Readme for JAVAB Agent

Building the Code:

1: Building the code in Terminal:

Code is provided in a tarball archive. To build the code execute following sequence of codes:

```
$ tar -xvzf nat_agent.tar
$ cd nat_agent
$ make all
```

The output of the code is saved in the output folder.

To clean the code, simple execute:

```
$ make clean
```

2: Building in Eclipse:

- Go to *File>>New>>Makefile Project with Existing Code*.
- Browse to the source and chose appropriate GCC (Linux or cross gcc) for build.
- Right click on the project and select "Build Project".
- The output is "libnat_agent". It is a shared object (.so) file and it'll be saved in output folder.

Using the Agent:

The agent can be used with any java program having a class file or an executable jar file in following way:

```
$ java -agentpath:/path/to/agent -cp /tmp:/your/class/path test.class
```

OR

```
$ java -agentpath:/path/to/agent -cp /tmp -jar test.jar
```

Source Code Walkthrough:

The important source files in the project have been explained in detail below:

- **agent.c:** This is the main agent file which initiates the bytecode analysis and parallelization.
- **class.h:** This header file is included in all agent files. It contains basic macros, definitions and function prototypes. Important Macros are given below:
  - `DEBUG`: This macro turns on or off the debugging prints in the code.
  - `DEBUG_THREADS`: This macro enables the printing of each thread initialization along with its name and owner. This is crucial to ascertain that whether a program is being parallelized or not.

- `COMP_FLAG`: Uncommenting this flag will turn on the JIT level instrumentation and analysis. Keeping it as a comment analyses and parallelizes the classes during class load time.
- `AUTO_QUERY`: This macro enables auto querying function of JAVAB. Whenever JAVAB requires programmer's intervention to ensure dependence safety of arrays, it asks a couple of questions. This macro automatically responds to those query questions as "Yes".

- **main.c:** This file contains the starting point of JAVAB. The function javab_main is the initial entry point for the JAVAB. It strips the class file into internal data structure for analyses and manipulation.

Modifying the MakeFile:

The makefile builds a shared object (.so) library of the agent code. It requires the include path of jvmti.h and jni.h which are provided with the JVM implementation by default. The paths have been shown here in the variables `JVMTI_PATH` and `JVMTI_PATH_LINUX` as absolute paths. These paths have to configured correctly before building the code on any machine. These paths provide `jvmti.h`, `jni.h` and `jni_md.h`.

```
CC = gcc    # C compiler

JVMTI_PATH=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.121-1b14.fc25.x86_64/include
JVMTI_PATH_LINUX=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.121-1.b14.fc25.x86_64/in
clude/linux                          # Path to the JVMTI Libraries

WNO= -Wno-sign-compare -Wno-discarded-qualifiers -Wno-unused-parameter
CFLAGS = -I${JVMTI_PATH} -I${JVMTI_PATH_LINUX} -fPIC -Wall -Wextra -O0 -g3
-fno-omit-frame-pointer
CFLAGS+=${WNO}
LDFLAGS = -L${JVMTI_PATH} -L${JVMTI_PATH_LINUX} -shared

RM = rm -f
TARGET_LIB = libnat_agent

SRCS = src/main.c src/agent.c src/basic.c src/byte.c src/class.c src/dump.c
src/par.c
OBJS = $(SRCS:.c=.o)

.PHONY: all
all: ${TARGET_LIB}

$(TARGET_LIB): $(OBJS)
      $(CC) ${LDFLAGS} -o output/$@ $^

$(SRCS:.c=.d):%.d:%.c
      $(CC) $(CFLAGS) -MM $< >$@

include $(SRCS:.c=.d)
```

```
.PHONY: clean
clean:
	${RM} output/${TARGET_LIB} ${OBJS} $(SRCS:.c=.d)
```