Debugging with Xcode

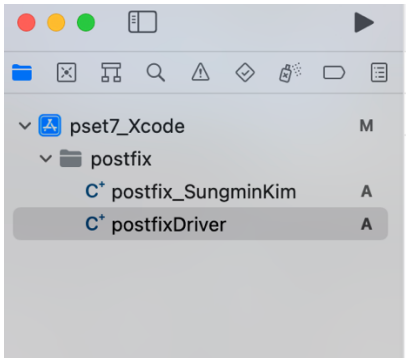
21900112 김성민

지금까지 강의를 들으면서 과제들을 수행할 때 vscode에서 소스코드를 편집하고 터미널에서 커맨드 입력을 통해 실행하는 식으로 진행하였다. 지금까지는 비교적 간단한 코딩이고, 만약 코드에 오류가 생겨도 출력되는 에러 메세지를 차분히 잘 읽어보면 몇 번째 줄에서 어떤 종류의 오류가 일어났는지 확인을 쉽게 할 수 있었고, 그렇기에 디버깅 또한 수월하게 할 수 있었다. 하지만 지난주부터 stack을 배우고 이를 이용한 과제들이 나오면서 터미널만 사용해서 디버깅을 진행하기에는 어려움이 있었다. 특히, stack을 사용하는 과정에서 잘못된 접근을 사용하게 된다면 어떤 부분에서 어떤 문제가 생겼는지 출력해주지 않고 바로 segmentation fault가 뜨면서 프로그램이 종료된다. 그렇기에 우리가 만든 코드에서 stack을 어떻게 사용하고 있는지를 보면서 코딩을 하고 싶어도, 어디서 잘못된 것인지 알 수가 없다. 이러한 문제점을 해결하기 위해 우리는 이제부터 Visual Stutdio나 Xcode를 사용하여 코딩을 할 것이다. 이제부터는 내가 Xcode를 사용하면서 익힌 사용방법들과 진행 과정을 소개하게 될 것이다.

사실 이번 과제에서는 stack사용에 어려운 점이 없어서 코딩 자체는 이전과 동일하게 vscode에서 진행하고 터미널에서 결과를 출력하는 식으로 진행하였다. 하지만 이는 어디까지나 이번 과제에서 사용하는 stack과 코드의 진행 방식이 비교적 간단하기 때문에 가능한 것이지, 앞으로 더욱 많은 stack이나 많은 data들을 사용하여 코딩을 할 때에는 이러한 방식이 어려울 수 밖에 없다. 그래서 이후의 활용을 위해 Xcode에서 디버깅을 하는 방법을 공부하였다.

우선, 내가 현재 사용하고 있는 노트북이 mac이기 때문에 Visual Studio 대신 Xcode를 사용하여 디버깅을 진행하였다.

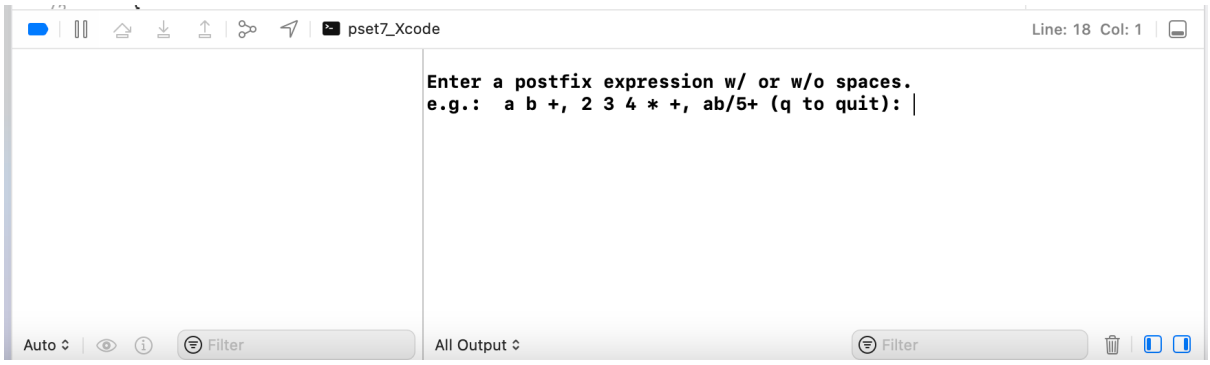
디버깅을 진행하기 위해서는 우선 새로운 프로젝트를 만들고, 터미널을 사용하여 디버깅을 진행해볼 것이기 때문에 Command line tool을 선택하여 준다. 이러한 기본적인 프로젝트 기초 설정은 교수님의 GitHub repository에 그 방법이 올라와 있기 때문에 생략하도록 하겠다.



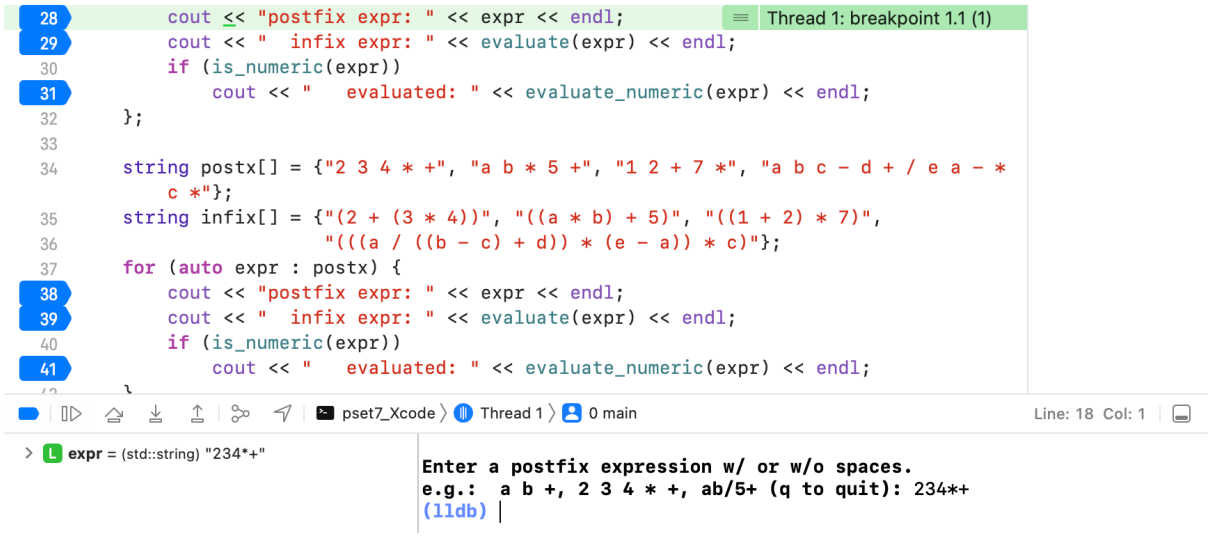
프로젝트를 생성한 후, 위 사진에 보이는 것처럼 프로젝트 아래에 group이 있는 것을 찾을 수 있을 것이다. 기본적으로 처음 프로젝트를 생성하는 것이라면, 그 group안에 main이 하나 존재할 것인데, 우리가 디버깅을 하고 싶은 파일을 불러와서 진행하고 싶다면 main을 지워줘야 불러온 파일을 실행할 수 있을 것이고, 그게 아니라면 main에서 코딩을 진행한 후, 이름을 원하는 대로 바꿔주면 될 것이다. 여기서 한가지 유의할 점이 있는데, 바로 파일을 불러올 때, 현재 선택된 곳이

어딘지 확인을 꼭 잘 해주어야 한다. 프로젝트에 파일을 처음으로 불러올 때는 아무 생각 없이 파일을 불러왔었다. 그렇게 하고 run을 시도해보니 자꾸 "5 duplicate symbols for ~" 오류와 "Undefined symbol: ~~~ " 오류가 뜨면서 진행조차 되지 않았다. Vscode에서 완성하고 문제 없음을 확인한 소스코드 파일이었기 때문에 코드 자체에서 이상이 있는 것 같지는 않다고 판단하여 Xcode 자체의 설정들을 하나씩 살펴보았다. 그러던 중, 불러온 파일들인 postfix.cpp과 postfixDrvier.cpp이 pset7_Xcode의 하위 폴더인 postfix가 아니라 pset7_Xcode 프로젝트 자체에 그냥 들어가 있는 것을 확인할 수 있었다. 또한, "Undefined symbol: ~~~ " 오류에서 define되지 않았다는 variable들을 잘 살펴보니 그 variable들이 모두 함수의 이름임을 확인할 수 있었다.
이를 통해 프로젝트는 main함수가 있는 파일을 자동으로 식별하고, 이 파일만 compile하여 실행시키기 때문에 postfix.cpp에 선언되어 있는 함수들을 undefined라고 판단하고 compile error가 발생한다는 것을 알 수 있게 되었다.
command line으로 나타내자면 g++ -std=c++11 postfixDriver.cpp -o driver , ./driver 을 실행하는 셈이다. 하지만 터미널에서 executable file을 만들 때에는 g++ -std=c++11 postfix.cpp postfixDriver.cpp -o driver , ./driver 을 실행해야 컴파일이 가능하다. 즉, 두 file을 link해야 하는 것이다.
그렇기에 postfix.cpp와 postfixDriver.cpp를 postfix폴더로 이동하여 run 하였더니 정상적으로 build가 되는 것을 확인할 수 있었다.

이제 코드가 실행이 되니 본격적으로 디버깅을 진행해보도록 하겠다.



지금 이 화면이 run을 한 후의 Xcode의 콘솔 창이다. 이 화면을 보면 사용자로부터 postfix 식을 입력 받기 위해 기다리고 있는 모습을 볼 수 있다. 처음에 프로젝트를 설정 할 때, command line tool을 선택하였기 때문에 콘솔 또한 터미널을 사용하는 것을 확인할 수 있다.

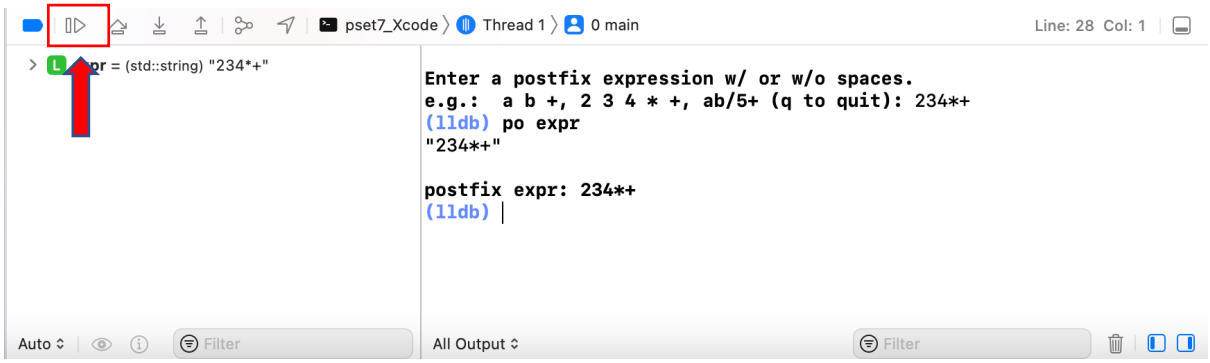


위 화면은 234*+라는 postfix 식을 입력하고 엔터키를 눌러 코드를 진행시킨 모습이다. 콘솔 하단에 (lldb)라는 메세지가 출력되면서 코드가 멈춰있는 것을 확인할 수 있다. 이는 breakpoint를 사용한 것인데, 사진의 왼쪽에 보면 line을 나타내는 숫자가 파란색으로 표시된 부분이 있다. 이것이 해당 line을 breakpoint로 설정하겠다는 뜻인데 이러한 breakpoint들로 인하여 코드가 진행되다가 이것을 만나면 실행을 멈추고 (lldb)를 출력한다. Breakpoint를 설정하면 멈춘 시점에서 여러 변수들이나 조건 등을 확인할 수 있다. 이때 (lldb)를 사용하게 되는데, 콘솔에 해당 command를 입력하면 그에 따른 여러 결과들을 확인할 수 있다. 만약 어떤 변수가 이 시점에서 어떤 값을 가지고 있는지를 확인하고 싶다고 하면 "po (variable name)"을 통해 확인할 수 있다.

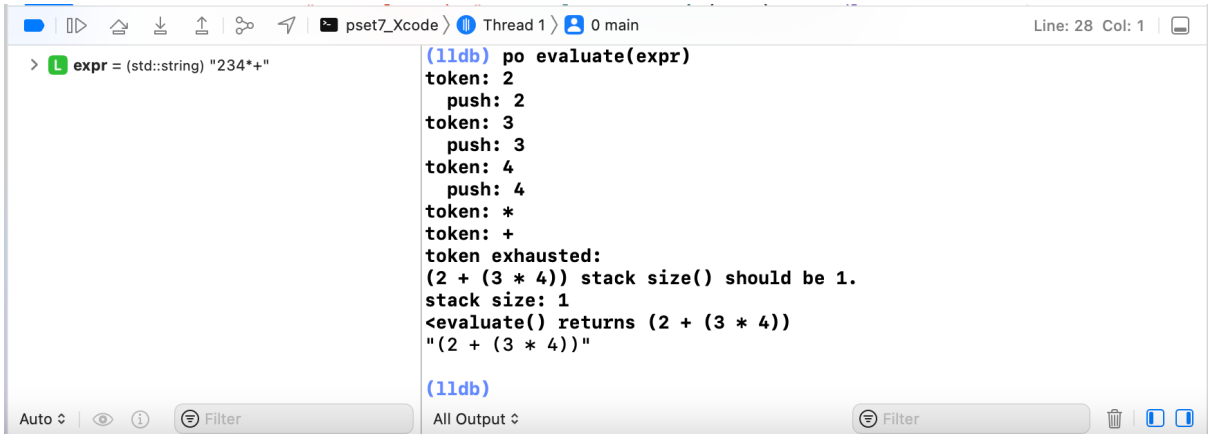
```
Enter a postfix expression w/ or w/o spaces.
e.g.:  a b +, 2 3 4 * +, ab/5+ (q to quit): 234*+
(lldb) po expr
"234*+"

(lldb)
```

위 사진에 나와있는 것처럼 "po expr" command를 입력하였다. Expr은 사용자로부터 입력받은 postfix 식을 저장하는 string형 변수이다. 이 변수에 어떤 값이 저장되어 있는지 확인하기 위해 command를 입력하였더니, "234*+"처럼 변수에 저장된 값을 출력하여 준 것을 볼 수 있다.



위 사진은 코드를 진행시켜 다음 breakpoint까지 실행시킨 결과를 보여준다. 빨간색 화살표로 표시된 부분을 클릭하거나 mac의 경우 control+command+Y를 누르면 continue를 하게 되며 다음 breakpoint까지 코드가 진행되게 된다. 바로 다음 줄에 breakpoint가 걸려 있기 때문에 한 줄을 실행한 후, 또 다시 break가 걸린 것을 볼 수 있다.



위 사진은 breakpoint에서 po command를 입력한 것이다. 현재 breakpoint를 지나게 되면 evaluate함

수를 호출하여야 하기 때문에 evaluate함수에 문제가 없는지 확인하기 위해 “po evaluate(expr)” command를 통해 함수의 출력 결과를 먼저 확인하는 것이다. 중간에 bold 글씨체로 출력되는 것은 함수의 진행 과정에 따른 출력 결과들을 DPRINT를 사용한 부분까지도 모두 출력하여 주는 것이다. 또한 마지막에 “(2 + (3 * 4))”라고 출력이 되어 있는데 return type이 string인 evaluate함수의 반환 결과까지 함께 출력해서 보여준다. 이런 식으로 변수의 내용 뿐만 아니라 함수의 결과 또한 먼저 출력하여 확인해보면서 디버깅 및 코딩을 진행하면 문제가 있는 부분을 찾기도 쉬울 것이다.

지금은 완성된 상태에서 디버깅 하기에 함수의 결과와 stack의 최종 결과만 출력해볼 수 있고 postfix.cpp에서 사용한 stack들의 status를 전부 확인할 수 없다.
만약 step-by-step으로 디버깅을 진행하고 싶다 한다면, postfix.cpp를 코딩할 때 그 파일 자체에 main함수를 생성하여 실행시킨 다음, for문에 breakpoint를 설정하여 반복이 한 번 진행될 때마다의 stack의 상황이 어떻게 되는지 po command를 통해 확인하면서 디버깅을 하며 코딩한다면 훨씬 복잡한 코딩을 해야 하는 상황이라도 visualizing을 통해 훨씬 수월하게 할 수 있을 것이다.

지금까지 vscode와 터미널에서만 코딩을 하면서 다른 오류들은 어느 정도 수월하게 원인을 찾을 수 있었는데 data 접근을 잘못해서 생기는 segmentation fault들은 에러 메세지를 출력해주지도 않고 종료하기 때문에 코딩할 때 가장 어렵고 두려운 에러들 중 하나였는데 이번 기회를 통해 Xcode의 디버깅 모드를 배웠고, 이제 그러한 문제들이 있을 때 쉽게 해결할 수 있을 것 같다는 자신감을 얻게 되었다!