구조체

2020년 12월 21일 월요일 오후 2:19

구조체(structure)란?

- 프로그래밍 입문에서 두번이나 봤음
  i. 클래스를 배우러 가는 과정에서 여러 가지 형의 데이터만 모아 놓은 것
  ii. 그리고 클래스를 배운 뒤에 참조형이 아닌 값형인 커스텀 데이터형
- C에서의 구조체는 첫 번째 개념과 같음
  - 데이터의 집합임 멤버 함수는 없음
  - 여러 자료형을 가진 변수들을 하나의 패키지로 만들어 놓은 것
- 구조체는 참조형 아니면 값형?
  - C에서는 모든 자료형이 값형이라 했음
  - 하지만 주소를 전달하면 참조형처럼 쓸 수 있음

구조체의 필요성

1. 사람은 세상을 바라볼 때 물체 단위로 봄
   - "지금 몇 시죠?"라고 물으면 사람들은
     "2시 39분이요"라고 말함
     - 이때 2시와 39분을 따로 떼어서 생각하는 사람은 없음
   - 예를 들어 마우스를 볼 때도 왼쪽 버튼/오른쪽 버튼을 같이 생각
2. 구조체를 사용하면 실수도 막을 수 있다
   - 같은 형의 데이터 여러 개를 매개변수로 받을 때 순서가 바뀌면 컴파일러가 실수를 찾을 방법이 없다

```
float calculate_BMI(float height, float weight);
```

```
calculate_BMI(75.2f, 180.3f); /* 키와 몸무게가 잘못 들어감 -_-; */
```

원래 모습 잘못 들어가버린 정보

   - 묵시적으로 변환 가능한 자료형이 여러 개일 때도 마찬가지

```
int save_data(unsigned char level, float money, const char* name)
```

```
unsigned char lvl = 10;
unsigned int exp = 12355;
float money = 256.10;
const char* name = "Griff Bright";

int result = save_data(lvl, exp, name);
```

     - int형 변수 exp가 float인 money 매개변수자리에 들어갔다
       int에서 float으로는 문제 없이 묵시적 변환이 일어나서 컴파일이 된다
   - 매개변수의 목록이 더 길어지면 더욱더 실수하기 쉬움

```
int save_data(unsigned char level, unsigned int exp, float money,
   unsigned char str, unsigned char intell, unsigned short death_count, ...);
```

```
save_data(level, money, exp, intel, str, death, assist, ...);
```

- 이제는 어디에 실수했는지 발견하기도 힘들다

- 중간에 누가 매개변수를 바꿔도 문제

```
void update_KDA(int kill, int death, int assist);
```
← 여기만 바꾸고

```
int kill = 40;
int assist = 60;
int death = 1;
update_KDA(kill, assist, death);
```
← 여기는 안 바꿈 -_-;

  - 함수의 매개변수 순서를 바꾸고 그 함수를 사용한 곳이 500군데가 있다고 하면 499군데를 성공적으로 바꾼다 해도 1군데를 바꾸지않으면 문제가 생긴다.

- 실수를 막으려고 또 다른 실수를 초래하는 예

```
void set_year(unsigned int year);
void set_month(unsigned int month);
void set_day(unsigned int day);
int is_monday();
```

```
set_year(2043);
set_month(10);
set_day(1);

if (is_monday()) {
    /* 컴퓨터 포매애애애앳~ */
}
```

이상

```
set_year(2043);
set_month(10);

if (is_monday()) {
    /* 컴퓨터 포매애애애앳~ */
}
```

현실

  - 매개 변수 순서가 헷갈리니까 매개변수별로 함수를 만들어버리면 실수가 줄까?
    - 이 경우 오히려 함수가 많아져 함수를 하나 까먹을 수도 있다

- 위와같은 실수들을 줄이려면
  - **원자성을 보장하는 연산**(atomic operation)을 사용하는게 좋다
    - 여러 단계 함수를 거쳐서 결과를 내는 것보다 하나의 함수로 결과를 내어 실수를 줄이는 것
    - 실수 없이 한 방에 구하려면 구조체를 사용해야 함

구조체의 선언

```
struct date {
    int year;
    int month;
    int day;
};
```

- 끝에 세미콜론 잊지 말 것
- date란 구조체(새로운 형)를 만든다
- 그 안에 들어가 있는 데이터는 총 세 개
  - year
  - month
  - day

구조체 변수 선언 및 사용하기

- date란 구조체를 date라는 변수명으로 선언

```
×
```

- 함수의 매개변수로도 사용 가능
- 인자로 전달하려면 이렇게

```
×
```

- 그 안에 있는 멤버 변수에 접근하려면 . 연산자를 사용

        - 당연히 지역 변수 선언시 0으로 초기화 안됨
            - C의 다른 자료형과 마찬가지
            - 스택 위치에 있던 값을 그냥 가져다 씀
    - 좀더 간결하게 구조체 선언 및 변수 선언을 할 수 없을까?
        - typedef를 사용하면 됨

typedef

- 우리가 사용중인 size_t가 보통 이렇게 되어 있다

    - 이미 있는 자료형인 unsigned int에 새로운 별명을 지어줌
    - 그 새로운 이름은 size_t
        - 근데 그냥 unsigned int를 써도 전혀 상관 없음
        - 서로 바꿔가며 써도 됨
    - 물론 size_t의 경우는 각 구현마다 자료형이 달라질 수 있어서 typedef를 해놓은 것
    - 구조체에 typedef를 쓰면 다른 자료형처럼 간결하게 변수 선언이 가능
- typedef 사용법 1

- typedef 사용법 2

    - 어느 방법을 써도 크게 상관없음
        - "typedef A 이름_t" 형식만 맞추면 된다
    - C에서 _t로 끝나는 자료형은 보통 이렇게 typedef 한 것
- enum에 typedef 사용하기

    - enum도 "typedef A 이름_t" 형식만 맞추면 된다
- 코딩 표준: 커스텀 자료형에 typedef를 쓰자
    - 가능하면 구조체, 열거형, 공용체에 typedef를 써서 정말 제대로된 자료형처럼 보이게 하자

구조체 변수 초기화 하기

- 구조체 선언과 동시에 초기화가 안 된다
    - 스택에 남아있는 데이터를 그대로 사용

- 그냥 구조체는 지역 변수를 여러 개를 따로따로 사용하는 것과 마찬가지라고 생각하자
    - 사실 기계는 구조체라는 개념을 모름
    - 구조체라는 개념은 프로그래밍 언어가 프로그래머가 편하고 실수하지 않도록 제공해준 개념임
    - 어셈블리어를 보면 알 수 있다

        - 완전히 똑같다.
            - 똑같이 3개의 변수를 스택 메모리에 순차적으로 쌓음
- 다른 구조체 초기화 방법

    - 이렇게 구조체를 초기화 할 수도있다
        - 멤버 변수 하나하나 0으로 초기화 됨
    - 배열 초기화 때 썼던 방법이다
        - 첫번째 원소 이외에 나머지 원소가 있다는 걸 나타내기 위해 { 0, } 이렇게 나타냄
    - 컴파일러에 따라 그냥 date가 차지하는 모든 메모리 공간을 0으로 채워주는 명령어로 바꿔줄 수도 있음
        - 그것은 바로 memset( )
        - 배열도 이런식으로 memset( )을 사용해서 0으로 초기화 되는 경우가 있다

② • 요소 나열법으로 구조체 초기화 하기

- 멤버 변수에 순서대로 차례대로 대입

    - 근데 되도록 이 방법은 쓰지 말자
    - 실수할 여지가 있음
        - 다른 프로그래머가 구조체의 멤버 변수 순서를 바꾸고
          변수 초기화 목록을 안 고치면 문제가 된다

- 요소 나열법이 유용한 경우가 딱 하나 있다
    - 멤버 변수가 const일 때, 선언 시 초기화 안하면 다시는 못하니까

        - 근데 const 멤버 변수는 잘 쓰지도 않고 쓰지 말라는 게 업계 표준

■ 구조체 매개변수

- 인자 전달도 기본 자료형과 똑같이 작동

- 완전히 똑같다
- int형 변수, 구조체 변수 모두 다 값형이므로 스택프레임에 복사되어 매개변수로 전달 된다
  - 복사본을 전달하기 때문에 원본 구조체 멤버에 접근할 수 없다
  - 원본을 바꾸고 싶다면 구조체 포인터를 사용하면 됨

구조체 포인터에서 멤버의 값에 접근하기

- ( ) 필요한 이유?
  - 연산자 우선순위 때문
  - . 연산자의 우선순위는 1, * 연산자의 우선순위는 2
- 고로 괄호가 없다면
  - *date.year은 *(date.year)과 같게 된다
    - 주소(date)에서 year멤버에 접근하게됨 -> 좀 말이 안 됨

    - 이걸 합친 연산자가 있음
        - 바로 -> 연산자
            - 우선순위는 1순위이다

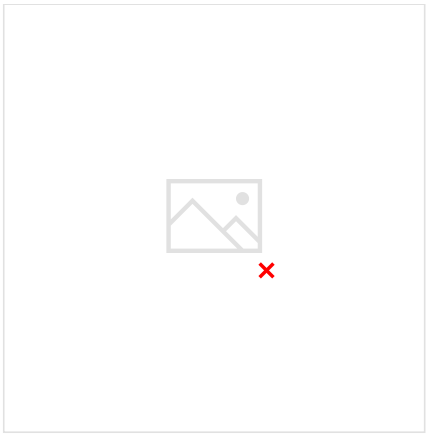
            - 되도록이면 ->연산자를 쓰는게 깔끔하다

- 구조체 매개변수 베스트 프랙티스
    1. 값으로 전달 vs 주소로 전달
        - 기본 자료형 전달할 때는 간단했음
            - 기본 데이터 크기가 작으니 원본 바꿀 때만 주소로 전달하면 되었음
        - 구조체의 경우 데이터 크기가 클 수도 있음
            - 구조체 안에 int형 멤버 변수가 5만개면 50,000 X 4 ~= 200KB
            - 이럴 때는 다 복사하는게 성능이 느릴 수도 있음 -> 이럴 때 주소로 전달
                - 구조체를 함수에 값으로 전달하면 스택 메모리가 부족할 수 있다
            - 포인터에 const 포인터를 붙이면 원본도 못 바꾸니 안전

                - 이렇게 주소만 전달하면 4바이트만 복사가 됨
    2. 구조체 매개변수 vs 여러 개의 개별 변수?
        - 딱히 정확한 규칙이 있지 않지만 보통 변수 많이 전달하는 대신 구조체 하나 전달하라고 함
        - 한 4개까지는 낱개 변수로 그 이후에는 구조체로 넘기라는 규칙을 쓰는 회사도 있음
        - 이유는 다양
            - 실수를 줄이기 위해서도 있고
            - 성능을 빠르게 하기 때문도 있고(특히, 주소로 전달할 경우)

■ 함수 반환값으로서의 구조체, 구조체 배열

- 함수 반환값으로서의 구조체



- int 형 하나 반환하는 것과 마찬가지로 복사에 의해 반환
- C 언어의 함수는 반환값이 하나라고 했음
  - C언어에서 두 개 이상의 값을 반환하기 위해서는 out 매개변수를 사용
- 이렇게 구조체를 반환하면 실질적으로 여러 개의 값을 반환하는 격

- 구조체를 대입 할 수도 있음
  - date = get_dday( ); 가 된다는 건 대입도 된다는 이야기
  - 개념상 그냥 각 멤버 변수를 돌아가며 하나씩 대입한다고 봐도 됨

  - 어떤 컴파일러들은 메모리를 통째로 그냥 복사해 줌
    - memcpy( )란 함수
      - 동적 메모리 배울 때 봄

- 구조체로 배열도 만들 수 있음
  - 정말 기본 자료형이랑 다르지 않다고 생각하는게 편함
  - 구조체마다 자료 크기가 딱 정해져 있으니 컴파일러가 다른 변수와 똑같이 처리해줄 수 있음
    - date_t의 크기
      - sizeof(int) + sizeof(int) + sizeof(int) = 12바이트
      - 12바이트 크기의 구조체 요소를 가진 배열을 만들 수 있다
  - 가족의 생일을 저장하는 구조체 배열 예

- 배열의 각 요소가 몇 바이트씩 떨어져 있는지 구하기
  - 요소 하나의 크기는 12바이트라고 했음
  - 실제로 그런지 확인하면

    - sizeof()에 구조체 배열의 이름이 아니라 구조체 변수 타입(date_t)을 넣어도 된다
  - 요렇게 해도 같은 결과

  - 실제 메모리

- 구조체에 문자열 포인터(pointer to a string)를 사용하면 문제점

    - 결과

    - name과 clone의 멤버들이 각각 같은 문자열의 주소를 가리키고 있어서 생기는 문제

- 얕은 복사
    - 이렇게 실제 데이터가 아니라 주소를 복사하는 걸 얕은 복사라고 함
- 깊은 복사
    - 이럴 땐 깊은 복사를 하는 게 맞는데 이건 대입만으로는 안 됨
    - 구조체 변수마다 독자적인 메모리 공간을 만들어주고 거기에 문자열을 복사해야 함
        - 동적 메모리 배우면 알 것임

- 파일을 읽고 쓸 때도 비슷한 문제가 발생

- 이 주소들을 기억

- 실행을 하면 이렇게 오류가 나거나 운좋게 실행이 될 수도 있다

- 디버깅해보니 구조체에 앞에서의 문자열들의 주소가 적혀 있음

- 파일을 열어서 보니 문자열이 있는게 아니라 앞에서 본 문자열의 주소가 적혀 있음
  - 0x009C30A, 0x009C300F, 0x009C3006 이렇게

- 문제는 포인터
  - 포인터는 주소를 저장함 그래서 파일에 주소가 적혀 있음

    - name_t의 크기를 보니 역시 8 바이트
    - 즉, 주소 2개만 들어있음!
  - 주소를 파일에 저장해 봐야 의미가 없다.
    - 어떤 주소라는 건 프로그램이 실행될 때만 의미가 있는 것
    - 프로그램이 끝나고 나면 그 주소는 의미가 없거나,
      접근하면 안되는 주소일 수도 있다
- 포인터 변수 없는 name_t

  - enum을 사용해서 상수를 선언
    - enum 타입은 따로 주지 않음
    - 다른 문제를 해결할 수 있어서 이렇게 한다고 한다

  - size가 8바이트가 아닌 64바이트
    - 함수의 인자로 전달해도 여전히 64비트
  - 구조체 배열을 매개변수로 쓸 수 있다는 건 구조체 배열로 대입도 할 수 있다는건가?
    - 맞다. 이걸로 배열을 복사도 할 수 있음
  - 이렇게 다시 파일에 저장해보자

        - 포인터가 아니므로 문자열의 주소만 대입하는식으로 안 된다
            - strncpy( )를 사용하여 실제 char배열에 문자열을 집어 넣어야 한다
                - strncpy( )사용할 때 팁
                    - 배열의 길이만큼 넣어도 마지막에 널 문자가 들어가지 않을 수 있으니 배열의 마지막에 직접 널 문자를 넣어준다
        - firstname1과 names[0].firstname의 주소가 다르다
            - 이제 각자만의 주소가 있는거임
        - 파일을 열어서 보면 이제 주소가 아닌 이름들이 잘 저장되어 있다

            - 32비트 안에 한 문자열이 저장되어 있어 중간에 의미없는 문자들이 있다
                - 2줄에 이름 하나씩 적혀 있는 모습을 볼 수 있음
    - 다시 파일을 읽어보자

        - sizeof(names[0])은 64바이트가 되고 NUM_NAMES는 2개이다. 총 128바이트
        - 읽어와서 저장하면 구조체의 주소가 바뀐 모습을 볼 수 있다

            - 구조체를 파일에 저장할 때 주소는 0x00f5f758
- 구조체 안에 있는 데이터는 포인터가 없는게 좋다
    - 가능한 이렇게 한 덩어리 메모리에 모든 데이터가 대입 가능한 구조체를 만드는 게 좋음
        - 즉, 포인터만 없으면 됨
        - 구조체에서 포인터만 사용 안 하면 대입과 파일 저장을 자유롭게 할 수 있따
        - 포인터를 사용 안 하면 구조체에 대입(=)하는 것만으로 복사를 쉽게 할 수 있다
        - 이렇게 구조체를 통째로 파일에 저장하고 읽어 올 수도 있고
        - 값을 변경해도 다른 것에 영향 받지 않고
            - 포인터 없이 독자적인 메모리를 사용하고 있기 때문

■ 구조체를 다른 구조체의 멤버로 사용하기

- user.name.firstname[NAME_LEN - 1]
  - 구조체 안에 구조체에 값을 대입하기 위해서 . 연산자를 두번 씀

- 파일 안에 몇 바이트가 쓰였을까?

  - 구조체 크기가 76바이트니까 파일에도 76바이트가 적혀있을까?

  - 파일에는 80바이트가 쓰였다...
  - 실제 구조체 크기를 봐도 80바이트

- user_info가 메모리에 어떻게 들어가 있는지 확인하는 코드

  - 0: id의 시작 위치
  - 4: name의 시작 위치
  - 68: height의 시작 위치
  - height는 short로 크기가 2바이트임
    그래서 weight의 시작 위치가 70이 되야 하는데 72이다.
    - 무슨일이 일어난 걸까?

- 키랑 나이를 저장하는 저장 공간이 2바이트가 아니라 4바이트로 되어 있다

- 바이트 정렬 요구사항 때문에 구멍이 생김
  - 살펴보니 4바이트를 꽉 안 채운 애들이 쓸데없는 공간을 먹음
  - 이건 각 시스템마다 메모리에 접근할 때 사용하는 주소에 대한 요구사항이 다르게 때문
    - 시스템 상의 제약이 있거나
    - 효율성 때문
  - 어떤 시스템은 n바이트 배수인 시작주소에서만 메모리를 접근 가능
  - x86 시스템은 4바이트(워드 크기) 경계에서 읽어오는 게 효율적
    - 이걸 4바이트 경계에 정렬된다(aligned)고 말함
    - x86은 4바이트씩 데이터를 처리하는데 방금처럼 2바이트 short를
      메모리에 저장할 때는 2바이트 빈공간(구멍)을 추가하여 4바이트 공간에 short를 저장한다
  - 따라서 컴파일러가 알아서 각 멤버의 시작 위치를 그 경계에 맞춤
  - 그러기 위해 안 쓰는 바이트를 덧 붙임 (padding)
    - 4바이트가 되게 2바이트의 빈 구멍을 만들어주는데 이를 덧붙인다(padding) 표현한다
  - 32비트 clang 윈도우는 4바이트 정렬을 하려고 함
  - 따라서 어떤 아키텍처에서 저장한 파일을 다른 아키텍처에서 읽으면 잘못 읽힐 수도 있음
    - 어떤 기기에서 구조체를 컴파일해서 fwrite로 파일에 저장을 했음
      그때 구조체의 한 요 소는 40바이트였다고 하자
    - 다른 환경(OS, CPU)에서 컴파일을 했음 그리고 저장된 데이터를 fread로 읽어왔음
      그때 바이트수가 안 맞으면 엉뚱한 데이터가 읽힐 수 있다
    - 다른 환경에서는 이 구조체가 30바이트 였다면
    - 40바이트의 데이터를 30바이트씩 읽어오니까
      첫 번째 읽어올 때는 데이터의 부분밖에 못 읽고
      두 번째 읽어올 때는 그 다음밖에 못 읽기 때문에
      첫 번째 데이터와 두 번째 데이터가 섞여서 이상하게 읽어오게 된다.
- align하는 현실의 예

  - 그림처럼 줄을 세우는 방식이 다르면 데이터 공유가 어려워진다
- 패딩 줄이기

  - 데이터의 순서를 바꿔서 모든 데이터가 4바이트에 예쁘게 들어갈 수 있으면 컴파일러도 패딩을 안 넣고 딱 필요한 만큼만의 구조체 크기를 잡아줄 수도 있다
  - 2개의 short 형 변수가 4바이트로 합체 됐음
  - 크기가 80바이트에서 76바이트가 됨!
- #pragma pack을 쓰는 방법도 있다

  - #pragma pack(push, 1)
    - 패딩 넣지 말고 1바이트씩 align으로 packing 하라는 뜻
  - 크기도 우리가 원하는 76
  - 그러나 표준은 아니고 요즘 컴파일러들이 잘 지원해준다고 함
- 구조체 베스트 프랙틱스
  - 구조체를 파일이나 다른 데 저장해야 해서 바이트 크기가 정확히 맞아야 한다면
    - 컴파일하고 실행할때 문제가 생기는 걸 잡기 위해서 바이트 크기가 정확히 맞아야 함
    - 보통 assert( )를 사용해서 크기를 확인

    - 이렇게 assert하면 만약에 다른 플랫폼에서 구조체 크기가 다를 때 디버그 모드에서 잡힌다. 잡히면 문제를 해결하면 된다.
  - 어쩔 수 없이 패딩이 생길거라면 구조체에 패딩을적으로 넣기도 함

- 특히 데이터의 전체 크기가 4바이트로 안 나눠 떨어질 때
  - 위의 경우 1바이트 자료형이 2개가 있어 4로 나누어 떨어지지 않는다
    이때는 pragma를 쓰지 않는 한 그냥 2바이트를 추가해준다