소 자체는 복사되지 않음. 원래 배열을 **참조**할 뿐. 포인터가 스택에 할당되나봄

배열크기는 compile전에 고정해놔야한다.(**static**)　: **call stack 할당**
나는 실행 도중에 마음껏 배열의 공간을 할당하고싶다.
이거 어떻게 하냐?
calloc과 malloc : 실시간 동적으로 (**dynamic**)으로 할당/ stdlib.h에 정의,실시간으로 동적할당되는 배열이다. 라고 생각

각원소의 크기가 el_size인 n개의 원소를 할당
calloc(n, el_size); 정형화된 크기(배열과 거의 같다)
malloc(n* el_size); 자유롭게

calloc에서는 모든 원소를 0으로 초기화하는 반면, malloc은 하지 않음 - heap공간에 할당
▶ **heap공간은 내가 관리를 해줘야한다. 뒷정리;. free()사용해줘야한다. call stack(정적할당 배열)은 자동으로 할당받은 메모리를 반환하지만 heap공간에서는 그런거 없음.**

문자열(문자열 또한 배열이므로 포인터랑 똑같이 쓸 수 있게된다.)
char형의 1차원 배열
문자열은 끝의 기호인 \0, 또는 널문자로 끝남.
널문자 : 모든 비트가 0인 바이트 ; 십진 값 0
문자열의 크기는 \0까지 포함한 크기

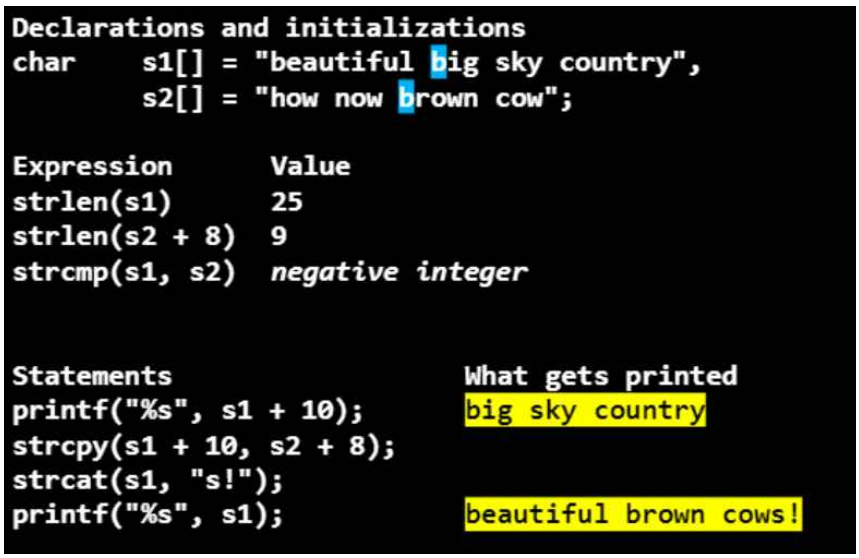
주의!!!! "a"와 'a'는 다르다. "a"는 문자열이므로 \0이 뒤에 붙어있는 배열. 2바이트짜리이다.
컴파일러는 문자열 상수를 배열이름과 같이 포인터로 취급한다.

문자열포인터 예제
```
char *p = "abc  ";
printf("%s %s\n", p, p + 1); /* abc   bc  is printed */
```
→변수p에는 문자 배열"abc"의 기본 주소가 배정
→**char 형의 포인터를 문자열 형식으로 출력하면, 그 포인터가 포인트하는 문자부터 시작하여 \0이 나올 때까지 문자들이 연속해서 출력됨**

문자열

- "abc"와 같은 문자열 상수는 포인터로 취급되기 때문에 "abc"[1] 또는 *("abc" + 2)와 같은 수식을 사용할 수 있음

6.11슬라이드 코드예제 숙제.



```
Declarations and initializations
char    s1[] = "beautiful big sky country",
        s2[] = "how now brown cow";

Expression      Value
strlen(s1)      25
strlen(s2 + 8)  9
strcmp(s1, s2)  negative integer


Statements                  What gets printed
printf("%s", s1 + 10);      big sky country
strcpy(s1 + 10, s2 + 8);
strcat(s1, "s!");
printf("%s", s1);           beautiful brown cows!
```

다차원배열
아파트만 생각하자.

2차원배열
사실 원소들은 하나씩 연속적으로 저장되지만
행, 열을 갖는 직사각형의 원소의 집합으로 생각하자.

선언 : int a[3][5];
a[n]에서 0칸 1칸 2칸 3칸 4칸 떨어진 위치 이렇게 생각하자.

2차원 배열과 같은 표현들
**(ai +j)*
((a+i))[j]*
**((*(a+i)) + j)*
**(&a[0][0] + 5 사상함수*

기억장소 사상 함수(시험)
- 배열에서 포인터 값과 배열 첨자 사이의 사상(맵핑)
- 예 int a[3][5];
- 배열 a의 a[i][j]에 대한 기억장소 사상함수
: *(&a[0][0] +5 *i +j);
한층당 5칸 이동이므로 + 5 이다.

형식매개변수 선언
-함수 정의에서형식 매개변수가다차원 배열일 때, 첫 번째 크기를제외한 다른 모든 크기를 명시해야함 - 기억장소사상 함수를 위해

3차원 배열 (동, 층, 호)
a[i][j][k]를위한 기억장소사상 함수:*(&a[0][0][0] + 9 * 2 * i + 2 * j + k)

==========

여기서부터 직접해보면서 따로 복습 필!!!

다차원 배열 초기화 방법
int a[2][3] = {1, 2, 3, 4, 5, 6};
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
int a[ ][3] = {{1, 2, 3}, {4, 5, 6}};
→내부 중괄호가없으면, 배열은a[0][0], a[0][1], ..., a[1][2] 순으로 초기화되고, 인덱싱은행 우선 임
→배열의 원소 수보다 더 적은 수의 초기화 값이 있다면, 남는 원소는0으로 초기화됨
→첫 번째 각괄호가공백이면, 컴파일러는내부 중괄호 쌍의 수를 그것의 크기로 함
→첫 번째 크기를 제외한 모든 크기는 명시해야함

초기화 예
int a[2][2][3]={{{1, 1, 0}, {2, 0, 0}},{{3, 0, 0}, {4, 4, 0}}};
▪이것은 다음과 같음
int a[][2][3]={{{1, 1}, {2}}, {{3}, {4, 4}}};
▪모든 배열 원소를0으로 초기화 하기
int a[2][2][3] = {0};/* all element initialized to zero */

포인터 배열
▶ 배열의 원소의 형은 포인터형을 포함하여 임의의 형이 될 수 있다.
▶ 포인터 배열은 **문자열을 다룰 때 많이 사용**

main()함수의 인자

래기드배열 둘의 차이 식별자 a 와 p의 차이
> 임의의 장소에 접근할 때 기억장소 사상함수 사용 유무
> 공간할당크기
> p[0][3] 는 가능?
> 래기드배열의 p[0], p[1]은 배열을 가리키는 피켓이다. 변경불가능!!!!

인자로서의 함수
>함수의 포인터가 함수를 가르키고, 파라미터로 전달할 수 있다. 배열도 만들 수 있다.

인자로서의함수
▪포인터f를 함수처럼 취급할 수도 있고, 또는 포인터f를 명시적으로 역참조 할 수도 있음
→즉, 다음 두 문장은 같음
sum += f(k) * f(k);
sum += (*f)(k) * (*f)(k);

함수명 자체가 포인터처럼 쓸수 있는건가? 배열명이 첫시작 피켓이듯 함수명 또한 함수시작주소를 나타낸다.

const
▶ assign금지.
변수가 const로 한정된다 해도, 다른 선언에서 배열의 크기를 명시하는 데는 사용될 수 없다.
const int k =3;
int a[k]; >> 안됨!!!! 문법상 금지. 컴파일러는 이를 변수로 받아들임. 금지.

1)
const int n = 3;
int *p = &n;
상수를 보통의 포인터가 가르키면 안된다. 포인터 이용해서 값을 변경할 수 있으므로.

2) 이렇게 하면 *p를 변경불가.
const int n = 3;
const int *p = &n;
이는 p에 다른 주소를 배정할 수 있지만, *p에 값을 배정할 순 없다.

int * const p = &a;
이는 포인터가 다른 주소로 옮겨다닐 수 없다.

volatile : 하드웨어에 의하여 어떤 방법으로 수정될 수 있다. extern한정자이다. 하드웨어는 변경 가능 but 코드로는 변경 금지.

비트연산과 이뉴머레이션 타입

배열 : a[2]의 의미 : *(a+2) a란 피켓부터 2칸 옆의 친구
포인터 : 대상을 가르키는 손가락, 표현하는 방법 : 주소값
문자열 : 문자로 이루어진 배열> 결국은 포인터
사상함수, 래기드배열

c는 기계를 다루는 최전방 언어.
c는 빼기가 없다. 음수는 2의 보수표현법을 사용하기 때문에.

비트단위 연산자
-이진숫자의 문자열로 표현된 정수적 수식에 사용
-시스템 종속적
-거의 표준에 가까운 것들을 가정을 해서 배운다. 8비트 바이트, 4바이트 워드, 2의 보수로 표현되는 정수, 아스키 문자코드를 갖는 컴퓨터를 가정한다.

논리곱 : 까다로운
논리합 : 너그러운
배타적 논리합?
왼쪽 이동 :
오른쪽 이동 : ‘

signed int의 음수 표현

▶ 부호비트 (부호절대값 방법)

미리빼놓는 개념 5+(-3)
▶ 1의보수
-
▶ 2의보수

비트단위 보수 : **~연산자 사용**

▶ 1의 보수 연산자.(통째로 반전시키자.)
▶ 2의 보수는 이에 +1만 해주면 된다.

signed int의 음수표현: w비트로 표현가능한 범위

논리곱 : 둘 모두 참일 때 참 반환
논리합 : 하나라도 참이면 참반환
배타적논리합 : 논리합이되 모두 참인 경우 빼고 참 반환

```
Declaration and initializations
   int a = 33333, b = -77777;

Expression   Representation                      Value
a            00000000 00000000 10000010 00110101   33333
b            11111111 11111110 11010000 00101111   -77777
a & b        00000000 00000000 10000000 00100101   32805
a ^ b        11111111 11111110 01010010 00011010   -110054
a | b        11111111 11111110 11010010 00111111   -77249
~(a | b)     00000000 00000001 00101101 11000000   77248
(~a & ~b)    00000000 00000001 00101101 11000000   77248
```

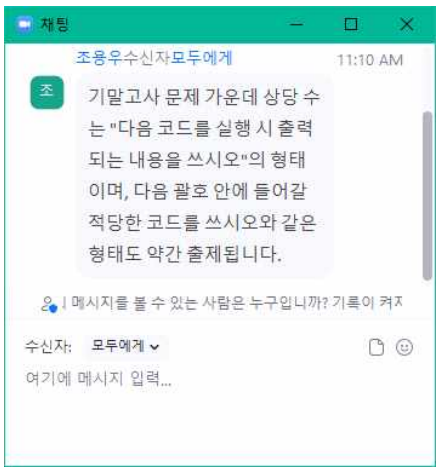
오른쪽 이동연산자
-부호가 없는 정수적 수식에서는 상위 비트로 0이 들어옴
-부호가 있는 형 일때에는 시스템에 따라 상위 비트로 0이 들어오는 것도 있고, 1이 들어오는 것도 있음

```
Declaration and initialization
    unsigned  a = 1,  b = 2;

Expression              Equivalent expression       Representation      Value
a << b >> 1             (a << b) >> 1               00000000 00000010   2
a << 1 + 2 << 3         (a << (1 + 2)) << 3         00000000 01000000   64
a + b << 12 * a >> b    ((a + b) << (12 * a)) >> b  00001100 00000000   3072
```

마스크 : 다른 변수나 수식으로부터 원하는 빝느를 추출하는데 사용되는 상수나 변수

열거형



전처리기

1의 보수는 0,1 뒤집기 : 미리 빼놓는 다는 뜻임.
2의 보수는 1의 보수+1

structure는 데이터의 확장

-c를 도와주는 조력자 **(c문법이 아니다.)**
-전처리기는 c를 알지 못함.
-**#으로 시작하는 행을 전처리 지시자라고 한다. (컴파일러가 보지않을 내용)**
: 컴파일 전에 전처리기가 처리.

#include ... :한줄을 파일과 통째로 바꿔라.
#define
#if
#error

#include
-지정된 파일을 읽어들임
-두가지형태로 파일을 표현
#include "filename" : **먼저 현재 디렉토리(내가 정의한 헤더)에서 검색**하고, 거기에 없다면 시스템 정의디렉토리에서 검색
#include <filename> : **시스템이 정의한 디렉토리에서만** 검색함

#define
-*글자 그대로* 쓰겠다.
-#define *identifier t_string(opt)*
id 와 t_string을 substitude 대치하기 위함.
- 프로그램의 **명확성과 이식성**(아키텍쳐 별로 코드 운용가능)을 높일 수 있다.
- 프로그램의 문서화에 도움을 줌, 가독성을 높임
- 시스템에 따라 달라지는 상수를 한번에 변경할 수 있으므로 이식성을 높여줌
- 상수의 reliability를 검사하는데 한곳만 검사하면 되므로 신뢰성 높여줌 **(자주 타이핑할때의 오타방지할 수 있으므로)**

ex) #define *SECONDS_PER_DAY (60*60*24)*
-수식으로 대치가 아님. 글자 그대로와 교체. "(60*60*24)"자체와 교체.

#define EOF (-1)
/* typical end-of-file value */

#define MAXINT 2147483647 /* largest 4-byte integer */
(32비트용 int 최대값) (8비트용 int 127)

> 머신별로 define만 달리 해주면 된다.
> 아키텍쳐가 바뀌어도 코드를 그대로 쓸 수 있다. define만 따로 해주면됨.

#define
-c의 구문을 사용자의 취향에 맞게 변경하는 것이 기술적으로는 가능.(비추.)
ex_) 논리 수식에서 ==대신 EQ사용
#define EQ ==

인자를 갖는 매크로
▶ 반복된 코드
▶ 퍼포먼스 향상 (콜스택을 만들지 않기 때문에)
▶ 마치 함수처럼 인자를 가지게 한다.

#define identifier(id_1,..., id_n) token_stringopt
→첫번째 identifier와 왼쪽 괄호 사이에는 공백이 없어야함
→매개변수 목록에는 식별자가 없거나 또는 여러개가 올 수 있음

ex) #define SQ(x) ((x) *(x))
-수식으로 대치가 아님. 글자 그대로와 교체. "((x) *(x))"자체와 교체.

인자를 갖는 매크로-유의사항
-매크로 정의 시 올바른 평가 순서를 유지하기 위해 괄호를 적절히 사용해야 함.

ex) 부적절예시- 괄호를 의도와 맞게 적절히 사용해야 하는 경우
#define SQ(x) x * x: SQ(a + b) ==> a + b * a + b ≠ ((a + b) * (a + b))
#define SQ(x) (x) * (x): 6 / SQ(2) ==> 6 / (2) * (2) ≠ (6 / ((2) * (2))

ex) 부적절예시 - 띄어쓰기 때문에 의도와 다르게 대치되는 경우
#define SQ (x) ((x) * (x)): **띄어쓰기하면 안됨. SQ를 (x) ((x) * (x))로 바꿔버림.**
SQ(7) ==> (x) ((x) * (x)) (7)
#define SQ(x) ((x) * (x)); /* error */: **쎄미콜론 넣으면 안됨./ c의 문법 x**
if (x == 2) x = SQ(y);else ++x; ==> if (x == 2) x = ((y) * (y));;else ++x;

#undef는 기존 매크로 정의를 무효화함, 그리고 경우에 따라 재정의 할 수 있다.

전처리기 결과 보기

cc -E file.c

▶ 전처리기가 작업을 수행한 다음에 더 이상의 컴파일이 일어나지 않음.(전처리만 일어남)

1바이트 짜리 처리하는데 function을 사용하면 처리 될 때마다 콜스택을 부른다.(calling overhead가 커짐)

▶ **이를 매크로로 바꿔주면 functoin call없이 해당일을 수행**한다. (빠르게)

▶ 문자, 버퍼 처리 시 매크로를 자주 사용한다.

stddef.h

이 헤더파일은 다른 곳에서 공통적으로 사용되는 몇 가지 형 정의와 매크로를 포함하고 있다.

stdio.h의 매크로

getchar() putchar()는 함수가 아니라 매크로다.

```
#define getchar() getc(stdin)
#define putchar(c) putc((c), stdout)
```

조건부컴파일

조건부컴파일을 위한 지시자

#if constant_integral_expression

...*조건 맞을 경우 이부분 코드에 포함*

#endif

토글

#ifdef identifier// or #if defined(identifier)

...

#endif

#ifndef dentifier : not define

...

#endif

조건부컴파일 예제 - 디버깅을 위해 사용 할 수있음

1,

```
#define DEBUG 1                    //0으로 바꾸면 한방에 디버깅모드를 해제할 수 있다.
…
#if DEBUG
        debugging code
#endif
```

2.

```
#define DEBUG
…
#ifdef DEBUG
        debugging code
#endif
```

매크로이름이 중복지정 되는 것을 피하기 위해 사용 할 수있음

```
include "everyting.h“
undefPIE
define PIE "I like apple.“
.....
```

눈에 익혀두자!!!!!!!

미리 정의된 매크로

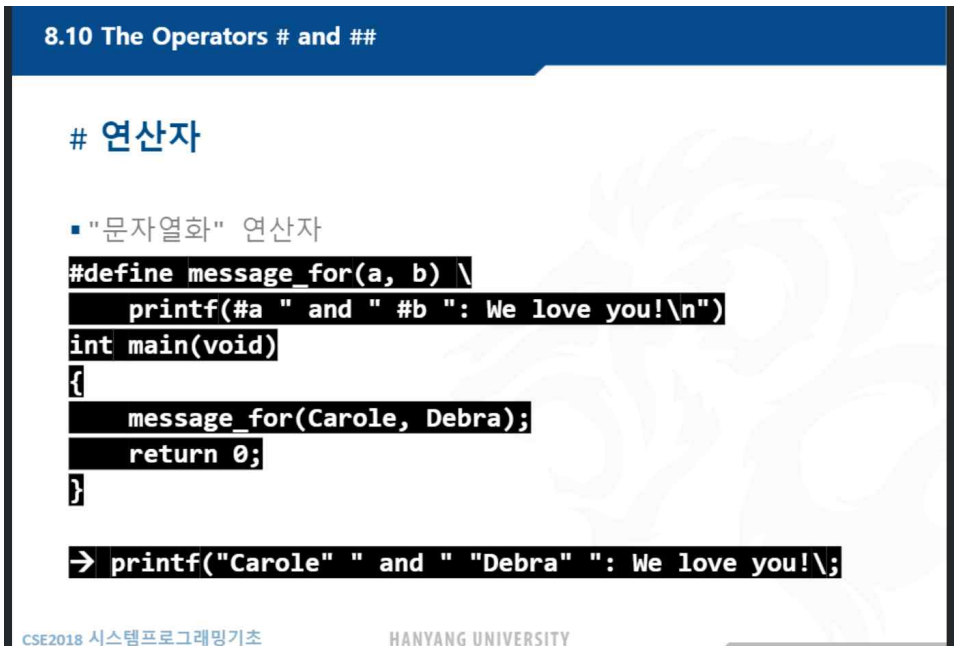
미리 정의된 매크로	값
__DATE__	현재 날짜를 포함하는 문자열
__FILE__	파일 이름을 포함하는 문자열
__LINE__	현재 라인 번호를 나타내는 정수
__STDC__	ANSI C 표준을 따르는 경우 0이 아닌 값을 가짐
__TIME__	현재 시간을 포함하는 문자열

#연산자(강의 다시 복습)

▶ 문자열화 연산자

문자열로 취급

#a // “a”



##연산자

토큰 결합 연산자, 스페이스바를 없애준다는 의미 같음.

#define X(i) x ## I

X(1) = X(2) = X(3);

> 전처리기수행후의결과:

x1 = x2 = x3;

assert()매크로

▶ 사고상황을 막기위한 브레이크목적 매크로

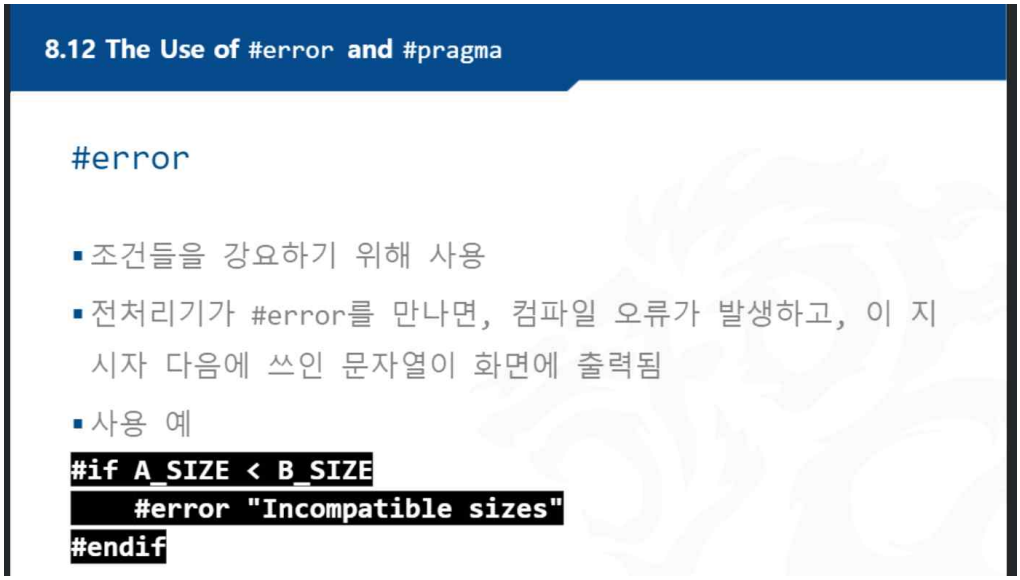
▶ 매크로라서 빠르다. 디버그때는 포함이 되지만 배포때는 미포함

▶ NDEBUG가 정의되어있으면, 모든 단정은 무시됨(디버깅모드가 해제돼있을 때)

▶ 예외 상황시 시스템이 멈추기 전에 프로그램이 멈춤.

#error

전처리기가 #error를 만나면, 컴파일 오류 발생, 이 지시자 다음에 쓰인 문자열이 화면에 출력, if구문과 함께 사용.



행번호(복습)

대응함수

매크로 대신 시스템콜 함수를 사용하기 위해서는 다음과 같이함

→방법1

-함수 사용 전에 다음과 같은 행을 삽입함

#undef isalpha

→방법2

-다음과 같이 사용자정의 함수를 호출함/ 괄호로 함수명 감싸서 호출

(isalpha)(c)

구조체와 공용체

구조체 - data확장

c언어의 확장 방법
-매크로와 라이브러리
-사용자 정의형 (배열, 구조체, 공용체)

구조체
-**서로 다른 형의 변수**들을 하나로 묶어주는 방법

ex)예제 - 카드
▶ 각 카드는 고유의 무늬와 숫자를 가진다. 구조체를 사용하여 표현하면 효율적이다.

구조체 선언 예시

1)

```
struct card {     //card가 사용자 정의형 변수가 된다.
int pips;         //  숫자
char suit;        //  무늬
};
```

▶ 이는 struct card형의 정의이고, 변수선언은 아니다

```
struct card c1, c2; // 따로변수선언
```

.
struct :키워드
card : 구조체 태그 이름
pips, suit : 구조체 멤버

문법이 익숙하지 않을 수 있어 혼돈할수 있다. 기존 변수들과 혼돈가능
이 선언들을 편리하게 typedef를 이용하여축약시킬 수 있다. 이는 우리에게 혼돈을 준다. 이에 익숙해져야한다.

2) struct 변수명은 다시말하자면 {~}의 내용인데,
개의 실제 변수는 a1, a2 ~~~~;이다. 라고 해석하자.

```
struct card {     //card가 사용자 정의형 변수가 된다.
int pips;         //  숫자
```

```
char suit;        //  무늬
}; c1, c2;        //  변수 c1,c2 선언
```

3)

```
struct card {     //struct card가 사용자 정의형 변수가 된다.
int pips;         //  숫자
char suit;        //  무늬
};
typedef struct card card;     // card썼는데 또 쓸수 있는거야?
                                 struct 로서 card라서 struct card가 id1이고 typedef의
                                 card는 그냥 id2일뿐이다.
card c1, c2;      //  변수 c1,c2 선언
```

4) 현실적으로 이게 가장 흔한 MINIMUM 정의이다.

typedef하되 무명인 struct{}에 대해 card라고 명하자.

```
typedef struct {     //무명인 struct을 typedef
int pips;         //  숫자
char suit;        //  무늬
};  card          // card로 정의
card c1, c2;      //  변수 c1,c2 선언
```

5)

다시는 선언하지 않을 때 사용.

```
struct {     //  무명인 struct
int pips;         //  숫자
char suit;        //  무늬
}; c1, c2;        //  변수 c1,c2 선언
```

6)

다시는 선언하지 않는다고 다짐하고 다시 사용하고 싶을 때 밑에처럼 했다면,
C1,C2 와 C3,34는 다른 형이다.

```
struct {     //  무명인 struct
int pips;         //  숫자
char suit;        //  무늬
}; c1, c2;        //  변수 c1,c2 선언
```

```
struct {     //  무명인 struct
int pips;          //  숫자
char suit;         //  무늬
}; c3, c4;         //  변수 c1,c2 선언
```

요약

1. struct 변수명 정의하고 변수를 struct변수형으로 따로정의하는 방법
2. 편리하라고 변수 따로 정의하지말고 struct정의 때 마다 변수선언하는 방법
3. 변수선언때마다 struct선언해야하니까 그러지말고 typedef형으로 변수선언 편리하게
4. 어차피 struct 변수명 typedef로 대체할거니까 무명인 struct선언하고 typedef형 변수선언
5. 무명인 struct사용하는데 한번쓰고 말거니까 typedef없이 무명 struct선언시 변수만 선언
6. 만약 다시 쓰고 싶어서 같은 무명 struct{내용}선언하고 다른 변수 선언하면 선후 선언 변수는 다른 타입의 변수임을 주의하자.

구조체멤버

한구조체 내에서 멤버 이름은 유일해야 하지만, 서로 다른 구조체에서는 같은 멤버 이름을 사용할 수 있다.

```
struct fruit{char *name;
             intcalories;};

struct vegetable {char *name;
                  intcalories;};
struct fruit       a;
struct vegetable    b;
```

멤버접근연산자 '.'

멤버접근연산자 '->' **(예제 필수 확인.)**

-포인터를 통한 구조체 멤버 접근 연산자
- 포인터형변수 -> 멤버
- (*포인터형변수).멤버

멤버 접근 연산자 -> 예제

```
struct complex {
   double    re;          /* real part */
   double    im;          /* imag part */
};

typedef   struct complex   complex;

void add(complex *a, complex *b, complex *c) {
   a -> re = b -> re + c -> re;
   a -> im = b -> im + c -> im;
}
```

멤버 접근 연산자

```
Declarations and assignments
struct student   tmp, *p = &tmp;
tmp.grade = 'A';
tmp.last_name = "Casanova";
tmp.student_id = 910017

Expression              Equivalent expression       Conceptual Value
tmp.grade               p -> grade                  A
tmp.last_name           p -> last_name              Casanova
(*p).student_id         p -> student_id             910017
*p -> last_name + 1     (*(p -> last_name)) + 1     D
*(p -> last_name + 2)   (p -> last_name)[2]         s
```

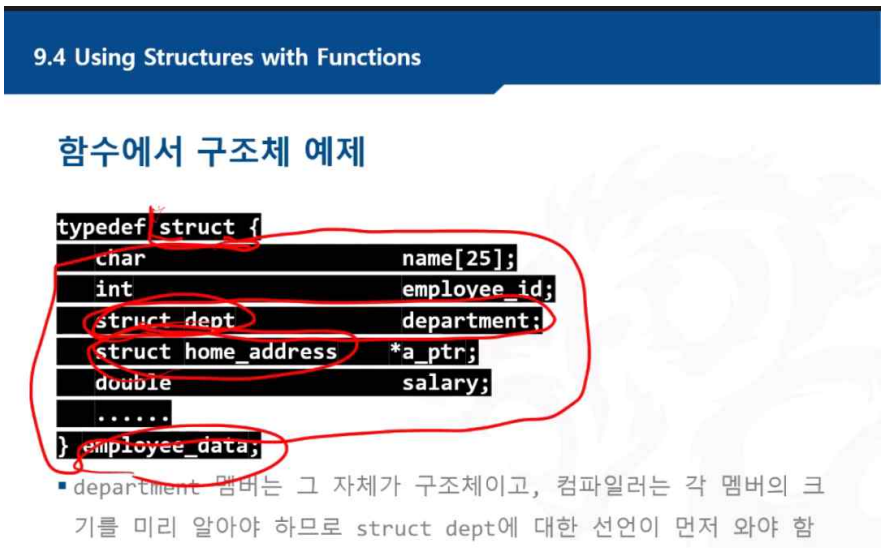
9.3 Operator Precedence and Associativity: A Final Look

Operators	Associativity
() [] . -> ++ (후위) -- (후위)	L→R
++ (전위) -- (전위) ! ~ sizeof (형) + (단항) - (단항) & (주소) * (역참조)	R→L
* / %	L→R
+ -	L→R
<< >>	L→R
< <= > >=	L→R
== !=	L→R
&	L→R
^	L→R
|	L→R
&&	L→R
||	L→R
?:	R→L
= += -= *= /= %= <<= >>= &= ^= |=	R→L
, (콤마 연산자)	L→R

이 페이지 무조건 중요!!!!!!!!!!!!!!!!!!!!!!!!

함수에서 구조체 (구조체는 데이터이므로)

함수에서는 call by value를 활용하는 데 value의 하나로 struct를 사용할 수 있다. 구조체는 함수 파라미터로 전달 될 수 있고, 함수로부터 리턴 가능함.



구조체멤버 그 자체가 구조체인데, 컴파일러는 각 멤버의 크기를 미리 알아야 하므로 struct dept에 대한 선언이 먼저 와야한다.

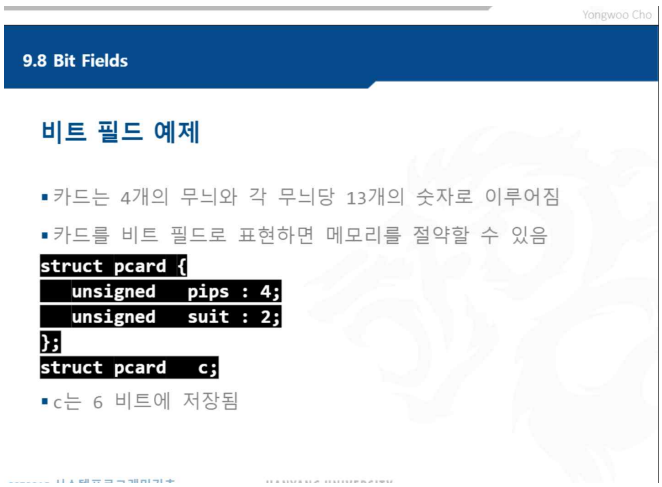
구조체의 초기화

공용체(구조체와 유사한 형태)

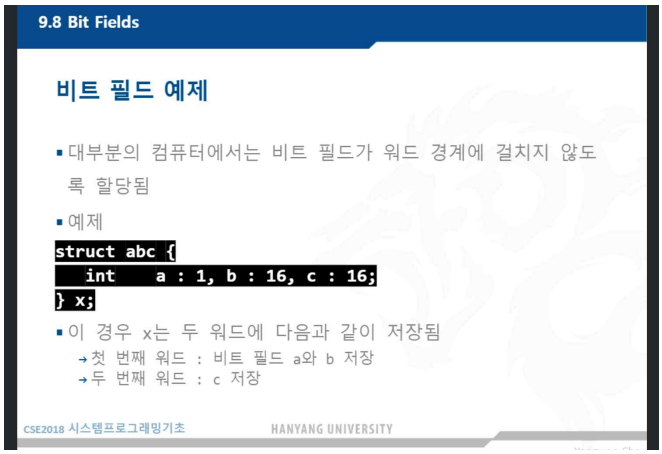
문법형태는 구조체와 비슷하지만, 각 멤버들은 같은 기억장소를 공유함. **공간은 하나지만 , 다양한 형태로(다형성)** 사용하고싶을 때 사용.

공용체형 변수는 멤버변수타입들 중 최대 사이즈로 메모리를 할당해서 사용

비트필드(01:24:00)
구조체나 공용체에서 int형이나 unsigned 형의멤버에 비트수(폭)를 지정 하는 것



▶ 대부분의 컴퓨터에서는 비트 필드가 워드 경계에 걸치지 않도록 할당된다.
▶ 워드단위가 32일 때 (아래)



데이터와 이에 대한 함수를 묶어서 하나의 덩어리로 보관하면? 이를 ADT라고 한다.

스택연산
push : 새로운 데이터 삽입
pop : 데이터 꺼내기
top : 현재 제일 위의 인덱스
empty: 비었는지
full : full인지?
reset : 리셋함

~01:43:00

++i, I++차이는 ?