CMake Aracının Kullanımı

CMake son yıllarda gittikçe daha poüler olmuştur. Genel kullanımı Make aracına göre daha kolaydır. Ancak giriş kısmında da belirtildiği gibi CMake aslında platforma bağlı olarak o platformda kullanılan build kaynak dosyalarını üretmektedir. Yani örneğin tipik olarak biz build işlemini CMake ile organize ederiz. CMake bize bir make dosyası üretir. Biz de onu make işlemine sokarız.

CMake hem Linux hem Windows hem de Mac OS X sistemlerinde kullanılabilen cross platform bir araçtır. Kurulumu oldukça kolaydır. Bunun için cmake.org sistesinde "download" kısmına gelinir. İlgili platformdaki kurulum dosyası indirilir. Örneğin Windows için ".msi" uzantılı "install dosyası" vardır. Linux sistemlerinde doğrudan ilgili dağıtımın sunduğu paket yöneticilerinden faydalanılabilir. Örneğin apt-get programı ile kurulum aşağıdaki gibi yapılabilmektedir:

```
sudo apt-get install cmake
```

Fakat zaten Linux sistemlerinde pek çok dağıtımda "cmake" temel bir araç olarak zaten kurulmuş durumda olmaktadır. Tabii biz onu güncellemek isteyebiliriz. Mac OS X sistemlerinde de kurulum dosyası cmake.org sitesinden indirilebilir. Her ne kadar cmake aslında komut satırından çalıştırılan bir araçsa da bir GUI ortamı da ayrıca bulunmaktadır. Windows kurulumunda hem komut satırı aracı hem de GUI aracı birlikte kurulmaktadır.

CMake'in de ayrı bir dili vardır. Bu dil komutlardan (commands) oluşmaktadır. Komutlar arasında istenildiği kadar boşluk karakterleri bırakılabilir. Komutların genel sentaks biçimi şöyledir:

```
<komut ismi> (argüman listesi)
```

Argüman listesi virgül atomlarıyla değil boşluk karakterleriyle birbirlerinden ayrılmaktadır. CMake komutlarında büyük harf-küçük harf duyarlılığı yoktur. Fakat küçük harf ağırlıklı bir yazım tercih edilmektedir.

Geleneksel olarak CMake dosyası "CMakeLists.txt" ismiyle bulundurulmaktadır. Bu isim cmake programı tarafından dizin içerisinde aranmaktadır. Tabii biz aslında CMake dosyasına istediğimiz bir ismi de verebiliriz.

En önemli komutlardan biri add_executable isimli komuttu. Bu komut istenildiği kadar çok argüman alabilir. İlk argüman hedef dosyanın ismidir. Sonraki argümanlar projeye dahil olan kaynak dosyaları belirtir. Örneğin:

```
add_executable(a a.c b.c)
```

Burada oluşturulacak hedef "a" isimli çalıştırılabilir dosyadır. Bunun için "a.c" ve "b.c" dosyaları derlenerek bağlanacaktır. Komuttaki argümanların başluk karakterleriyle birbirlerinden ayrıldığına dikkat ediniz.

project isimli komut projenin ismini belirtmektedir. Her projeye bir isim vermek zorunlu olmasa da tavsiye edilmektedir. Örneğin:

```
project(a)
```

Bu durumda minimal bir CMake dosyası aşağıdaki gibi oluşturulabilir:

```
project(a)
```

```
add_executable(a a.c b.c)
```

Bu CMake dosyası nasıl işleme sokulur? Bunun için komut satırında CMake dosyasının ismi verlerek cmake programı çalıştırılabilir. Örneğin:

```
cmake CMakeLists.txt
```

Ya da dosya ismi yerine bir dizin ismi verilirse cmake o dizinde CMakeLists.txt dosyasını arar. Bulursa zaten onu işleme sokar. Örneğin proje dizininde olduğumuzu varsayalım:

```
cmake .
```

CMake bize default durumda hangi platformdaysak o platforma ilişkin default build dosyası üretmektedir. Örneğin Linux ortamında cmake bize MakeFile isimli GNU make dosyası üretecektir. O halde bizim cmake işleminden sonra make işlemi yaparak build işlemini yapmamız gerekir. Tabii geliştirme sırasında projeye yeni bir kaynak dosya eklemediysek yeniden cmake işlemini yapmamıza gerek yoktur. Yalnızca make işlemi yapabiliriz.

CMake dosyalarının başında genellikle bir cmake_minimum_required komutu bulunur. CMake aracı zamanla farklı özelliklere sahip olduğu için CMake dosyasının eski bir versiyon ile işleme sokulmasını engellemek amacıyla bu komutun bulundurulması tavsiye edilmektedir. Komutun genel biçimi şöyledir:

```
cmake_minimum_required(VERSION x.x.x)
```

Örneğin:

```
cmake_minimum_required(VERSION 2.8.9)
project(a)
add_executable(a a.c b.c)
```

Projelerde genellikle include dosyaları ayrı dizinlerde tutulmaktadır. İşte derleyicinin o include dizinlerine bakmasının sağlanması için include_directories komutu bulundurulmuştur. Komutun yalın genel biçimi şöyledir:

```
include_directories(dizin1 dizin2 dizin3....)
```

Dizinlerin yol ifadeleri göreli ya da mutlak verilebilir. Örneğin:

```
cmake_minimum_required(VERSION 2.8.9)
project(a)
include_directories(inc)
add_executable(a a.c b.c)
```

cmake programı pek çok doğal build aracı için kod üretebilmektedir. Eğer komut satırında -G seçeneği ile bu belirleme yapılmamışsa default durum ele alınır. Default durum PATH çevre değişkeninde bulunan derleme arfaçlarıyla değişebilmektedir. Windows'ta default olarak Microsoft'un "msbuild" aracı için ".sln" uzantılı "solution" ve buna bağlı proje dosyaları üretilmektedir. Linux sisteemlerinde default durum "GNU Make" dosyasının üretilmesidir. Tabii biz -G seçeneği ile bu default durumu değiştirebiliriz. Örneğin Windows'ta gcc'nin MinGW port'u ile mingw32-make aracı için kod üretilmesini -G "MinGW Makefiles" seçeneği ile sağlayabiliriz. Örneğin:

```
cmake -G "MinGW Makefiles" .
```

Diğer seçenekleri görmek için komut satırında -G seçeneğini vererek yardım alabilirsiniz:

```
cmake -G
```

CMake dosyası içerisinde tıpkı Make aracında olduğu gibi makrolar (değişkenler) kullanılabilir. Bir makroya değer yerleştirmek için set komutu kullanılmaktadır. set komutunun genel biçimi şöyledir:

```
set(<makro ismi> <değer1> <değer2> ...)
```

Örneğin:

```
set(SOURCES a.c b.c c.c d.c)
```

Makroların değerleri ${<makro ismi>} ile elde edilir. Örneğin:

```
cmake_minimum_required(VERSION 2.8.9)
project(a)
include_directories(inc)
set(SOURCES a.c b.c)
add_executable(a ${SOURCES})
```

CMake dosyasında '#' karakteri yorumlama anlamına gelmektedir. Bu karakterden satır sonuna kadarki tüm karakterler cmake programı tarafından dikkate alınmamaktadır.

Birden fazla dosya ismini joker karakteri kullanarak bir makroya atamak için file komutu kullanılmaktadır. Eğer file komutu bu amaçla kullanılacaksa komutun önekinde GLOB belirlemesinin yapılması gerekir. Örneğin:

Örneğin:

```
cmake_minimum_required(VERSION 2.8.9)
project(a)
include_directories(inc)
file(GLOB SOURCES *.c)
add_executable(a ${SOURCES})
```

Burada dizindeki bütün C dosyaları build işlemine dahil edilmiştir.

Kütüphane dosyası yaratmak için add_library komutu kullanılmaktadır. Komutun yalın genel biçimi şöyledir:

```
add_library(<isim> [STATIC | SHARED] <dosya1> <dosya2> ...)
```

Default durum static kütüphane biçimindedir. Bir kütüphane ya da çalıştırılabilen dosyanın link aşamasında bir kütüphaneyi kullanabilmesi için target_link_libraries komutunun uygulanması gerekir. Örneğin:

```
cmake_minimum_required(VERSION 2.8.9)
project(a)
include_directories(inc)
add_library(b STATIC b.c)
add_executable(a a.c)
target_link_libraries(a b)
```

Burada “a” isimli çalıştırılabilen dosya elde edilmek istenmiştir. Ancak “b” isimli bir static kütüphane yapılmış ve a’nın o static kütüphaneyi kullanması sağlanmıştır. Benzer biçimde dinamik kütüphane de aşağıdaki gibi oluşturulabilir:

```
cmake_minimum_required(VERSION 2.8.9)
project(a)
include_directories(inc)
add_library(b SHARED b.c)
add_executable(a a.c)
target_link_libraries(a b)
```

cmake programının ürettiği build dosyalarında zaten “clean” isimli bir hedef vardır. Ancak install hedefi install komutuyla oluşturulmaktadır. install komutunun genel biçimi şöyledir:

```
install(<tür> <isim> DESTINATION <yer>)
```

Tür olarak TARGETS kullanılırsa isim de bir hedef ismi olur. Bu durumda ilgili hedef ile üretilen dosya DESTINATION ile belirtilen dizine kopyalnır. Örneğin:

```
cmake_minimum_required(VERSION 2.8.9)
project(a)
include_directories(inc)
add_library(b SHARED b.c)
add_executable(a a.c)
target_link_libraries(a b)
set(CMAKE_INSTALL_PREFIX .)
install(TARGETS b DESTINATION bin)
install(TARGETS a DESTINATION bin)
```

DESTINATION ile belirtilen dizin göreli ise SMake onun kök dizinini CMAKE_INSTALL_PREFIX isimli makronun belirttiği yerde aramaktadır. Bunun da platforma bağlı olarak default bir değeri vardır. Ancak yukarıdaki örnekte biz set komutuyla bu yeri değiştirdik. Şimdi artık komut satırında “make install” ya da “mingw32-make install” yaptığımız zaman kütüphane dosyaları ile çalıştırılabilen dosyalar bin dizinine kopyalanacaktır.

CMake dilinde tıpkı GNU Make dilinde olduğu gibi IF gibi deyimler de vardır. if deyiminin genel biçimi şöyledir:

```
if ([Tür] <ifade>)
...
else
....
endif
```

Buradaki tür EXISTS gibi, IS_DIRECTORY gibi bazı operatörlerden oluşturulabilmektedir. Bunların listesi için cmake dokümanlarına bakılabilir. Örneğin EXISTS bir özelliğin var olup olmadığını sorgulamakta kullanılır. Örneğin:

```
if (MINGW)
        ....
endif
```

Burada eğer MinGW build sistemi kullanılıyorsa ilgili komutun dahil edilmesi sağlanmaktadır. MINGW gibi pek çok önceden tanımlanmış makro vardır. Bu makrolar hangi sistem söz konusuysa ona göre true ya da false değeri verirler. Örneğin:

```
if (WIN32)
        ...
endif
```

if komutunun dışında döngü komutları da vardır.

Burada biz temel bir CMake kullanımını ele aldık. Halbuki bu dilin pek çok ayrıntısı da vardır. Bunun CMake'in orijinal dokümanlarından ya da kurstaki EBook dizininde bulunan "Mastering CMake" kitabından faydalanılabilir. CMake'in orijinal kaynakları çeşitli kategoriler altında dokümante edilmiştir.