< 디버깅 에세이 Debugging Essay >

22000690정이주

// On my honour, I pledge that I have neither received nor provided improper

// assistance in the completion of this assignment. Signed: \_\_\_\_Jungyiju\_\_\_\_\_\_ Student Number: \_\_\_\_22000690\_\_\_\_\_

***----- Debugging 프로그램을 사용하면서 느꼈던 점 -----***

Debugging을 사용하지 않았다면 이번 과제를 하는데 매우 큰 어려움이 있었을 것 같다. 처음에 Visual Studio Code로 파일을 컴파일 하였는데 어떤 부분에서 잘못 값이 push되었는지 어떤 반복문에서 빠져나오지 못하고 무한루프를 도는지 찾기가 매우 어려웠다.

그리고 항상 내가 생각한대로 코드가 돌아가 output을 내는 것이 아니기에 내가 어떤 부분에서 잘못 생각하였는지 또는 코드를 구현함에 있어 내가 놓치는 부분이 어떤 부분인지 생각해내기가 매우 어려웠다. 하지만 Xcode의 debugging을 사용하여 내가 확인하고자 하는 부분에 breakpoint를 걸어 하나씩 눈으로 코드가 돌아간 후의 결과를 볼 수 있었기에 훨씬 수월하였던 것 같다. 또한 각각의 스택에 들어간 내용을 종이에 그려서 내가 머릿 속으로 돌려본 코드를 하나씩 검산할 때 정말 좋았고 정확히 확인 해 볼 수 있어 내가 제대로 개념들과 코드를 이해하였는지 확인할 수 있어 좋았다.

과제를 해나가는 과정 중에 많은 에러들이 있지만 error: Segmentation fault이 문구가 나왔을 때 bebugging 프로그램이 정말 큰 도움이 되었던 것 같다. 물론 다른 에러들을 수정할 때에도 bebugging 프로그램을 사용하는 것이 큰 도움이 되었지만 그래도 error 문구를 보면 어림잡아 어떤 부분에서 막히는 것 인 지 알 수 있기 때문이다. 하지만 error: Segmentation fault이 에러가 떠 구글링을 해보았을 땐 메모리 관련 error 문구인 것 만 알 수 있었고 정확히 어디라인에서 어떤 문제가 있는 지 알 수가 없었다. 그래서 처음에는 evaluate에 breakpoint를 걸었는데 한 조건문 안의 반복문에서 무한루프에 빠진다는 것을 알게 되었다.

정말 debugging 프로그램이 없었다면 정말 찾기 힘들었을 것 같고 알아내기 힘들었을 것 같았다. 또한 정말 pset7을 하면서 내가 생각한 것이 맞는 것인지 모를 정도로 복잡했던 순간들이 있었는데 그때마다 한단계 한단계 씩 보여주는 step into을 사용하여 생각 들을 정리 할 수 있었고 내가 이해한 개념들과 상황들을 눈을 확인하면서 내가 잘 못 생각했던 것들을 바로잡고 제대로 이해할 수 있었다. 이번 과제를 통해 bebugging의 필요성과 중요성을 알게 되었다.

(첫번째 사진)

🡪 debugging을 통해 evaluate 함수 안에 맨 마지막의 정답을 계산하는 부분에서‘ ( ‘가 op\_stack에 남아있어 while문의 조건인 “ ! op\_stack.empty() ”를 만족하지 못해 무한 루프에 빠지고 이로 인해 값이 제대로 계산이 되지 않고 있다는 것을 확인했었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

(두번째 사진)

🡪 조건문 안에 while문이 있는 곳에서 다음에 계산을 진행 할 때무한 루프로 돌고 값이 제대로 계산되지 않았다. 그러므로 while으로 계산을 완료한 후에 op\_stack의 맨 위에 부분에 다른 operator 가 아니라 ‘ ( ‘ 이게 들어있다면 삭제해달라는 코드르 추가하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

***----- 이번 과제에서 내가 debugging 을 사용한 예시 -----***

위의 사진 두개는 내가 코드를 작성하고 에러를 찾는 과정에서 debugging을 사용한 예시이다. Token이 ‘ ) ’일 때 이전에 stack에 저장된 값들을 모두 반복적으로 계산해야 하므로 op\_stack에 남아있는 operator가 ‘ ( ‘ 일 때 까지 값을 계산해줘야한다. 하지만 계산을 완료하고 반복문은 종료가 되지만 op\_stack에 ‘ ( ‘ 값이 남아있어 evaluate 함수의 마지막 부분에서 op\_stack이 비어있지 않을 때까지 반복 루프를 돌리는 코드에서 무한 루프에 빠지게 된다. (첫번째 사진) 처음에 디버깅을 하지 않고 코드를 돌렸을 때는 정확히 어떤 반복문에서 반복되는지도 왜 프로그램은 종료가 되지 않고 값이 더이상 출력되지 않는지 찾을 수가 없었다. 하지만 디버깅 프로그램을 통해 evaluate 함수의 각 조건문과 반복문에 breakpoint를 설정하여 프로그램이 돌아가는 과정을 한 라인 씩 확인하는 과정을 통해서 내가 작성한 코드에서는 괄호가 들어간 수식이 input 값으로 작성된다면 op\_stack이 비워지지 않아 무한 루프에 빠지게 된다는 것을 확인 할 수 있었다. 처음에는 반복문 안에 op\_stack을 pop하는 코드를 작성하였다. 하지만 그랬더니 내가 원하는 ‘ ( ‘ 가 사라지지 않고 ‘ + ‘ , ‘ – ‘, ‘ \* ‘, ‘ / ‘ 들 중 하나가 지워져 다음 계산된 값에 영향을 미치거나 error: Segmentation fault 가 발생하였던 것 같다. 그래서 다시 한번 op\_stack 안의 값을 확인하면서 debugging으로 한 라인 씩 확인한 결과 반복문을 종료한 후의 op\_stack의 맨 위의 값이 ‘ ) ‘ 인 경우에 삭제해줘야 하는 코드를 추가적으로 작성해야한다는 것을 알게 되어 if문을 사용하여 pop을 해주는 코드를 작성하였다.

if(op\_stack.top() == '('){

op\_stack.pop();

}

이것이 내가 추가한 if문 코드이다.

만약 op\_stack에 들어간 operator가 ‘ ( ‘ 이면 지워 달라는 코드를 추가함으로써 코드가 빌때 까지 돌아가는 마지막 while 반복문에서 무한 루프에 빠지지않게 해줬다. 처음에는 무조건 op\_stack에 가장 위의 값을 지워주도록 다른 조건을 덧 붙이지 않았으나 그렇게 되면 여러 괄호가 포함된 수식을 계산 할 때 내가 원하는 대로 프로그램이 돌아가지 않았고 값의 결과에도 영향을 끼치게 되었다. 그러므로

If ( op\_stack.top() == ‘ ( ‘와 같은 조건을 추가해 줘야지만 결과 값에도 영향을 끼치지 않고 내가 원하는 순간에만 ‘ ( ‘이 지워진다는 것을 확인 할 수 있었다.

***----- Xcode debugging 사용법-----***

(두번째 사진에 표시된 숫자)

1 : Step Over로 코드를 한 줄씩 읽어 내려갑니다.

일반적으로 코드가 적용되는 과정을 차례차례 확인하기 위해 많이 사용합니다.

2 : Step into로 breakpoint로 설정한 라인의 함수나 변수로 바로 넘어가게 만듭니다.

3 : Step into로 들어간 함수나 변수를 빠져 나올 때 사용합니다.

화면의 하단에 위치한 콘솔 화면으로 디버깅을 할 때 변수의 값들이 어떻게 변하는 지 확인 할 수 있습니다.

\*\* 디버깅이 중요한 이유는 프로그램이 찾지 못하는 오류를 찾아 낼 수 있고 쉽게 알기 어려운 오류를 찾아 내는데 큰 도움을 주기 때문입니다\*\*