펭귄 스터디 내용04

작성자 : 류성수
작성일자 : 24.06.30

url : https://www.youtube.com/watch?v=m_qObdtatVg&list=PLz--ENLG_8TMdMJIwyqDIpcEOysvNoonf&index=5

● 변수 선언에 관한 팁

컴퓨터 리소스가 풍부하다면, 변수는 해당용도에 맞게 객관적으로 알아볼 수 있는 변수명과 용도별로 각각 선언을 해주자
이로 인한 버그 가능성을 크게 줄일 수 있다.

● 연산자 우선순위

기호 [1]	연산 유형	associativity
`[ ]` `( )` `.` `->` `++` `--`(후위)	식	왼쪽에서 오른쪽
`sizeof` `&` `*` `+` `-` `~` `!` `++` `--`(전위)	단항	오른쪽에서 왼쪽
형식 캐스팅	단항	오른쪽에서 왼쪽
`*` `/` `%`	곱하기	왼쪽에서 오른쪽
`+` `-`	더하기	왼쪽에서 오른쪽
`<<` `>>`	비트 시프트	왼쪽에서 오른쪽
`<` `>` `<=` `>=`	관계	왼쪽에서 오른쪽
`==` `!=`	같음	왼쪽에서 오른쪽
`&`	비트 AND	왼쪽에서 오른쪽
`^`	비트 제외 OR	왼쪽에서 오른쪽
`|`	비트 포함 OR	왼쪽에서 오른쪽
`&&`	논리 AND	왼쪽에서 오른쪽
`||`	논리 OR	왼쪽에서 오른쪽
`?` `:`	조건식	오른쪽에서 왼쪽
`=` `*=` `/=` `%=` `+=` `-=` `<<=` `>>=` `&=` `^=` `|=`	단순 및 복합 할당 [2]	오른쪽에서 왼쪽
`,`	순차적 계산	왼쪽에서 오른쪽

MS 공식 홈페이지의 연산자 우선순위이다. 가장 상위가 최우선적으로 처리하는 연산자이고, 밑으로 갈수록 우선순위가 줄어든다.
컴파일러는 이 규칙에 의거하여, 컴파일을 해서 어셈블리 코드를 뱉어낸다.

1. 연산자 우선순위에 관한 팁

우선순위 규칙에 의거하여 코드를 작성하면, 코드는 매우 단순해지고, 소스파일자체의 크기에도 영향을 줘서 파일이 가벼워진다는 장점이 있지만, 그러한 장점보다 단점이 더 크기 때문에 너무 의존해서 사용하는 것은 추천하지 않는다고 말씀하신다.

강사님은 첫 번째 이유로 항상 **팀플레이에서 발생하는 문제**를 꼽으셨다.
모두가 우선순위 규칙에 빠삭하지 않기 때문에, 우선순위 표를 옆에 띄워놓고 코드를 해석해도 사람마다 다르게 해석할 수 있다는 가능성을 얘기했다.

두 번째 이유로 코드의 본인 **스스로 코드의 논리적 오류를 일으키지 않기 위해서** 지양하지 않아야 한다고 꼽으셨다

나 스스로도 이러한 규칙을 수업시간에 배웠고 이러한 규칙을 이용해서 손코딩해서 푸는 문제들도 시험문제로 나왔었던 것을 기억한다.
근데, 솔직히 지금은 잘 모르겠다. 후위연산과 전위 연산의 우선순위가 이렇게 높은지도 이번에 다시 알게되는 것 같다.
그래서 강사님이 해당 이유를 말씀하시자마자 무슨말씀을 하고싶은지 알아챌 수 있었다.

● if문에서 논리연산의 short 서킷 룰

만약, (A&&B) 인데, A가 거짓이면, CPU는 A만 확인하고 B를 읽지도 않는다.
그리고, (A||B) 인데, A가 참이면, CPU는 역시 A만 확인한다.

이는 CPU의 계산효율성 때문에 발생하는 현상이다. 또한, 컴파일러가 소스코드를 처리하는 과정에서도 살펴볼 수 있따.
컴파일러는 기본적으로 왼쪽 -> 오른쪽 순서로 읽기 때문이다.

간단한 예시로 확인한다면, 다음과 같은 코드예시를 들 수 있다.

```c
int thisisA()
{
        cnt ++;
        printf("A func has been executed!\n");
        return 0;
}

int thisisB()
{
        cnt ++;
        printf("B func has been executed!\n");
        return 1;
}

int thisisC()
{
        cnt++;
        printf("C func has been executed!\n");
        return 1;
}

int thisisD()
{
        cnt++;
        printf("D func has been executed\n");
        return 0;
}

void main()
{
        if(thisisA() && thisisB())
        {
                printf("And logical operator has been executed!\n");
        }

        if(cnt != 2)
        {
                printf("But both of func are not executed : cnt = %d\n",cnt);
        }
        cnt = 0;

        if(thisisC() || thisisD())
        {
                printf("Or logical operator has been executed!\n");
        }

        if(cnt != 2)
        {
                printf("But both of func are not executed : cnt = %d\n",cnt);
        }

}
```

현재, A는 0을 반환하고, C는 1을 반환한다. 또한, 각 함수는 cnt의 값을 증가시킨다
그리고 각 논리연산종류를 통과하면, cnt의 값이 2인지를 확인한다.

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy04# sudo gcc logical_sequence.c -o lo
gical_sequence.out
root@LAPTOP-BQ4AK11N:/project/24-Cstudy04# ls
logical_sequence.c  logical_sequence.out
root@LAPTOP-BQ4AK11N:/project/24-Cstudy04# ./logical_sequence.out
A func has been executed!
But both of func are not executed : cnt = 1
C func has been executed!
Or logical operator has been executed!
But both of func are not executed : cnt = 1
```

이를 실행하게 되면, A와 C만을 확인했다는 것을 확인 할 수 있다.
그래서 cnt의 값이 1로 나와서 cnt 값을 확인하는 조건문안의 문자열이 출력된 것을 확인 할 수 있다.

굉장히 사소한 문제이고, 이러한 부분으로 나또한 예전에 큰 고생을 했었고 지금은 이 부분에 대해서 숙지한 상태이다.
강사님께서 이러한 세세한 부분을 짚어주셔서 다시한번 중요성을 확인하는 계기가 된 것 같다.