함수 포인터

2020년 12월 30일 수요일 오후 12:31

함수 포인터, 함수를 변수에 저장할 수 있을까?

- 포인터가 뭐에요? 하면
  - '메모리 주소를 저장하는 변수' 라고 딱 나와야 하고 이렇게 이해하는게 가장 좋다
- switch문을 사용한 사칙연산 프로그램

```
double add(double, double);
double sub(double, double);
double mul(double, double);
double div(double, double);
```

calculator.h

```
double add(double x, double y)
{
    return x + y;
}

double sub(double x, double y)
{
    return x - y;
}
```

```
double mul(double x, double y)
{
    return x * y;
}

double div(double x, double y)
{
    return x / y;
}
```

calculator.c

```
double op1 = 10.234;
double op2 = 3.521;
double result;
char op = '+';

/* 입력받는 코드 생략 */
```

```
switch (op) {
case '+':
    result = add(op1, op2);
    break;
case '-':
    result = sub(op1, op2);
    break;
case '*':
    result = mul(op1, op2);
    break;
case '/':
    result = div(op1, op2);
    break;
default:
    fputs("unknown op!", stderr);
}
```

main.c

- Switch문 말고 다른 방법이 없을까?
  - If문은 당연히 아니고
  - 어차피 사칙연산은 모두 피연산자 2개를 필요로 하고
    그러다 보니 함수 시그니처도 모두 비슷하고
    유일하게 다른 점은 함수 이름 정도임

```
result = add(op1, op2);
result = sub(op1, op2);
result = mul(op1, op2);
result = div(op1, op2);
```

  - 언제나 변하는 피연산자를 매개변수로 전달하듯이
    함수도 어디다 저장해 둔 뒤 매개변수로 전달할 수 없을까?

```
result = calculate(op1, op2, operator); /* operator = add() */

result = calculate(op1, op2, operator); /* operator = sub() */

result = calculate(op1, op2, operator); /* operator = mul() */

result = calculate(op1, op2, operator); /* operator = div() */
```

    - 함수를 매개변수로 전달하면 하나의 함수로 4개의 연산을 할 수 있다
- 함수 호출 방법을 다시한번 떠올려보자
  - 어떤 함수를 호출할 때는 직접 함수명을 씀

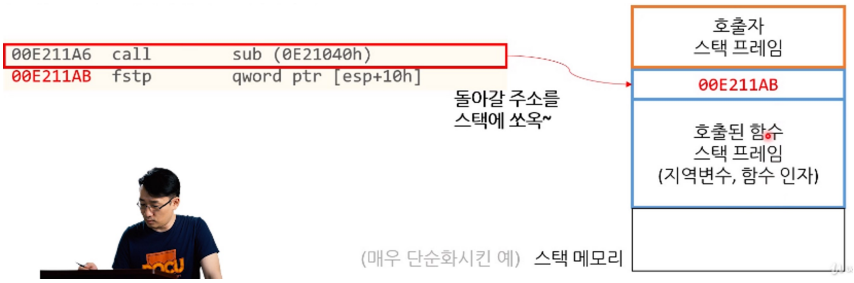
```
result = sub(op1, op2);
```

  - 그러나 어셈블리어로는 그 함수의 주소로 점프

```
00E211A6  call        sub (0E21040h)
```

```
     8: double sub(double x, double y)
     9: {
00E21040  push        ebp
00E21041  mov         ebp,esp
00E21043  and         esp,0FFFFFFF8h
00E21046  sub         esp,18h
00E21049  movsd       xmm0,mmword ptr [ebp+10h]
00E2104E  movsd       xmm1,mmword ptr [ebp+8]
00E21053  movsd       mmword ptr [esp+10h],xmm0
00E21059  movsd       mmword ptr [esp+8],xmm1
```

        - Call sub (0E21040h)
            - sub함수를 call 함
            - sub옆에는 그 함수가 시작하는 주소가 괄호 안에 있다
            - 00E21040주소로 점프를 하는 것임
    - 0E21040h는 컴파일 할 때 정해지는 주소임
      실행 중에 바뀌는 주소가 아님
        - 함수를 호출할 때 함수의 주소로 점프하는 걸 이용할 수 있겠지만
          이 주소는 컴파일 할 때 정해므로 실행 중에 원하는 함수가 있는 주소로 변경할 수 없음
- 실행 중에 다른 코드로 이동하는 경우는 뭐가 있었을 까?
    - 실행 중에 다른 코드로 이동한다는 것은
        - 이동할려는 코드의 주소가 어셈블리어 코드에 있는게 아닌 (컴파일 할 때 정해지는 주소가 아닌)
          스택 프레임에 주소가 들어가 있어야 한다 (실행 중에 주소가 스택 프레임에 들어갔다 나왔다 정해지는 거임)
    - **함수에서 반환할 때 돌아가야 하는 호출자 코드의 주소**가 이런 경우였다.
        - 돌아갈 주소는 스택 메모리에 저장되어 있었음
            - 돌아갈 주소가 스택에 들어있단 의미는?
              실행 도중에 결정된다는 것



            - 함수를 호출할 때 돌아갈 주소인 다음 주소 00E211AB를 스택 프레임에 푸쉬하고 함수의 주소로 이동.
              00E211AB 이 주소는 호출자에 따라 실행 도중에 변할 수 있는 것임.
              (그러니까 어셈블리어 코드에 직접 들어가 있는 주소가 아니라 실행할 때마다 바뀌는 주소)
    - 즉, 모든 것이 다 메모리 주소
        - 그럼 실행 도중 조건에 따라 어떤 함수를 실행해주려면 무슨 주소를 변수에 기억하고 있어야 하는 거지?
            - 그 함수의 시작 주소
        - 실제 이렇게 어셈블리 보면 모든 코드 앞에 메모리 주소가 있고 함수의 시작코드도 마찬가지

```
     5: int main(void)
     6: {
00E21100  push        ebp
00E21101  mov         ebp,esp
00E21103  and         esp,0FFFFFFF8h
00E21106  sub         esp,68h
00E21109  mov         dword ptr [esp+64h],0
     7: double op1 = 10.234;
00E21111  mov         dword ptr [esp+5Ch],402477CEh
00E21119  mov         dword ptr [esp+58h],0D916872Bh
```

■ 함수를 매개변수로 전달할 때 필요한 것들, 함수 포인터 선언

- 이제 실마리는 다 잡았음
    - 그 함수 코드의 시작 메모리 주소를 넣고 그것을 실행해달라고 하면 되겠다!
    - 그래서 그것을 함수 포인터라 하는 것이고
- 함수 포인터를 매개변수로 넣기

```
result = calculate(op1, op2, operator); /* operator = add() */
result = calculate(op1, op2, &operator); /* operator = add() */
```
나의 꿈

- 이런식으로 하면 될 듯?
- 둘 다 가능
- 그러나 보통 위의 방법을 더 많이 씀

- 문제는 선언

```
double calculate(double, double, function*);
```
시도

- 함수의 주소를 저장할 테니 포인터니까 이렇게 하면 될 까?
  - 안 된다

```
In file included from calculator.c:1:
./calculator.h:12:34: error: unknown type name 'function'
double calculate(double, double, function*);
                                 ^
1 error generated.
In file included from main.c:3:
./calculator.h:12:34: error: unknown type name 'function'
double calculate(double, double, function*);
                                 ^
1 error generated.
```

    - function 이란 타입 이름은 없음..
- 생각해보니 이렇게 하는 건 말이 안 됨
  - 어떠한 함수도 담을 수 있는 자료형은 말이 안 됨
  - 함수는 다양한 매개변수 목록과 반환형을 가지는데 이를 하나로 표현 불가

```
void do_something(const char*);
int do_useless(void);
float do_yourcode(int, int);
char* do_anything(int, float, short, double, ...);
void do_nothing(void);
```
이걸 어떻게
'하나'의 통일된 자료형으로 보여주지?

    - Int, double, float을 구별하듯이 함수들도 매개변수와 반환형에따라 구별을 해야 함
- 함수가 어떻게 실행되는지 떠올려보자
  - 매개변수로 전달된 함수가 무사히 실행되려면
  - 그 함수에서 쓸 매개변수가 스택에 들어가 있어야 함

```
double calculate(double, double, 매개변수로_전달된_함수*);
```
  - 여기서 앞의 두 double은 calculate( )의 매개변수지 매개변수로_전달된_함수의 매개변수가 아님
  - 매개변수로_전달된_함수의 매개변수도 스택 프레임에 복사해야 하므로 알아야 함
    - 지금 모양에서는 컴파일러가 이 함수에 대해 아는 유일한 정보는 함수 포인터이며 그 크기는 4바이트
    - 이 함수를 호출하려면 어떤 형의 매개변수를 몇 개나 스택 메모리에 넣어줘야 하는지 모름
    - 반환값이 있는지, 있다면 어떻게 가져다 써야 하는지도 모름
  - 이것이 설사 사용이 된다고 치면

```
double calculate(double op1, double op2, 매개변수로_전달되는_함수* func)
{
    ...
    double r = func(op1, op2);
}
```
    - 함수 포인터를 이용해서 이 함수를 호출하면 컴파일러 입장에서는 굉장히 생뚱맞은 상황이 되어버림
      - 왠진 모르지만 갑자기 func가 double 형 두 개를 매개변수로 받고
      - 왠진 모르지만 갑자기 func가 반환하는 값을 double 형에 대입 가능
      - 그리고 갑자기 밑에서 double 매개변수를 3개를 받고 int형을 반환해버릴 수도 있다
  - 즉, 매개변수로 전달되는 함수는 다음과 같은 내용도 있어야 함
    - 자기 자신이 받아야 하는 매개변수 목록
    - 자기 자신이 반환하는 자료형

올바른 함수 포인터

```
double add(double x, double y)
{
    return x + y;
}
```

함수 포인터 변수의 선언과 사용

```
double (*func)(double, double) = add;
result = func(op1, op2);
```

- Func는 함수의 시작주소를 저장하는 포인터인데
  매개변수가 double형 두 개이고, 반환값이 double임
- Func에 함수를 대입하고 나면(소괄호는 사용 안해도 됨)
  대입한 함수같이 사용할 수 있다

함수 포인터 매개변수의 선언과 사용

```
double calculate(double, double, double (*)(double, double));
```

```
double calculate(double x, double y, double (*func)(double, double))
{
    return func(x, y);
}
```

```
result = calculate(op1, op2, add);
```

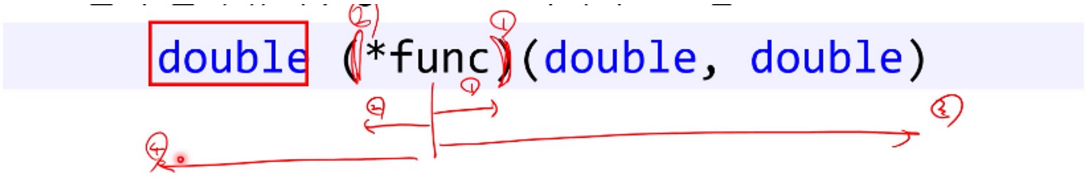
- 함수 포인터 매개변수를 선언할 때는 변수 이름을 빼줘도 됨

- 함수 포인터 선언

```
<반환형> (*<변수명>)(<매개변수 목록>);
```

  - 함수의 시작 주소를 저장하는 변수
  - 함수의 매개변수 목록과 반환형을 반드시 표기해야 함

- 함수 포인터 읽는 방법: 오른쪽 - 왼쪽 규칙(Right-Left Rule)
  - 오른쪽-왼쪽 규칙(Right-Left Rule)



    - 오른쪽으로 읽었다가 괄호에 의해 막히면 왼쪽으로 읽는 방법
      - 마치 도돌이표 같다
      - 한번 지난 괄호는 무시함여기에 수식을 입력하십시오.
    - 영어의 어순으로 읽자면
      - (1)변수 func는 (2)포인터이다. 그 포인터는 뭘 가리키냐 (3)두 개의 double 매개변수를 갖는 함수를 가리킴. (4)그 함수의 반환값은 double임.
    - 한글 어순으로 바꾸면
      - 변수 func는 두 개의 double형 매개변수를 받고 double 형을 반환하는 함수의 포인터이다.
    - 다른 예 읽어보기

×

      - 변수 func는 포인터이다. 그 포인터가 가리키는 것은 한 개의 int 매개변수를 갖는 함수임. 그 함수의 반환값은 void임.

- 함수 포인터를 함수의 매개변수로 쓰기

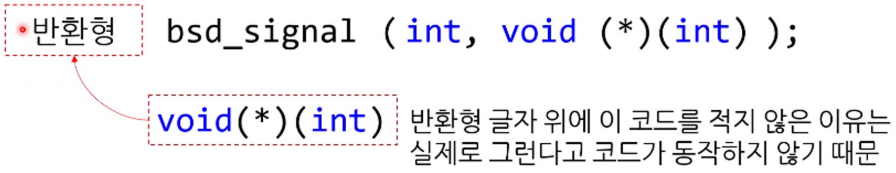
×

  - 우리가 원하던 것
  - 함수 calculate( )는 매개변수가 총 3개
    1. double형
    2. double형
    3. 함수 포인터
       - 두 개의 double 형 매개변수를 받고 double 형을 반환하는 함수를 가리킴

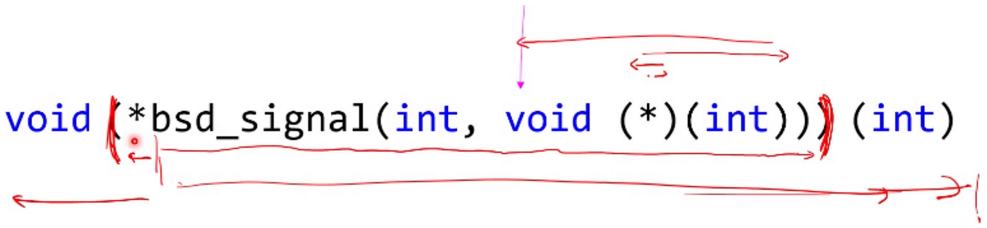
    - 반환형은 double 형
- 함수 포인터 배열도 만들 수 있음

    - 똑같이 오른쪽-왼쪽 법칙을 사용해서 읽을 수 있음
        - ops는 배열임. 그 배열은 포인터를 담음. 그 포인터는 int 변수 두개를 매개변수로 갖는 함수를 가리킴 그 함수는 반환값이 int 임.
- 좀 더 복잡한 함수 포인터 읽기

    - bsd_signal 함수의 선언
        - 매개변수
            - Int 형
            - 함수 포인터: void (*)(int)
        - 반환형
            - 함수 포인터: void (*)(int)
        - 읽기가 힘듦..
    - 코드를 분해해서 이해해보기
        - 원래 우리가 함수를 선언할 때 어떻게 했나?
            - 이런식으로 함
        - bsd_signal을 이처럼 표현해보면



        - 원래 코드로 돌려보면



            - 오른쪽 - 왼쪽 규칙을 사용해서 읽어보자
                1. bsd_signal은 매개변수로 int와 함수 포인터(void (*)(int))를 받는 함수이다.
                    - 이 함수 포인터는 내부적으로 뭐냐 포인터임. 이 포인터는 매개변수로 int를 받는 함수를 가리킴. 이 함수는 void를 반환함.
                2. 근데 bsd_signal은 포인터를 반환하는 함수임. 그 포인터는 int를 매개변수로 갖는 함수를 가리킴. 그 함수는 void를 반환함.
        - 함수 포인터 복습 퀴즈
            - `char (*foo)(void*)`
                - foo는 포인터임. 그 포인터는 void*를 매개변수로 갖는 함수를 가리킴. 그 함수는 반환값이 char임
            - int* (*foo)(int, void(*)(int))
                1. Foo는 포인터임. 그 포인터는 매개변수로 int와 함수 포인터를 갖는 함수를 가리킴.
                    - 함수포인터는 이름이 없음. 함수포인터는 Int를 매개변수로 하는 함수를 가리킴. 그 함수의 반환값은 void임.
                2. 그 함수는 int*를 반환함.
            - unsigned int* *(*foo)(char(*)(void*))

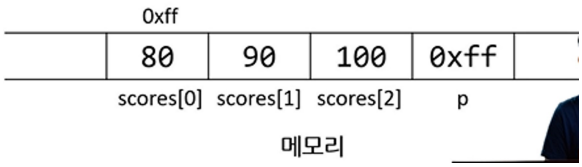
1. Foo는 포인터임. 그 포인터는 함수 포인터를 매개변수로 갖는 함수를 가리킴.
   ◇ 함수 포인터는 이름이 없음. 함수 포인터는 void*를 매개변수로 하는 함수를 가리킴 그 함수의 반환값은 char임
2. 그 함수의 반환값은 unsigned int*의 포인터임(이중 포인터).

② 배열의 포인터

- 지금까지 이렇게 포인터를 사용해서 배열에 접근했었음

```
int scores[3] = { 80, 90, 100 };
int* p = scores;
```

p는 scores[0]을 가리키고 있음



- 그러나 배열 전체를 다 가리키는 포인터도 있음

```
int scores[3] = { 80, 90, 100 };

int(*p)[3] = &scores; /* OK */
```

  - 그게 바로 배열의 포인터
    - 영어로는 pointer to array
  - 배열 자체의 주소를 저장함
    - 배열은 원래 포인터인데 배열의 주소를 가져오면 이중 포인터가 됨
  - int(*p)[3]
    - (오른쪽 왼쪽 법칙 사용) p는 포인터이다. 그포인터는 요소가 3인 배열을 가리킨다. 그 배열에 담긴 타입은 int임.

```
int scores[5] = { 11, 22, 33, 44, 55 };

int(*p)[3] = &scores; /* 컴파일 오류 */
```

  - 크기가 5인 배열을 크기가 3인 배열 포인터에 대입하려 하면 컴파일 오류

- 배열의 포인터

```
<자료형> (*<변수이름>)[<요소의 수>];
```

  - 어디에 쓸지는 정말 모르겠고 사실 거의 쓰는 경우도 없음
  - 단, 2차원 배열을 매개변수로 받을 때 쓸 수 있음
    - 1차원 배열의 2번째 요소에 접근할 때,
    - arr[1]로도 쓰지만 arr + 1도 된다고 했음
    - 2D배열은? arr[1][2] == *(*(arr + 1) + 2)
  - 배열의 포인터를 이용해서 2차원 배열을 매개변수로 받는 함수

```
void do_magic(int (*matrix)[10], size_t m)
{
    /* 테스트를 위해 하나만 출력 */
    printf("m[1][2]: %d", *(*(matrix + 1) + 2));
}
```

  - 원래 함수

```
void do_magic(int matrix[][10], size_t m)
{
    /* 테스트를 위해 하나만 출력 */
    printf("m[1][2]: %d", matrix[1][2]);
}
```

■ 함수 포인터 예: 퀵 정렬

```
void qsort(void *ptr, size_t count, size_t size,
                      int (*comp)(const void*, const void*));
```

- <stdlib.h> 안에 있는 함수 포인터를 받는 표준 라이브러리 함수
- 실행 속도: O(NlogN)
  - 다른 정렬 알고리즘은 O(N^2)로 느린데 퀵 소트는 빨라서 업계에서 많이 쓰인다
- 매개변수
  - ptr: 정렬하고자 하는 배열의 시작 주소를 전달
    - 배열 말고 다른 자료형도 된다
  - count: 그 배열에 들어가 있는 요소의 수
  - size: 각 요소의 크기
  - `int (*comp)(const void*, const void*)`
    - 두 매개변수 중 뭐가 더 먼저 와야하는지 판단하는 함수를 가리키는 함수 포인터
- 어떤 데이터라도 처리 가능
  - 매개변수 ptr은 void*(void포인터)이므로 어떤 데이터라도 넣을 수 있다
- 자료마다 정렬하는 기준이 다를 수 있음
  - 그 기준에 맞는 정렬 함수를 넘겨주기 위해 세 번째 인자가 존재

    ```
    int (*comp)(const void*, const void*)
    ```
  - 두 매개변수 중 뭐가 더 먼저 와야하는 지 반환하는 함수
    - 즉, 자료를 정렬하는 기준을 알려줌
  - qsort 함수를 호출하는 사람이 이 함수를 따로 구현을 해야 함

void*

- 범용적 포인터
- 어떤 포인터라도 여기에 대입 가능
  - 그 반대도 그냥 대입 가능
    - void 포인터를 char로도 float으로도 아무거나로 대입 가능하다는 것
  - 따라서 어떤 변수의 주소라도 곧바로 대입 가능
  - 매개변수 형으로 void*를 사용하면 어떤 포인터도 받을 수 있는 함수 탄생
- 단, 다음과 같은 경우 다른 포인터로 캐스팅 또는 대입해서 써야 함
  - 역 참조
    - 몇 바이트 읽을지 모르기 때문
  - 포인터 산술 연산
    - 몇 바이트 이동할지 모르기 때문
- void* 예

```
float pi = 3.14f;
void* p;
float* q;

p = &pi;
q = p;

printf("%f\n", *p);                    /* 컴파일 오류 */
printf("%f\n", *(float*)p);
```

```
int add(void* op1, void* op2)
{
    int result;
    result = *op1 + *op2;              /* 컴파일 오류 */
    result = *(int*)op1 + *(int*)op2;  /* OK */
}
```

  - void*인 p에 pi의 주소 대입 가능

    q에 p를 대입 가능
  - 다만 p를 출력할 때는 캐스팅해줘야 함

- void*인 op1과 op2를 그냥 역참조하면 컴파일 오류

다른 언어에도 함수 포인터가 있음

- C#
  - C#에서의 델리게이트(delegate)도 사실 함수 포인터
  - 단, 형 안정성 보장 (type-safe)
    - 다른 타입으로 캐스팅이 안되는 것

```
public delegate void Del(string message);
```

```
public static void PrintMessage(string message)
{
    Console.WriteLine(message);
}
```

```
Del func = PrintMessage;
func("Hello POCU");
```

  - delegate인 Del을 선언해주고
    Del의 변수 func를 선언하고 그안에 PrintMessage 대입
    func를 PrintMessage함수처럼 사용할 수 있다
- Javascript

```
var bar = function() { alert("Hello"); }
var foo = bar;
bar = function() { alert("Good bye"); };
foo(); // "Hello" 출력
```

  - 약한 타입 변수에 함수도 집어 넣을 수 있음(bar)
  - 그 변수를 다른 변수에 집어넣을 수 있음(foo)
  - 변수를 함수같이 사용 가능 foo()