데이터 구조 디버깅 에세이(VSCode)

21900102 김민혁

처음으로 “비주얼 스튜디오 2019”를 사용하기 위해 새 프로젝트 만들기 메뉴를 클릭하였다. 그런 다음 빈 프로젝트를 클릭하여 프로젝트 이름과 솔루션이름, 또한 프로젝트 위치를 설정하여 프로젝트를 시작한다. 여기서 프로젝트란 컴파일의 결과물로 하나의 실행파일 혹은 라이브러리 등을 만들어내는, 하나의 프로그램을 만드는 묶음이고, 솔루션은 하나 이상의 프로젝트가 모인 집합이라고 볼 수 있다. 고로 대형 프로젝트를 만들 때 프로젝트들을 묶어 하나의 솔루션으로 관리하는 것이 좋다.

프로젝트를 세팅하기 위하여 가장 먼저 소스파일을 추가해야 한다. 오른쪽 메뉴의 “소스파일” 아이콘을 클릭하여 추가를 한다 이때 추가 옵션에는 새 소스를 추가할 수도 있고 기존 소스를 추가할 수 있다. 나는 새소스를 추가하여 교수님의 infix.cpp와 infixDriver.cpp파일을 복사하였다. 다음 파일 들에는 헤더파일 즉, include파일이 필요 없지만 include 파일이 필요하다면 헤더파일에 추가를 할 수 있다. 또한 현재 빌드하려는 프로젝트의 세팅을 위하여 프로젝트의 속성메뉴로 들어간다. 세팅해야 할 것은 C/C++ 메뉴의 일반에서 추가 포함 디렉터리의 경로를 지정해 주는 것이다. 또 세팅해야 할 것은 링커 메뉴의 일반에서 추가 라이브러리 디렉터리의 경로를 지정해고, 입력애서 추가할 라이브러리 자체를 지정하여야 한다. 위의 추가 포함 디렉터리를 지정해주면 include파일의 경로가 지정되는 것이므로 앞서 추가한 헤더파일은 추가 하지 않아도 정상적으로 돌아간다. 즉 생략이 가능하다는 말이다.

이를 실행하여 보기 위해 비주얼 스튜디오 위의 메뉴판에 솔루션 빌드 기능을 통하여 infix.cpp와 infixDriver.cpp파일을 빌드하였다. 이후 먼저 그냥 실행 값을 알기 위하여 디버그 메뉴에 디버그 하지 않고 시작을 하여 보았다. 그 결과 다음과 같이 무한루프가 돌아가는 것을 확인할 수 있었다.

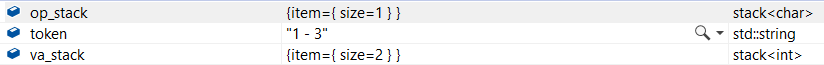


이 버그를 확인하기 위하여 다시 디버그 메뉴에 디버깅시작을 하였다. 그 결과 위의 결과와 같이 무한루프를 돌고 있었고 버그의 위치를 확인하기 위하여 위의 일시정지 아이콘처럼 생긴 모두 중단 아이콘을 클릭하였다 그랬더니 infix.cpp코드 안의

while (!op\_stack.empty()) {

cout << "your code here\n";

}

다음과 같은 while구문에서 무한루프가 돌고 있었다는 것을 확인할 수 있었다. 그리고 밑에 다음과 같이  
현재 변수들의 값이 나와 있었다. 자세히 보니 현재 무한루프 중인 whlie구문은 op\_stack이 비어 있지 않을 때 돌아가는데, 위의 값을 보면 op\_stack의 size가 1로 비어있지 않은 것을 볼 수 있다. 이 이유때문에 무한루프가 도는 것이다. 교수님께서 주신 infix알고리즘을 보면 op\_stack이 빌 때까지 va\_stack에서 두 값을 t, op\_stack에서 하나의 값을 top(), pop()하여 계산한후 그 값을 다시 va\_stack에 push해야 한다. 그래서 위의 코드를

while (!op\_stack.empty()) {

int answer = compute(va\_stack, op\_stack);

va\_stack.push(answer);

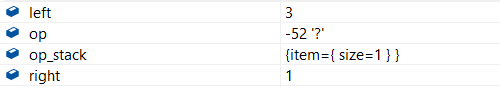
}

와 같이 수정하였다. 맞게 수정했는지 확인하기 위해 아까 멈춰져 있던 지점에 Break Point를 하여 아까 전의 변수들이 각각 어떤 값을 갖게 되는지 확인하였다.



그 결과 다음과 같이 op\_stack의 size가 0인 것을 알 수 있었다. 그리고 이제 while을 벗어날 수 있는지 확인해야 한다. 이를 위하여 위의 화살표모양 메뉴 중 한 단계씩 코드실행 기능을 누르면 프로그램이 다음 단계로 넘어가는 것을 볼 수 있다.

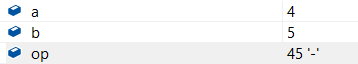
계속을 클릭하여 실행을 계속 해보면, infix의 계산 값이 이상한 것을 알 수 있다. 그래서 compute함수에 BreakPoint를 두고 디버깅을 해보기로 하였다. 실행하여 compute함수에서 break한 후 한 단계씩 코드 실행을 계속 클릭하며 각 변수들의 값의 변화에 대하여 관찰하였다. 그런데 갑자기 첫번째 예시의 “1-3”에서 원래대로라면 left에 1이 들어가야 하는데 left에 3이 들어간 것을 확인할 수 있었다.



그래서 left와 right를 서로 변경하여 다시 디버깅 하였다.



다음과 같이 값이 제대로 들어간 것을 확인할 수 있다. 이렇게 Step1까지는 제대로 된 값을 얻을 수 있었다. 그러나 이번에는 Step2의 값이 맞지가 않았다. 아마도 Step2가 Multi-digits를 계산하는 것이라서 token[i]마다 각각 받아들이게 되어 발생한 오류라고 예상하였다. 그래서 이번에는 Step2에 BreakPoint를 두고 디버깅 하였다. 그렇게 하여 한 단계씩 값의 변화를 관찰하였더니,





다음과 같이 “1 + ( 234 - 5 )”의 234-5대신에 한자리수만 인식하여 4-5를 계산하고 있음을 확인할 수 있었다. 그래서 교수님의 알고리즘대로 숫자가 나오면 먼저 읽고 다 읽으면 push할 수 있도록 코드를

if (isdigit(token[i])) {

int va = 0;

while(isdigit(token[i])) {

va = va \* 10 + (int)(token[i] - '0');

i++;

}

i--;

va\_stack.push(va);

DPRINT(cout << " va\_stack.push: " << va << endl;);

}

다음과 같이 바꾸었다. 수정 후 다시 디버깅하여 BreakPoint에서 값의 변화를 관찰하니



다음과 같이 va 변수의 값이 234로 알맞게 들어가는 것을 확인할 수 있었다. 그래서 계속을 클릭하여 코드를 계속 실행하였더니 프로그램이 알맞게 돌아가는 것을 확인할 수 있었다.

