# IDL 프로그램 실행하기

## 이 문서에 대하여

뜬금없이, 여태껏 잘 써오던 IDL 프로그램을 실행하는 방법을 설명하겠다는 것이 이상하게 들릴 수 있지만, 거의 대부분 그동안 IDL 환경을 실행하고, 그 안에서 IDL 프로그램을 실행시키는 방법만 사용해 왔을 것입니다. 이 문서에서는 운영체제의 프롬프트(IDL 프롬프트가 아닌)에서(또는 다른 프로그래밍 언어에서) IDL 프로그램을 바로 실행시키는 방법을 소개하고자 합니다.

## 실습 프로그램

"두 지점의 경도와 위도를 주면 두 지점의 거리를 출력하는 프로그램"을 목적으로 한다면(실제 작업에서는 더 복잡한 일들을 해야겠지만, 이 문서의 주제에 초점을 맞추기 위해 간단한 문제로) 보통의 경우, IDL 환경으로 들어가서는 다음과 같이 간단히 끝납니다.

신대방동 기상청 : 37.49N, 126.92E
제주 고산 기상대 : 33.29N, 126.16E

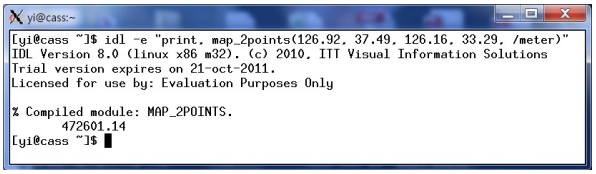
```
IDL> print, map_2points(126.92, 37.49, $
          126.16, 33.29, /meters)
       472601.14
```

map_2points() 함수는 이보다 좀 더 다양한 기능을 가지고 있습니다만, 두 지점의 거리를 미터(m) 단위로 출력하는 방법은 위와 같습니다.

## 셸(DOS창/터미널)에서의 실행 ( -e 옵션 )

앞의 사용 예는 평소처럼, IDL 프롬프트에서 실행을 한 것이고, 유닉스 터미널의 셸 프롬프트나 DOS 창의 프롬프트에서 바로 실행을 하려면 다음과 같이 -e 옵션을 활용합니다.

```
$ idl -e "print, map_2points(126.92, 37.49, 126.16, 33.29, /meters)"
```




idl 실행 옵션 중 -e 옵션은 그림에서 보는 바와 같이 이후에 등장하는 명령어 한 줄을 실행합니다. 따옴표로 감싼 것은 운영체제(리눅스 등)에서 일반적으로 띄어쓰기가 나타날 경우 분리된 argument로 인식하기 때문입니다. 실행문 하나를 통째로 넘겨줘야 하므로 따옴표로 명령문의 시작과 끝을 둘러싼 것입니다.

-e 옵션만으로도 상당히 많은 일을 할 수 있습니다. 실행 방법도 매우 간단하여, IDL 안에서 실행하는 것과 완전히 동일하기 때문에, 더 배워야 할 것도 없습니다.

## -e 옵션의 제약

-e 옵션 실행 방식은 실행하는 순간 마다 컴파일을 수행합니다. 실행 시에 compiled module : Map_2POINTS 라는 메시지를 보셨을 것입니다. 아무래도 실행 속도에 손해가 있겠지요. 더 큰 문제는, 이 방식의 실행을 하려면 IDL Full 라이선스(개발자용 라이선스)가 필요하다는 것입니다. IDL Full 라이선스를 개발된 프로그램의 실행만을 위해 소모하는 것은 큰 손해입니다.

다음 방식은 Runtime 라이선스라는, 개발된 프로그램을 실행만 할 수 있는 라이선스(또는 무료의 VM 라이선스)로 IDL 프로그램을 실행하는 방법입니다.

## Step 1. 배포를 위해 프로시저의 형태로

실행을 위해 배포하는 프로그램은 이미 컴파일이 다 되어 있어, 운영체제에서 실행 명령만 내리면(윈도우즈의 경우 더블클릭을 하면) 프로그램이 시작되는 상태가 되어야 합니다. 이 때, 실행 프로그램(실행 시 호출되는 일종의 메인프로그램)은 반드시 프로시저 형태로 만들어져야 합니다. 1차적으로 프로시저 형태로 만든다면 다음과 같이 만들 수 있겠지요.

```
pro map_2points_run, x1, y1, x2, y2
  d=map_2points(x1, y1, x2, y2, /meters)
  print, d
end
```

별로 손 댄 것 없지요? 그저 프로시저의 형태로 만든 것입니다. 소스코드를 저장하고 실행해 보세요. 소스코드 저장은 map_2points_run.pro (프로그램 구동시 가장 먼저 실행될 프로시저의 이름과 같도록) 로 해 주어야 합니다.

```
IDL> map_2points_run, 126.92, 37.49, 126.16, 33.29
       472601.14
```

## Step 2. 더 이상 IDL 명령문이 아닙니다.

위 프로그램은 IDL 개발 환경에서 멋지게 작동하지만, Runtime이나 VM 환경에서 정상 작동하지 못합니다. 셸이나 DOS 운영체제 상에서는 IDL 명령 실행 방식으로 인수(Argument)를 전달하지 못하기 때문인데요, 이후에 COMMAND_LINE_ARGS() 함수를 이용하여 이를 해결합니다. 이는 다음 단계에서 확인하도록 하고, 일단은 임시로 디폴트 값을 이용해 실행 프로그램을 만들어 보겠습니다.

```
pro map_2points_run
  x1=126.92 & y1=37.49
  x2=126.16 & y2=33.29
  d=map_2points(x1, y1, x2, y2, /meters)
  print, d
end
```

## Step 3. 컴파일 및 실행 프로그램 저장.

1. **프로그램 소스코드를 저장**합니다. 이 때 파일 이름은

map_2points_run.pro (프로그램 구동시 가장 먼저 실행될 프로시저의 이름과 같도록)으로 해 주어야 합니다.

2. 메모리에 있는 모든 컴파일된 루틴들을 저장하여 실행판을 만들 계획입니다. 그래서 일단 **메모리를 깨끗이 합니다.** 필요 없는 프로그램들의 잔재가 함께 저장되어도 실행은 문제없지만 바람직하지는 않습니다.
   IDL> *.reset_session*
   워크벤치 상단의 "세션 초기화"버튼으로도 같은 기능을 수행할 수 있습니다.
3. **컴파일**을 수행 합니다. (단축키 Ctrl+F8)
   IDL> *.compile* map_2points_run
   % Compiled module: MAP_2POINTS_RUN.
4. 관련 있는 소스 코드 중 아직 컴파일 되지 않는 것들을 **모두 수색하여 컴파일** 해 놓습니다.
   IDL> resolve_all
   % Compiled module: RESOLVE_ALL.
   % Compiled module: MAP_2POINTS.
   이 단계는 평소 보지 못했던 분들 꽤 계실 겁니다. IDL은 프로그램을 실행하는 중에 "이 함수 아직 컴파일 되지 않았네"라고 판단되면 그 때 바로 컴파일을 합니다. 그래서 미리 컴파일 해 놓지 않아도 문제없이 실행이 되었던 것이지요. 그런데, Runtime이나 Virtual Machine 모드에서는 컴파일을 할 수가 없습니다. 그래서 미리 샅샅이 컴파일을 해 놓아야 하는 것인데, 별로 어려울 것은 없습니다. Resolve_All 이 모든 것을 다 해 줍니다.
5. 컴파일된 내용을 **저장**합니다.
   IDL> save, filename='map_2points_run.sav', /routines
   파일이름은 역시 처음 시작할 프로시저의 이름과 같도록, 대신 확장자는 .sav로 하는 것이 중요하고, 뒤의 /routines 키워드를 꼭 붙여야 "컴파일된 프로그램 저장"의 의미를 가집니다.

이렇게 저장된 .sav 파일은 **운영체제에 무관**하게 실행됩니다. 즉, 이 sav 파일은 어느 운영체제에서 생성되었든 상관 없이 어떤 운영체제에서도 실행이 가능합니다.(JAVA와 원리가 같습니다).

## Step 4. 실행

IDL로 들어가지 않고 유닉스의 셸 또는 DOS에서 컴파일 된 프로그램을 실행시키는 방법은 다음과 같습니다.

```
$ idl -rt='map_2points_run.sav'
```

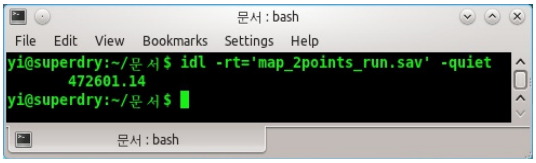
계산 결과가 나올 것입니다. 만일 IDL 버전 정보, 라이선스 정보 같은 것들이 보이는 것이 성가시다면 -quiet 옵션을 사용해 보십시오.

```
$ idl -rt='map_2points_run.sav' -quiet
```

라이선스가 없는 경우(무료 라이선스)로 실행할 경우 Virtual Machine을 다음과 같이 실행합니다.

```
$ idl -vm='map_2points_run.sav' -quiet
```

-vm 옵션은 IDL 광고창이 뜨고 이를 한번 클릭해야 합니다.. 그 밖에는 -rt 옵션과 사용법이 같습니다.




## Step 5. 인수 전달 장치

매번 똑같은 x1, x2, y1, y2 값을 사용하실 계획은 아니겠지요? IDL 프로그램 내에서 **COMMAND_LINE_ARGS()** 함수를 이용하면 운영체제로부터 실행될 때의 인수를 받아들입니다. 프로그램이 다음과 같이 수정됩니다.

```
pro map_2points_run
  args=command_line_args(count=ct)
  if ct ne 4 then begin
    x1=126.92 & y1=37.49
    x2=126.16 & y2=33.29
  endif else begin
    x1=float(args[0]) & y1=float(args[1])
    x2=float(args[2]) & y2=float(args[3])
  endelse
  d=map_2points(x1, y1, x2, y2, /meters)
  print, d
end
```

command_line_args() 함수는 IDL이 -rt나 -vm 모드로 실행될 때 -args(또는 -arg) 옵션 뒤에 나오는 값들을 문자열로 받아들입니다(그래서 FLOAT() 함수로 실수형 변환을 했습니다). 이 때 각각의 인수들은 띄어쓰기로 구분됩니다(DOS나 Unix의 Shell이 원래 그렇습니다). COUNT 키워드는 이들 인수의 개수를 세어 주고, 예제 코드에서 보듯이 인수의 개수에 따라 다른 반응을 하도록 만들 수 있습니다. Step 3을 다시 거쳐 새로 수정된 실행판을 만들어 보세요.

실행은 다음과 같이 합니다.

```
$ idl -rt='map_2points_run.sav' -args 126.92 37.49 126.16 33.29
```

이제 입력 값은 자유롭게 사용할 수 있습니다. 예를 들어 보현산 천문대와 소백산 천문대의 거리를 계산하기 위해서는 다음과 같이 사용하면 됩니다.

```
$ idl -rt='map_2points_run.sav' -args 128.98 36.16 128.46 36.93
    97519.188
```

## 활용 방안

이러한 방식을 사용하여 셸 스크립트나 다른 프로그래밍 언어 중간에 IDL 프로그램을 실행시킨다든지, CRON 스케줄러를 이용하여 주기적으로 IDL 프로그램을 실행시키는 것이 충분히 가능할 것이라고 생각합니다. 참고로 idl 실행 옵션인 -rt, -args, -quiet 등은 순서에 상관없이 사용하시면 됩니다.