4-3. 포인터, 포인터, 포인터

창의적소프트웨어프로그래밍
2022년도 여름학기
Racin

이번 시간에는

- **포인터**, **포인터**, **포인터**
  - 우리가 적은 **수식**을 컴파일러가 어떻게 deduce해 가며 컴파일을 수행하는지 간단하게 다시 한 번 소개해요
    - 간단한데 디스어셈블리 나옴. 미안해요
  - 몇 가지 예시 코드들을 확인해 보며,
    C 컴파일러의 deduce 규칙들 및 각 **연산자**의 실질적인 의미들을 짚어 봐요
    - 보다 보면 이번 슬라이드 제목이 납득되기 시작할 거예요

이번 시간에는

- 일단 시작해 봅시다.

C 컴파일러의 deduction

- 아래 코드를 봅시다:

```
int number;

double rate = 1.5;

int main()
{
    number = rate + 1.5;

    return 0;
}
```

C 컴파일러의 deduction

- 아래 코드를 봅시다:

```
int number;

double rate = 1.5;

int main()
{
    number = rate + 1.5;

    return 0;
}
```

이 **문장**의 **실행**이 끝난 시점에,
number 자리에는 어떤 **값**이 담겨 있을까요?

C 컴파일러의 deduction

- 아래 코드를 봅시다:

```
int number;

double rate = 1.5;

int main()
{
    number = rate + 1.5;

    return 0;
}
```

이 **문장**의 **실행**이 끝난 시점에,
number 자리에는 어떤 **값**이 담겨 있을까요?

아마도 3이 담겨 있겠지요?

C 컴파일러의 deduction

- 방금 전 코드를 컴파일한 결과를 보면...
  - 보기 쉬운 x86 버전으로, 강사만 ㄱㄱ해 볼게요

```
00F71AEC 8D BD 40 FF FF FF    lea         edi,[ebp-0C0h]
00F71AF2 B9 30 00 00 00       mov         ecx,30h
00F71AF7 B8 CC CC CC CC       mov         eax,0CCCCCCCCh
00F71AFC F3 AB                rep stos    dword ptr es:[edi]
    number = rate + 1.5;
00F71AFE F2 0F 10 05 00 90 F7 00 movsd       xmm0,mmword ptr [rate (0F79000h)]
00F71B06 F2 0F 58 05 30 6B F7 00 addsd       xmm0,mmword ptr [__real@3ff8000000000000 (0F76B30h)]
00F71B0E F2 0F 2C C0          cvttsd2si   eax,xmm0
00F71B12 A3 5C 91 F7 00       mov         dword ptr [_number (0F7915Ch)],eax

    return 0;
```

C 컴파일러의 deduction

- 방금 전 코드를 컴파일한 결과를 보면...
  - 보기 쉬운 x86 버전으로, 강사만 ㄱㄱ해 볼께요

```
00F71AEC 8D BD 40 FF FF FF    lea         edi,[ebp-0C0h]
00F71AF2 B9 30 00 00 00       mov         ecx,30h
00F71AF7 B8 CC CC CC CC       mov         eax,0CCCCCCCCh
00F71AFC F3 AB                rep stos    dword ptr es:[edi]
    number = rate + 1.5;
00F71AFE F2 0F 10 05 00 90 F7 00 movsd       xmm0,mmword ptr [rate (0F79000h)]
00F71B06 F2 0F 58 05 30 6B F7 00 addsd       xmm0,mmword ptr [__real@3ff8000000000000 (0F76B30h)]
00F71B0E F2 0F 2C C0          cvttsd2
00F71B12 A3 5C 91
    return 0;
```

double 덧셈식 rate + 1.5의 경우
적절한 double용 명령어를 써서 잘 만들어 둔 것을 확인할 수 있습니다.

double **형식 수식** 1.5에 대해 즉시값을 쓰지 않고
그냥 메모리 특정 **위치**(컴파일러가 정한)에 1.5를 담아 놓고 가져와 쓰고 있어요.
그렇다고 **수식** 1.5가 **lvalue 수식**이 되는 건 아니에요(우리는 그 **위치**를 특정할 수 없어요)

C 컴파일러의 deduction

- 방금 전 코드를 컴파일한 결과를 보면...
  - 보기 쉬운 x86 버전으로, 강사만 ㄱㄱ해 볼께요

```
00F71AEC 8D BD 40 FF FF FF    lea         edi,[ebp-0C0h]
00F71AF2 B9 30 00 00 00       mov         ecx,30h
00F71AF7 B8 CC CC CC CC       mov         eax,0CCCCCCCCh
00F71AFC F3 AB                rep stos    dword ptr es:[edi]
    number = rate + 1.5;
00F71AFE F2 0F 10 05 00 90 F7 00 movsd       xmm0,mmword ptr [rate (0F79000h)]
00F71B06 F2 0F 58 05 30 6B F7 00 addsd       xmm0,mmword ptr [__real@3ff8000000000000 (0F76B30h)]
00F71B0E F2 0F 2C C0          cvttsd2si   eax,xmm0
00F71B12 A3 5C 91 F7 00       mov         dword ptr [_number (0F7915Ch)],eax

    return 0;
```

그렇다면 이 사이에 끼어 있는 이 명령어는 무얼 하는 친구일까요?

C 컴파일러의 deduction

- 방금 전 코드를 컴파일한 결과를 보면...
  - 보기 쉬운 x86 버전으로, 강사만 ㄱㄱ해 볼게요

```
00F71AEC 8D BD 40 FF FF FF    lea         edi,[ebp-0C0h]
00F71AF2 B9 30 00 00 00       mov         ecx,30h
00F71AF7 B8 CC CC CC CC       mov         eax,0CCCCCCCCh
00F71AFC F3 AB                rep stos    dword ptr es:[edi]
    number = rate + 1.5;
00F71AFE F2 0F 10 05 00 90 F7 00 movsd       xmm0,mmword ptr [rate (0F79000h)]
00F71B06 F2 0F 58 05 30 6B F7 00 addsd       xmm0,mmword ptr [__real@3ff8000000000000 (0F76B30h)]
00F71B0E F2 0F 2C C0          cvttsd2si   eax,xmm0
00F71B12 A3 5C 91 F7 00       mov         dword ptr [_number (0F7915Ch)],eax

    return 0;
```

그렇다면 이 사이에 끼어 있는 이 명령어는 무얼 하는 친구일까요?

맞아요. xmm0(레지스터)에 담긴 double **값**에 대한 int **값**을 eax에 담고 있어요
(ConVerT with Truncation Scalar Double To Scalar Int의 줄임말이래요)

C 컴파일러의 deduction

- 방금 전 코드는 이런 느낌으로 deduce되어 있다고 볼 수 있어요:

```
int number;

double rate = 1.5;

int main()
{
    number = (int)(rate + 1.5);

    return 0;
}
```

C 컴파일러의 deduction

- 방금 전 코드는 이런 느낌으로 deduce되어 있다고 볼 수 있어요:

```
int number;

double rate = 1.5;

int main()
{
    number = (int)(rate + 1.5);

    return 0;
}
```

이런 친구를 'casting **연산자**'라 불러요.
(type) 과 같은 느낌으로 적어요.
C에서는 **수식** 적을 때 **형식**을 그냥 적을 순 없고,
여간해서는 괄호로 둘러 싸 주어야 하도록 되어 있어요.

type 자리에는
조사식 탭에서 볼 수 있는 것들 중 일부를 적을 수 있어요.
일단 (int *) 같은 것은 가능해요

C 컴파일러의 deduction

- 방금 전 코드는 이런 느낌으로 deduce되어 있다고 볼 수 있어요:

```
int number;

double rate = 1.5;

int main()
{
    number = (int)(rate + 1.5);

    return 0;
}
```

아무튼 이렇게 적어 두면
대상 **수식**을 **계산**한 결과**값**'에 대한 int **형식 값**'
...이 전체 **수식**의 **계산** 결과**값**으로 나와요.

다행히 우리 컴파일러는
'double **값**에 대한 int **값**'을 얻는 방법을 알고 있으니,
그 명령어를 써서 그 **값**을 얻도록 컴파일해 줄 거예요.
(이 코드는 아까랑 동일하게 컴파일돼요)

C 컴파일러의 deduction

- 다시 원래 코드로 돌아와 보면…

```
int number;

double rate = 1.5;

int main()
{
    number = rate + 1.5;

    return 0;
}
```

C 컴파일러의 deduction

- 다시 원래 코드로 돌아와 보면...

```
int number;

double rate = 1.5;

int main()
{
    number = rate + 1.5;

    return 0;
}
```

이 덧셈식을 '겉에 (int) 붙은 것처럼' 컴파일한 이유는,
당연하겠지만 = **수식**의 좌항 자리 **수식** 때문이지요?

전체 **수식**이 number + (rate + 1.5)였다면
double이 더 우세하겠지만
여긴 = **수식**이기에 무조건 좌항 **수식**의 **형식**이 이겨요.

C 컴파일러의 deduction

- 다시 원래 코드로 돌아와 보면...

```
int number;

double rate = 1.5;

int main()
{
    number = rate + 1.5;

    return 0;
}
```

여기서 오늘 조금 더 부연한다면,
= **수식** 좌항 자리에 적어 둔 '**수식**으로써 적어 둔 **이름**' number
...에 대한 **선언**에 int가 적혀 있기 때문이라 말할 수 있어요.

여기 int 대신 double을 적어 놨다면
cvttsd2si / mov 명령어 콤보 대신 그냥 movsd 명령어 썼을 듯!

C 컴파일러의 deduction

- 요약하면...
  - 컴파일러의 deduction은 주로 '**형식**'에 대해 수행돼요
    - '**수식**의 **형식**'에 대해 수행된 예시를 방금 구경했고,
      C++ 에서는 더 집요한 deduction 규칙이 적용되거나, 심지어 우리가 직접 지정할 수도 있어요
  - 컴파일러는 각 **수식**의 **형식**에 대한 deduce를 수행할 때 context를 고려해요
    - 예를 들면 '다음' **연산자**를 통해 엮이는 다른 **수식**의 **형식**을 참고해요
    - 왜냐면, deduce를 하는 이유가 보통은 '무슨 형식 덧셈 할 지 정하기 위해' 등이기 때문이에요
  - Deduce 과정에서 필요한 경우, 컴파일러는
    각 **수식**에 있는 '**수식**으로써 적은 **이름**'들에 대한 **선언**을 참고해요
    - 달리 말하면, deduction은 컴파일 시점에 발생해요(**runtime**이 아님!)

C 컴파일러의 deduction

- 좋아요. 이 정도 구경해 두면 일단 오늘 본론을 시작할 준비는 끝난 듯 해요.
  - 이전에 몇 번 등장했으니 지금은 그럭저럭 익숙할듯

포인터, 포인터, 포인터

- 좋아요. 이 정도 구경해 두면 일단 오늘 본론을 시작할 준비는 끝난 듯 해요.
  - 이전에 몇 번 등장했으니 지금은 그럭저럭 익숙할듯

- 그러면 이제,
  내가 **선언**한 각 **이름**들을 **수식**으로써 적을 때
  컴파일러가 이들을 어떤 규칙에 따라 deduce하여 다루는지 살펴 봅시다.

- 이번 복습의 최종 보스에 해당하는 만큼,
  각 주제들을 예시 파일로 만들어서 올려 두었어요.
  - 다운로드한 다음 하나씩 올려 놓고 실행해 가며 감상해 봅시다
  - 조금 뒤에 강사와 함께 천천히 구경해 볼께요(그동안 쉬어도 좋아요)

마무리

- 여기까지가 우리 앞 수업 복습이에요.
  - 원래 창소프 수업은 이런거 다 안다 가정하고 스타트함
  - 아무튼 보느라 고생 많았어요
- 내일부터는 C++로 넘어가기 위한 중간 다리에 해당하는 것들을 몇 가지 가져와서 구경해 볼께요.
- 오늘은 최종 목표는 따로 없어요.
  - 느낀 점 적은 다음 집에 갑시다