IDE Programming

Héctor Menéndez[1]

AIDA Research Group
Computer Science Department
Universidad Autónoma de Madrid

November 14, 2013

[1] based on the original slides of the subject

Index

1 Modularity

2 Make

3 Netbeans

Index

1 Modularity

2 Make

3 Netbeans

Modularity

- The source code of any application must be modular. This means that code must be split and grouped in different modules. This modularity makes working with the code easier, because code is better organized.
- It was developed for Java but can be used for other languages (Ruby, C/C++, Python o PHP).
- It was created by Sun Mycrosystems.
- It is easy to use.
  - Main: All code is included in the main function. In this case there is not any organization and it is difficult to work with this code.
  - Functions: Code that is (or can be) repeated several times must be declared in separated functions.
  - Modules: Those functions that perform similar tasks are written in a separate module with the corresponding header file.

Modularity

- In real software development process, the final application is composed by a lot of lines of code, and a huge number of functions.
- It is not recommended to write all those functions in the main file. For that reason, real applications have several files containing some functions grouped by its functionality.
- Imagine we have two .c files (main.c and math.c), we have in math.c a function called myAddition(..) and we want to use this function in our main.c. The only way to do that in C is:
  - Creating a header file for math.c
  - Including the created header file in our main.c

Modularity: Header Files

- Header files allow the function created in a module to be executed by external programs.
- It can contain:
  - Function declarations:

    ```
    <returnType> functionName
    ```
  - Other header files:

    ```
    #include "myHeader.h"
    ```
  - Comments
  - Basic data type definitions: `typedef`
  - Constants
- It never contains:
  - Variable declarations: `int var1;`
  - Functions definitions: `void myFunction1()  ....`

Modularity: Header Files

- If we have our math.c with its corresponding header file math.h and we want to use this functions, we only have to write in our main.c the following line:

  ```
  #include "math.h"
  ```

- With math.c and math.h we have created a module. Note that modules does not have a main function.
- The advantage of creating modules is that in the future we need to use a function from math.c in any project, we only have to include math.h in our application.

Index

1 Modularity

2 Make

3 Netbeans

Make

- If we have an application composed by several .c files, in order to build the executable file we need to do the following:

  ```
  gcc -o executable main.c file1.c file2.c ...
  ```

- The problem is that if any file changes, this command compiles all source files though there are no changes on them.
- For this reason, we need something to compile only those files that have changed (Make tool)

Make

- Make is a tool to automatically update the different programs that compose a software project.
- This update is done using rules that are written in a text file called makefile.
- It is invoke executing the command make. This program executes the different rules described in the makefile taking into account only those parts that have been modified from the last compilation, links those parts and build the executable file.
- Makefile can be used in any programming language whose tasks can be executed in command line.
- It can be executed with parameters to modify its behaviour.

Make

- In a makefile we can have two types of lines:
  - Comment lines start with # symbol.
  - Rules.
- Rule lines have the following structure:

```
destination : requirements
     commands
```

- Destination is the name of the resulting object. It could be the name of the executable file or the name of an object file (.o). It can be also the name of a task, for example clean.
- Requirements. If "Destination" depends on other files or object files, their names must be listed here.
- Commands are the set of commands to execute within this rule.

Make

- Here is an example of a makefile for a calculator:

```
# makefile, version 1
# use tabs (no espaces) in the command line
calc: main.o stack.o getop.o getch.o
	gcc -o myCalc main.o stack.o getop.o getch.o
main.o: main.c calc.h
	gcc -c main.c
stack.o: stack.c calc.h
	gcc -c stack.c
getop.o: getop.c calc.h
	gcc -c getop.c
getch.o: getch.c
	gcc -c getch.c
clean:
	rm myCalc main.o stack.o getop.o getch.o
```

Make

- We can create variables and use them ($(varName)) within the make file.
- A variable is declared as: VARNAME = value (usually variable names are written in uppercase)
- In order to use a variable in a rule we must write: $(VARNAME)

Make Variables	Description
CC	Program for compiling C programs; default ‘cc’.
RM	Command to remove a file; default ‘rm -f’.
CFLAGS	Extra flags to give to the C compiler

Make

```
# makefile, version 2
# use tabs (no spaces) in the command line
CC = gcc
OBJECTS = stack.o getop.o getch.o
CFLAGS = -g -Wall
calc: main.o $(OBJECTS)
	$(CC) $(CFLAGS) -o myCalc main.o $(OBJECTS)
main.o: main.c calc.h
	$(CC) $(CFLAGS) -c main.c
stack.o: stack.c calc.h
	$(CC) $(CFLAGS) -c stack.c
getop.o: getop.c calc.h
	$(CC) $(CFLAGS) -c getop.c
getch.o: getch.c
	$(CC) $(CFLAGS) -c getch.c
clean:
	$(RM) myCalc main.o $(OBJECTS)
```

Make

- Make also has some predefined rules in such a way that some rules are not needed. For example, given a rule whose destination is output.o, Make understand that the file output.c must be compiled. Using this rules makefile can be simplified:

```
# makefile, version 3
# use tabs (no espaces) in the command line
CC = gcc
OBJECTS = stack.o getop.o getch.o
calc: main.o $(OBJECTS)
	$(CC) -o myCalc main.o $(OBJECTS)
main.o: calc.h
stack.o: calc.h
getop.o: calc.h
getch.o:
clean:
	$(RM) myCalc main.o $(OBJECTS)
```

Make

- Make tool can be executed with different parameters:
  - -n print the different commands but commands are not executed.
  - -f <file> Indicates that <file> must be used as a makefile, though <file> has a different name.
  - -p prints the definitions of macros and implicit rules.
  - -C fold Changes the folder and execute the makefile located in fold

Index

1 Modularity

2 Make

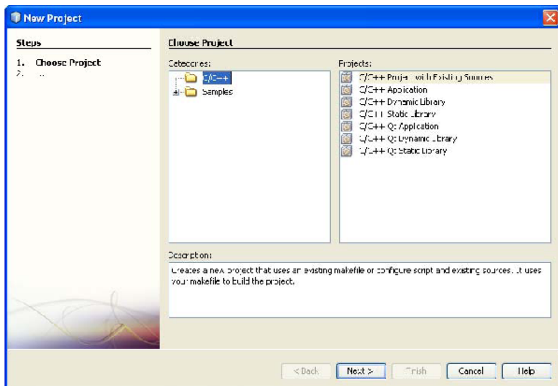
3 Netbeans

Introduction to Netbeans

- NetBeans is an Integrated Develop Environment (IDE), i.e., a developing tool uses to code, compile, debug and run programs.
- It was developed for Java but can be used for other languages (Ruby, C/C++, Python o PHP).
- It was created by Sun Mycrosystems.
- It is easy to use.

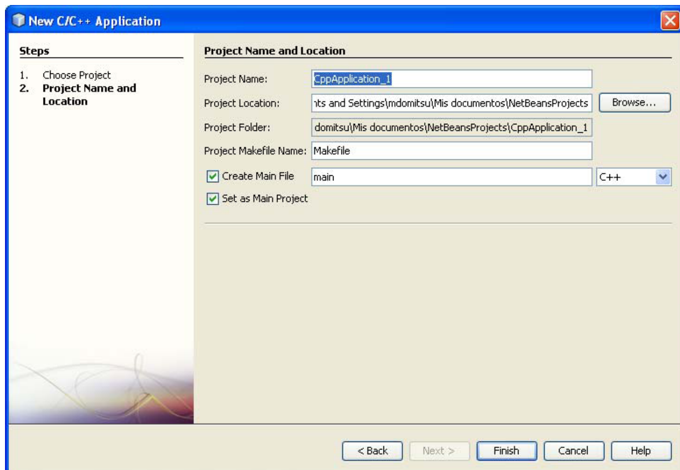
New Project

- File/New Project is used to create a new Project using a Wizard. We will choose C/C++ as Category and C/C++ Application as Project.



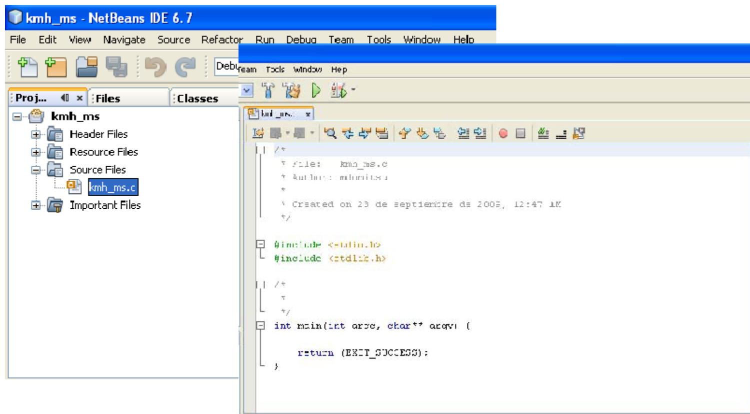
New Project

- After, set the name and project folder.



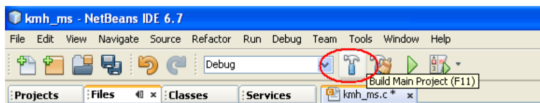
New Project

- The project will be created and the project tree will be shown. Source Files contains the main files.



Compile

- Build Main Project: F11.



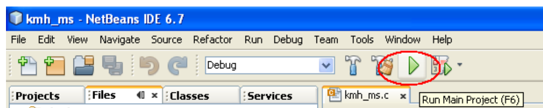
- Build logs are shown below.

Output - kmh_ms (Build) | Tasks

```
make[2]: Entering directory `/c/Documents and Settings/mdomitsu/Mis documentos/NetBeansProjects/kmh_ms'
mkdir -p build/Debug/MinGW-Windows
rm -f build/Debug/MinGW-Windows/kmh_ms.o.d
gcc    -c -g -MMD -MP -MF build/Debug/MinGW-Windows/kmh_ms.o.d -o build/Debug/MinGW-Windows/kmh_ms.o kmh_ms.c
mkdir -p dist/Debug/MinGW-Windows
gcc     -o dist/Debug/MinGW-Windows/kmh_ms build/Debug/MinGW-Windows/kmh_ms.o
make[2]: Leaving directory `/c/Documents and Settings/mdomitsu/Mis documentos/NetBeansProjects/kmh_ms'
make[1]: Leaving directory `/c/Documents and Settings/mdomitsu/Mis documentos/NetBeansProjects/kmh_ms'
BUILD SUCCESSFUL (total time: 4s)
```

Run

- Run Main Project: F6.



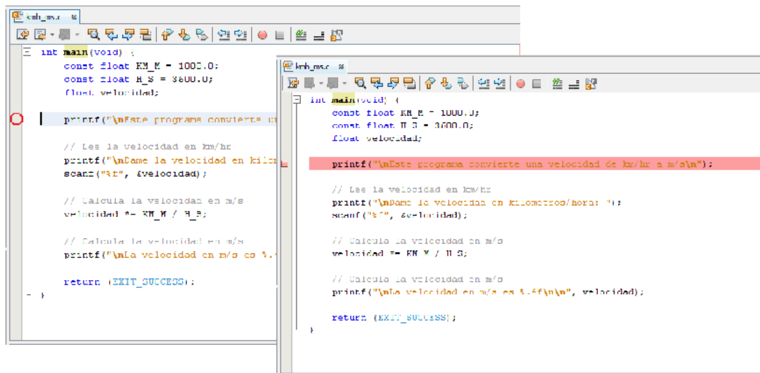
- The application is run in a terminal.

Debugging

- Debugger is able to:
  - Run the program step by step.
  - Create breakpoints.
  - Check and set the variable values.
- To create or remove a breakpoint, press Ctrl+F8.

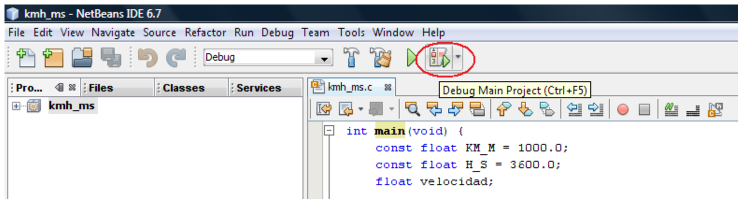
Debugger

- You can also press on the left margin of the chosen line to set a breakpoint.



Debugger

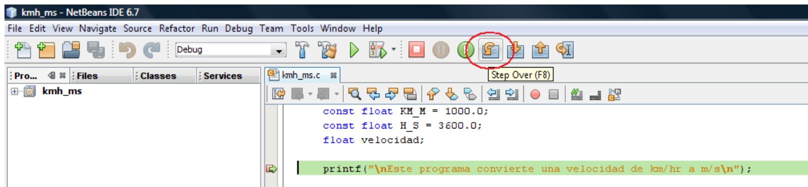
- Run the debugger with Debug Main Project or Ctrl+F5.



- The program will run until the first breakpoint.

Debugger: step by step

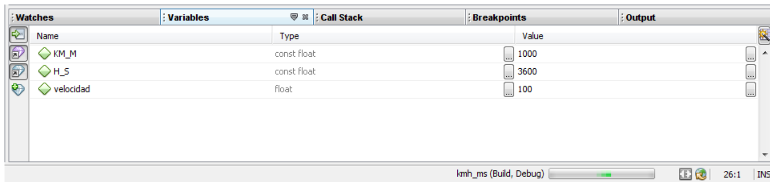
- Step Over: F8.



- Th program is executed step by step.

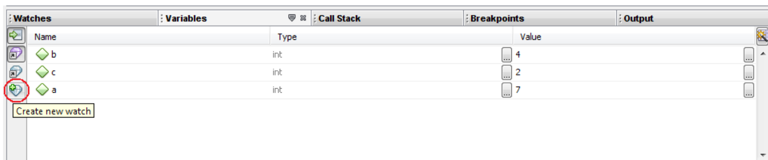
Debugger: variables

- You can manage variables through the variable panel.

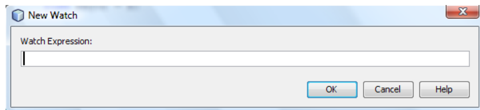


Debugger: expressions

- We can define expressions as variables. Use Watches panel.

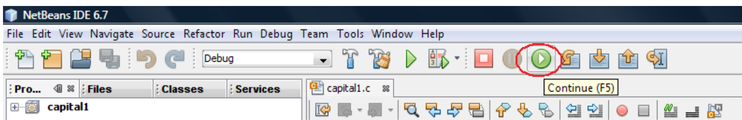


- Add a new expression using New Watch or Ctrl + Shift + F7.

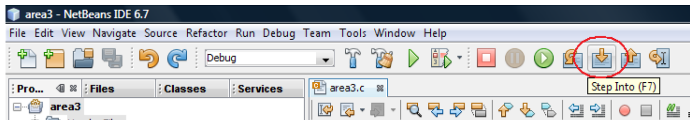


Debugger: other options

- Continue until the Next Breakpoint: F5.



- Step into a function: F7.



Debugger: other options

- Finish: Shift + F5.