**13. 구조체**

**★자료형의 분류**

1) 기초 자료형: int, char, double, void……

2) 파생 자료형: 배열, 포인터, 구조체, 공용체

3) 사용자정의 자료형: typedef, enum

**구조체 :** 여러 개의 자료형을 하나의 새로운 자료형으로 만들어 버림

**구조체 정의 :** **struct 구조체 이름(태그)** **{**

int number;

char name[10];

double grade;

**}; //마지막에 ; 필수!!**

- 구조체 정의 시, 태그를 설정 안 하는 경우가 있다. 이 경우 구조체를 정의할 때 변수를 선언하는데 그 이후 다시는 이 구조체를 사용해 변수를 선언할 필요가 없다는 뜻이다.

**Ex)** struct 구조체 이름(태그) {

int number;

char name[10];

double grade;

}**s1; //마지막에 변수 선언;**

★ **정의와 선언을 구분할 줄 알아야 한다.**

정의는 어떤 ‘틀’을 만드는 것이고, 선언은 그 틀을 통해 실체 ‘객체’ 등을 찍어내는 것을 말한다. Ex) int i; 에서 int는 자료형인데 C언어 개발자가 프로그램 어딘가에 정수형 자료형 int에 대해서 ‘정의’ 해 놓았을 것이고, 우리는 그 정의(틀)를 이용해서 i라는 실제 사용할 변수를 ‘선언’ 하는 것이다.

**구조체 선언: struct student s1 = {24, “Kim”, 4.3};**

(s1이 변수 이름)(동시에 초기화)

* 변수 선언 시, 메모리 영역이 할당
* **구조체 변수의 멤버에 접근(참조): s1.name**

**구조체를 멤버로 가지는 구조체**

struct date { //구조체 정의1

int year;

int month;

int day;

};

struct student { //구조체 정의2

int number;

char name[10];

**struct date dob;** //구조체1에 대한 변수 선언 //구조체 안에 구조체 포함

double grade;

};

struct student s1; //구조체 변수 선언

* 멤버 접근(참조): **s1.dob.year**

**구조체 변수의 대입 / 비교**

1. **구조체 변수끼리 대입 가능(한쪽 변수의 구조체 멤버들과 all 똑같아짐)**
2. **변수끼리 비교는 불가능, 반드시 . 연산자를 사용하여 멤버끼리 비교!!**

Ex) struct point{

int x;

int y;

};

struct point p1 = {10,20};

struct point p2 = {30,40};

**p1=p2; //가능**

**if(p1.x == p2.x) //이렇게 비교**

**구조체 배열:** 구조체를 여러 개 모은 것

struct student{

int num;

char name[10];

double grade;

};

**struct student list[3] ={ //구조체 배열 선언 및 초기화**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**{1, “Kim”, 4.3}**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**{2, “Lee”, 3.9}**

**{3, “Min”, 2.8}**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**};**

**구조체와 포인터**

1. **구조를 가리키는 포인터**

struct point p1; //구조체 변수 p1선언

**struct point \*s;** //구조체 포인터 s선언

**s = &p1;** //변수 p1의 주소값을 s에 저장

**★ -> 연산자 : 포인터로 구조체 멤버를 참조할 때**

**s -> name = (\*s).name = p1.name** (이 세 개가 모두 같은 뜻)

1. **포인터를 멤버로 갖는 구조체**

struct date{

int month;

int day;

int year;

};

struct student{

int number;

char name[10];

**struct date \*dob;**

//student 구조체 안에 있는 포인터 dob는 date구조체 변수의 주솟값을 가리킴

};

int main(void){

struct date d ~~~~ ; //date 구조체 변수 d선언

struct student s~~~; //student 구조체 변수 s선언

**s.dob = &d;**

return 0;

}

**구조체와 함수**

1) 구조체의 **함수의 인수로** 전달 가능 (값에 의한 전달)

* 구조체의 **복사본**이 전달(원본 변경 x) / 구조체의 크기 클수록 시간, 메모리 소모 크다

2) 구조체의 **포인터를 함수의 인수**로 전달 가능 (참조에 의한 전달)

* 원본의 **주소가 전달** 되므로 원본 변경 가능 / 시간, 메모리 절약

1. 구조체를 반환하는 경우에는 **복사본을 반환**한다.

* Return문은 하나만 반환 할 수 있기 때문에 여러 개의 데이터를 구조체로 묶어서 반환하면 편리하다.

struct student create( ){

**struct student s; // 함수 호출이 끝나면 사라진다.**

……..

**return s;** } //구조체 s의 복사본 반환

int main(void){

**a = create();** //

return 0; }

**공용체 :** 같은 메모리 영역을 여러 개의 변수가 공유 (메모리 영역 절감 / 계산의 편리성(IP주소))

**★ union** ex {

char c;

int I;

};

* **가장 큰 멤버의 크기만큼 메모리** 생성하고 다른 변수들이 이걸 공유

**열거형** : 변수가 **가질 수 있는 값들을 미리 열거**해놓은 자료형

**★ enum days**(이름,태그) **{SUN, MON, TUE, WED, THU, FRI, SAT};**

**enum days today; //변수 선언**

**today = SUN;**

* **열거형을 사용하지 않는다면, int today; 를 선언하고,**

**today = 0; //일요일 today = 1; //월요일**

**이렇게 의미없는 숫자에 의미를 부여해 사용하므로 가독성↓오류↑이다.**

* 따로 인덱스를 지정하지 않으면 sun =0 부터 시작한다.

Sun = 6 등으로 설정하면 그 다음부터는 1부터 시작한다.

Ex) enum days{SUN = 7, MON, TUE, WED, THU, FRI, SAT}; //mon은 0

enum days {SUN, MON, TUE, WED, THU, FRI, SAT};

char \*days\_name[ ] = {

“Sunday”, “Monday”, “tuesday”, “Wednesday”, “Thursday”, “friday”, “Saturday”};

Int main(void){

enum days d; //열거형 변수 선언

d = WED;

printf(days\_name[d]); //d = WED이므로 d는 3의 값을 갖는다.



**typedef:** 새로운 데이터 타입(자료형)을 정의하는 키워드

* **typedef** **unsigned char**(기존의 자료형) **BYTE**(새로운 자료형);

(unsigned char가 쓰기 귀찮으니깐 BYTE로 쓰는 법만 바뀐 거지 의미는 같다.)

* 이식성↑, 코드를 컴퓨터 하드웨어에 독립적으로 만들 수 있다.
* 문서화의 역할(타입을 내가 정하므로 주석 붙이는 것처럼 활용 가능)
* #define의 이용: #define BYTE unsigned char

1. 구조체도 typedef를 사용해 새로운 타입으로 정의할수 있다.

Ex 1) typedef struct point **POINT**;

Ex 2) struct point{

…….

}**POINT;** //구조체 정의와 typedef동시에(이렇게 한번에 정의할 땐 태그를 생략 가능)