COMP8049
Embedded Systems Engineering
Project – Part b

Completion Date: 22nd December 2020

Value: 25 marks

You may choose one of the two projects below.

Project 1

Pacman application for the QEMU emulated versatilepb board.

1. Briefly explain how the firmware in the pacman-qemu.zip works.

2. Add code to move pacman with the keys w,e,s,d. Pacman should not leave a trail when moving.

3. Add more ghosts to the game. The ghosts should not leave a trail when moving.

4. Write the code for the function black_point in vid.c.

5. Add code to stop pacman and the ghosts from crashing into the walls.

6. Add code to move the other ghosts within the game board. The ghosts should attempt to follow pacman.

See code in Pacman.zip for a linux version. The source is here https://github.com/AXDOOMER/Pacman.

Project 2

The attached java code (ast.zip) is an interpreter for c. The code takes as input a clang generated Abstract Syntax Tree (AST) in a text file (out.txt in the ast2 directory) and traverses the tree while interpreting the nodes. The code only "works" for simple expressions and with integer variables. You are to:-

a) modify the code to handle more complex expressions, "i*f statements*", "*while loops*" and "*for loops*".

b) java classes have an outline gen() method which shows how the code could be used as a compiler to generate ARM assembly. Outline how you would generate code for *if* statements and *while* loops.

We will use clang to generate an AST for a given c simple c function. We use the -ast-dump command line argument to generate a textual representation of the AST, which is stored in a file. For example:

```
clang-check -ast-dump for2.c --extra-arg="-fno-color-diagnostics" > out.txt
```

The attached code reads in this text file and constructs a tree with a java object for each node in the clang AST. For example, the VarDecl class models a c declaration of a variable. When the interpreter sees a VarDecl AST node it creates a corresponding java variable to model it.

If the c variable is used in an expression in t1.c, the AST will have a corresponding DeclRefExpr node to model that use. When the interpreter sees the DeclRefExpr node in the AST it uses the current value of corresponding java variable when evaluating the expression.

For example, the c statement:

```
j = 9;
```

will be modelled using the following AST nodes (dumped using the -ast-dump flag to the compiler):

```
-BinaryOperator 0x30d7968 <line:3:1, col:3> 'int' '='
   | |-DeclRefExpr 0x30d7920 <col:1> 'int' lvalue Var 0x30d7890 'j' 'int'
   | `-IntegerLiteral 0x30d7948 <col:3> 'int' 9
```

The interpreter will “evaluate” each of these nodes and place the value 9 into the java variable j, which was created in the corresponding VarDecl statement.

You may use libclang or libtooling in this project instead of the code in ast.zip if you wish. Libclang is easy to use but only gives minimal access to the AST node context. Libtooling on the other hand is very powerful but is complex to use.