**Debugging using Visual Studio**

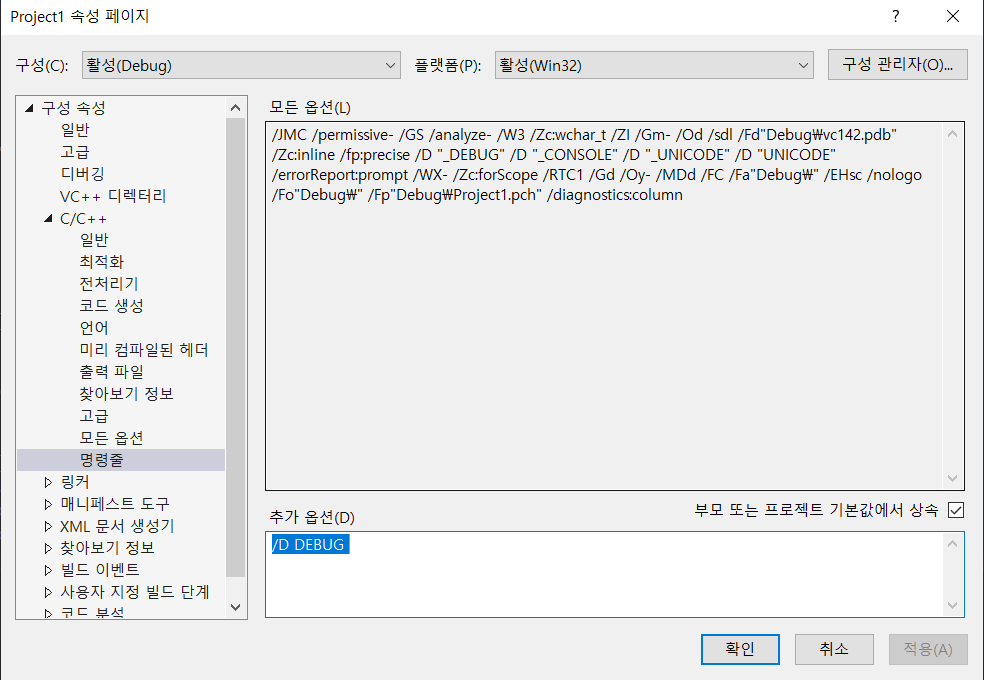
21600124 김우빈

Visual Studio는 마이크로소프트에서 개발한 IDE로, 기존에 사용하던 ATOM에 비해 몇 가지 부분들에 있어서 굉장한 편리성을 제공한다. 후에 언급하는 부분들은 본인이 ATOM을 사용하여 개발을 하다가 Visual Studio를 사용하게 되면서 느낀 편리한 점들이고, 그 중에 특별히 Visual Studio에서 Debug가 얼마나 간편하도 다양한 기능을 제공하는지에 대해 분석해 보려고 한다.

첫 번째로는 프로젝트 빌드하기가 쉽다는 것이다. ATOM은 기존에 프로젝트를 만들기 위해서는 Terminal에 들어가서, 해당 디렉토리를 찾아서 들어가고, 컴파일할 파일들을 찾은 다음, 헤더파일이 있는 디렉토리 경로를 추가하고, 라이브러리의 경로도 추가해주어야 했다. 이것을 모두 사용자가 직접 Typing해서 작업을 하였다. 이런 번거로움 때문에 Make라는 빌드 툴을 사용하기도 하였으나, 새로운 프로젝트를 만들 때마다 매번 그에 맞는 새로운 Makefile을 작성해야 하는 것은 상당히 고역이라고 할 수 있었다.

하지만 Visual Studio는 그것을 ‘프로젝트 빌드’버튼 하나로 구현하였다. 물론 이를 하기 위해서는 사전에 몇 가지 설정들이 필요하긴 하다. 예를 들면, 헤더파일의 위치와 라이브러리의 위치정도는 사용자가 추가해주어야 한다. 하지만 그것은 ATOM을 사용하면서 해야 했던 많은 작업들에 비하면 훨씬 수월한 것이었다.

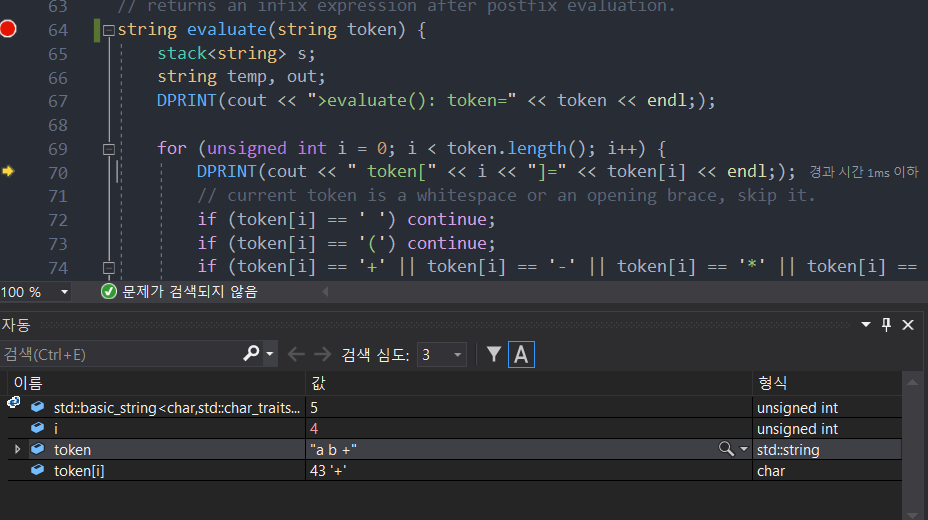
두 번째 장점은 이 보고서에서 중점적으로 다룰 부분인 ‘디버깅’에 관한 측면이다. ATOM에서 디버깅을 하려면 -DDEBUG 옵션으로 컴파일을 할 때마다 입력해 주었다. 하지만 Visual Studio에서는 Project 우 클릭 -> 속성 -> C/C++ -> 명령줄에 원하는 Debugging option을 입력해주고 저장해주면 된다. (<Figure 1> 참조) 이렇게 되면 매번 -DDEBUG를 입력하는 번거로움을 줄일 수 있고, 더 다양한 디버깅 옵션들을 추가할 수 있다.



<Figure 1>

Visual Studio에서 제공하는 디버깅 기능은 여기서 그치지 않는다. 위와 같이 사용자가 직접 추가적인 Option들을 제공해야 하는 경우들도 있지만, Visual Studio 자체에서도 상당히 강력한 디버깅 옵션들을 제공하고 있다. 그 중 가장 빈번하게 쓰이는 것은 바로 Break Point(중단점) 설정이다. 사용자는 코드의 원하는 위치에 Break Point를 설정할 수 있다. Break Point를 설정하는 방법은 소스코드에서 중단점 삽입을 원하는 Line에서 F9 키를 누르는 방법이 있고, 아니면 소스코드 맨 왼쪽을 클릭하는 방법이 있다.

Break Point의 기능은 다음과 같다. Break Point가 삽입되면 소스코드가 실행되다가 Break Point 삽입 전 Line까지 실행하고 일시 중지된다. 디버거가 Break Point에서 일시 중지가 되면, <Figure 2>와 같이 이제 실행해야 할 소스코드의 Line(Break Point를 삽입한 Line)에 노란색 화살표가 표시된다. 또 Break Point 이전까지의 변수들의 이름들, 그 변수들에 할당되어 있는 Value들, Value들의 Type들까지 보여준다. 그리고 이 Break Point에서 계속 멈춰 있는 것이 아니라 다양한 디버깅 메뉴들을 통해 소스코드를 Line 단위로 실행할 수 있다. <Figure 3>는 디버깅 메뉴이다. 여기서 주목할 기능은, <Figure 3>의 4, 5번째 버튼이다. 4번째 버튼은 ‘한 단계씩 코드 실행’이다. 즉, 짜여진 논리의 한 Step by Step으로 실행을 하는 것이다. 만약 소스코드 중에 함수를 호출하고 그 리턴 값을 사용하는 Line이 있다고 가정했을 때, ‘한 단계씩 코드 실행’을 하게 되면, 호출한 함수의 소스코드로 화살표가 넘어가서 디버깅을 한다. 하지만 5번째 버튼은 ‘프로시져 단위로 실행’이다. 위의 예시에서 ‘프로시져 단위로 실행’을 하게 되면, 호출한 함수로 넘어가는 것이 아니라, 함수를 디버거가 실행시키고, 리턴 값을 바로 받아오게 된다. 그리고 다음 줄로 화살표가 바로 넘어가게 된다. 이렇게 한 줄 한 줄 디버깅이 가능하게 되면서 Visual Studio는 엄청난 편리성을 사용자들에게 제공한다.

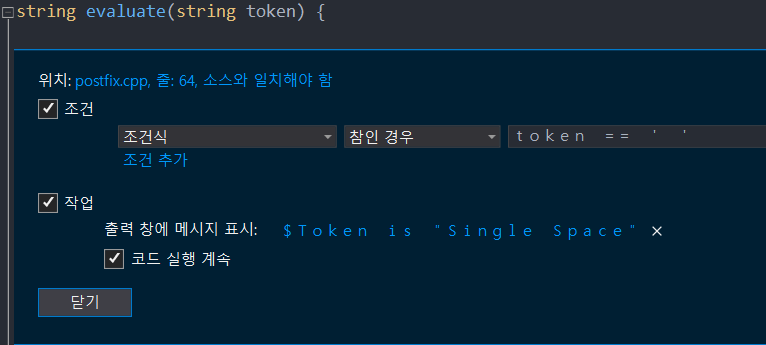


<Figure 2>



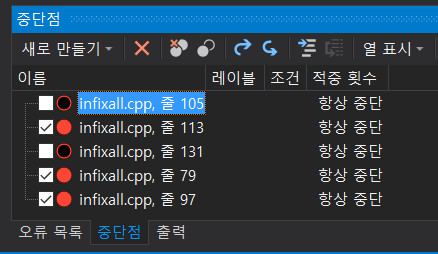
<Figure 3>

Break Point를 설정 할 때, <Figure 4> 와 같이 ‘조건’과 ‘작업’을 설정해 줄 수도 있다. ‘조건’에는 Break Point를 활성화 할 조건을 설정할 수 있다. 그 조건으로는 적중횟수, 조건식, 필터가 들어갈 수 있다. <Figure 4>에서는 조건식을 사용하여, token의 값이 ‘ ‘(Single Space)일 때, Break Point가 동작하게 설정하였다. 또한 특정한 작업을 할 수 있게 설정을 할 수도 있다. 출력창에 사용자가 원하는 메시지를 표시하게 하는 기능은 실제로 매우 유용하게 쓰였다. <Figure 4>에는 출력 창에 “$Token is “Single Space””라는 메시지를 출력하게 하는 작업을 수행하도록 하였다. 이후에도 코드 실행은 계속 된다.



<Figure 4>

또한 여러가지 Break Point들을 사용할 때, 만들어 놓은 Break Point들을 한 번에 관리하기 쉽게 모아 놓은 창이 있다. <Ctrl> + <Alt> + <B> 키를 통해 활성화 할 수 있다. <Figure 5>을 참조하면, 빨간 색 원으로 색이 칠해져 있는 Break Point들이 현재 ‘활성화 되어 있는’ Break Point들이고, 테두리만 빨간색이고 검은색으로 칠해져 있는 Break Point들이 ‘비활성화 되어 있는’ Break Point들이다.



<Figure 5>

Visual Studio는 앞서 알아본 것과 같이 많은 편리함을 개발자에게 제공한다. 이런 이유들이 Visual Studio를 가장 강력하고 많이 사용하는 IDE로 꼽는 이유이기도 할 것이다. 이 외에도 사실 보고서에서 다루지 못한 다양한 디버깅 기능들이 있다. Line을 수행하는 데 걸리는 시간, Memory 사용 등등이 있다. 이러한 기능들을 잘 활용해서 툴을 최대한 효과적으로 사용해야 할 것이다. 이로써 Visual Studio의 Debugging 에 관한 분석을 마치고자 한다.