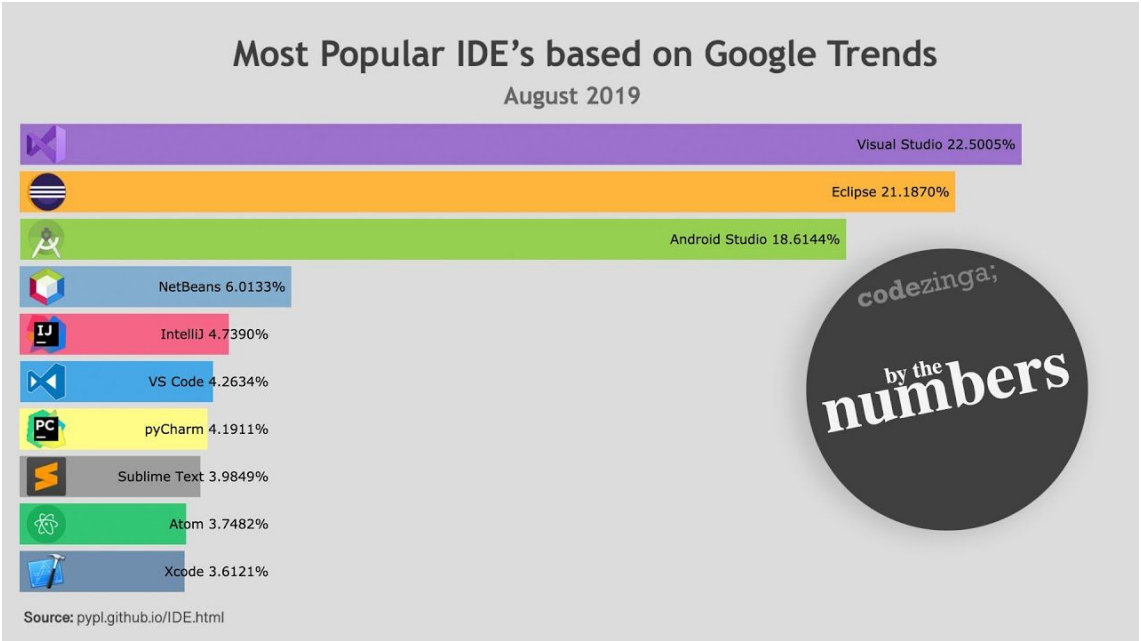
CSCI 274 - Intro to Linux OS

Week 16 - Extra: Using Compilers

Max Gawason (maxgawason@mines.edu) [A]
Rocco Marchitto (rmarchitto@mines.edu) [B/C/D]

Clicking Run

When you use an IDE (Integrated Development Environment) , ‘run’ attempts to compile and link the high level language code. If that is successful, then it will run in a command window based on your operating system such as .exe file for Windows and executable files for Linux and Unix systems. Basically, your IDE makes it easier for people to run code without worrying about the compiler and linking language specific libraries.



Compilers

A compilers is a software which converts a program written in a high level language (i.e. C,C++,Java, etc.) to low level language (i.e. Machine Language). They are language specific, which is why so many exist.

Popular Compilers:

Additional List of compilers: https://www.codechef.com/wiki/list-compilers

GCC

The GNU Compiler Collections which is used to compile mainly C and C++. Besides from just compiling code, gcc has options for linking libraries, creating object files, debugging, displaying warnings, etc.

Simple command:

- gcc [source name]
- execute output file: ./a.out

By default, a.out of the output file name given by gcc compiler

```
mona@LAPTOP-38UGV8T2:~/week17$ cat main.cpp
// very simple hello world

#include <stdio.h>
using namespace std;

int main() {

    printf("Hello World \n");
    return 0;
}
mona@LAPTOP-38UGV8T2:~/week17$ gcc main.cpp
mona@LAPTOP-38UGV8T2:~/week17$ ls -l
total 16
-rwxrwxrwx 1 mona mona 8304 May  1 18:18 a.out
-rw-rw-rw- 1 mona mona  134 May  1 18:17 main.cpp
mona@LAPTOP-38UGV8T2:~/week17$ ./a.out
Hello World
mona@LAPTOP-38UGV8T2:~/week17$
```

GCC - useful options

- To compile without the given default name, use gcc option **-o**
  - gcc [source file] **-o** [new object name]
- To checkout for errors and all kinds of warning, compile with the option **-Wall**
  - gcc **-Wall** [source file]
- To use the **c11** version of standards for compiling
  - gcc **-std=c11** [source file]
- There is a **verbose** option available
  - gcc **-v** [source file]

These are not all the options that are available. You can also compile options to do more than one action or objective.

Ex. gcc -Wall -v -std=11 [source file] -o [new object name]

Makefile

To compile large C/C++ code baselines, Makefiles are used. They are a simple way to organize code compilation. Basically, it is a script with targets that **GNU Make** software reads to compile C and C++ code. The following is the generic target entry form:

```
# comment
# (note: the <tab> in the command line is
# necessary for make to work)
target:  dependency1 dependency2 ...
    <tab> command
```

File must be named: Makefile

To run GNU Make: $ make

Makefile

```
1 # build an executable named hello from main.cpp
2
3 all: main.cpp
4         gcc -Wall -v main.cpp -o hello
```

Terminal

```
mona@LAPTOP-38UGV8T2:~/week17$ cat Makefile
# build an executable named hello from main.cpp

all: main.cpp
        gcc -Wall main.cpp -o hello
mona@LAPTOP-38UGV8T2:~/week17$ make
gcc -Wall main.cpp -o hello
mona@LAPTOP-38UGV8T2:~/week17$ ls -1
Makefile
a.out
hello
main.cpp
mona@LAPTOP-38UGV8T2:~/week17$ ./hello
Hello World
mona@LAPTOP-38UGV8T2:~/week17$
```

Notes

- Makefiles can compile several source and header files at one target
- Make can link libraries with object files
- Same principles from scripting applies to a Makefile
- You can have several targets
- **g++** is the GCC C++ compiler

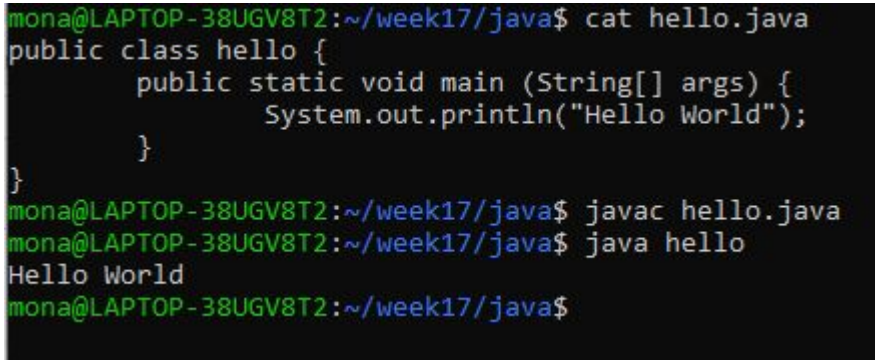
```
#
# Makefile
# Computer Networking Programing Assignments
#

CXX = g++
LD = g++
CXXFLAGS = -g -pthread -std=c++11
LDFLAGS = -g -pthread

#
# Any libraries we might need.
#
LIBRARYS = -lpthread

dug: dug.o dug_help.o log.o
	${LD} ${LDFLAGS} dug.o dug_help.o log.o -o $@ ${LIBRARYS}

dug.o : dug.cc dug.h
	${CXX} -c ${CXXFLAGS} -o $@ $<

dug_help.o : dug_help.cc dug_help.h
	${CXX} -c ${CXXFLAGS} -o $@ $<

log.o : log.cc log.h
	${CXX} -c ${CXXFLAGS} -o $@ $<

```

Compiling Java (quick overview)

As with C/C++, use a compiler to create executable java files. **javac** is the primary Java compiler included in the Java Development Kit from Oracle Corporation.

Simple command

- javac [java file]
- execute: java [class name]



Maven

Very similar to GNU Make, **Maven** is a build automation tool user primarily for Java projects. Maven uses **POM** (project object model) files to manage a libraries, documentations, and compiling java code. There are XML files that contain tasks or goals (very similar to Makefile targets). It builds jar files for you to execute.

```
mvn clean package
```

```
$ java -cp target/com.vogella.build.maven.java-1.0-
SNAPSHOT.jar \
com.vogella.build.maven.java.App
Hello World!
```

COPY CONSOLE

```
<project xmlns = "http://maven.apache.org/POM/4.0.0" xmlns:xsi = "http://www.w3.org/

2001/XMLSchema-instance" xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.companyname.project-group</groupId>
   <artifactId>project</artifactId>
   <version>1.0</version>
   <build>
      <sourceDirectory>C:\MVN\project\src\main\java</sourceDirectory>
      <scriptSourceDirectory>src/main/scripts</scriptSourceDirectory>

      <testSourceDirectory>C:\MVN\project\src\test\java</testSourceDirectory>
      <outputDirectory>C:\MVN\project\target\classes</outputDirectory>
      <testOutputDirectory>C:\MVN\project\target\test-classes</testOutputDirectory>
      <resources>
         <resource>
            <mergeId>resource-0</mergeId>
            <directory>C:\MVN\project\src\main\resources</directory>
         </resource>
      </resources>
      <testResources>
         <testResource>
            <mergeId>resource-1</mergeId>
            <directory>C:\MVN\project\src\test\resources</directory>
         </testResource>
      </testResources>
      <directory>C:\MVN\project\target</directory>
      <finalName>project-1.0</finalName>
```