

THE AUSTRALIAN NATIONAL UNIVERSITY

First Semester 2008

COMP2100/2500
(Software Construction, SC for Software Engineers)

Writing Period: 2 hours duration

Study Period: 15 minutes duration

Permitted Materials: one A4 page of notes both sides

Instructions *Answer all five questions.*

The questions do not have equal value, the marks total 100.

Marks for each part are indicated on the exam paper.

Write your answers in the examination book provided.

Start each question on a new page.

Extra exam books are available if you run out of space.

Do not use red ink.

QUESTION 1 [13 marks]

Consider the method `findFirst` in the Java code fragment below.

The type `Ttype` represents an arbitrary type.

```
public static int findFirst (Ttype a[], Ttype t) {  
Precondition:  
  -- see question part (a)  
Postcondition:  
i is a valid index into a and  
i is the index of the first instance of t in a.  
  
int i = 0;  
while (i < a.length && a[i] != t)  
  
  {  
    /*  
    * Invariant: There is no j with 0 <= j < i and a[j] = t  
    * Variant: a.length - i  
    */  
    i++;  
  }  
}
```

Note that this program and its conditions have been changed in small ways compared to the example in the course notes.

- (a) The *postcondition* requires the code to establish two conditions. **Without changing the program code statements**, state the *precondition* that is required for this code to guarantee the postcondition. **[5 marks]**
- (b) Explain with words and diagrams the meaning of the Invariant condition in this case. **[4 marks]**
- (c) Describe how this loop code guarantees to preserve the invariant. **[4 marks]**

QUESTION 2 [25 marks]

The following Java code fragment (headed **fragment A: ExpressionRecursive**) is part of a program that represents Expression trees.

- (a) Write the missing statements in method `main`, that will result in assigning to the variable `e` the representation of the expression

$$(3 \times 1) + (1 + 2)$$

[5 marks]

- (b) The methods named `ratorCount` in each class are designed to count the number of Operators within an expression tree. For example, the expected value for the tree in part(a) is 3 , by counting the instances of Multiplication and Addition operators \times , + and +.

What is the missing statement (or more than one statement) in method `ratorCount` for classes `Constant`, `Multiplication`, and `Addition`? Explain briefly. [10 marks]

- (c) The second program fragment (headed **fragment B: ExpressionWithVisitor**) is also a representation of expressions. It uses the Visitor pattern to traverse the expression tree. The class `ratorCounter` contains the visitor methods that provide an equivalent to `ratorCount`.

What is the missing statement (or more than one statement) in method `visitAddition` in class `ratorCounter`? Explain briefly. [10 marks]

fragment A: ExpressionRecursive

```

1  public class ExpressionRecursive {
2
3      abstract static class Expression
4      {
5          public abstract int evaluate();
6          public abstract int ratorCount();
7      }
8
9      static class Constant extends Expression {
10         public int value;
11         public Constant(int v) {
12             value = v;
13         }
14         public int evaluate(){
15             return value;
16         }
17         public int ratorCount(){
18
19             }
20         } // end class Constant
21
22         static class Addition extends Expression {
23             public Expression left;
24             public Expression right;
25             public Addition(Expression l, Expression r) {
26                 left = l;
27                 right = r;
28             }
29             public int evaluate() {
30                 return left.evaluate() + right.evaluate();

```

```

// MISSING: one or more statements to
// implement ratorCount for class Constant

```

```

31         public int ratorCount() {
32
33             }
34         } // end class Addition -----
35         static class Multiplication extends Expression {
36             public Expression left;
37             public Expression right;
38             public Multiplication(Expression l, Expression r) {
39                 left = l;
40                 right = r;
41             }
42             public int evaluate() {
43                 return left.evaluate() * right.evaluate();
44             }
45             public int ratorCount() {
46
47                 }
48             } // end class Multiplication -----
49         public static void main(String[] args) {
50             Expression a,b,c,d,e, f, g;
51
52             }

```

```

// MISSING: one or more statements to
// implement ratorCount for class Addition

```

```

// MISSING: one or more statements to
// implement ratorCount for class Multiplication

```

```

// MISSING: statements to assign to variable e
a=?   b=?   c=
c=?   d=?   f=?   g=? e=?

```

```

System.out.println("e= "+e.evaluate() +
" no. of operators= " + e.ratorCount());

```

fragment B: ExpressionWithVisitor

```

101 public class ExpressionWithVisitor {
102
103     abstract static class Expression
104     { public abstract void accept(ExpressionVisitor e); }
105
106     interface ExpressionVisitor {
107         public void visitConstant(Constant c);
108         public void visitAddition(Addition c);
109         public void visitMultiplication(Multiplication c);
110     }
111     static class Constant extends Expression {
112         public int value;
113         public Constant(int v) {
114             value = v;
115         }
116         public void accept(ExpressionVisitor e) {
117             e.visitConstant(this);
118         }
119     }
120     static class Addition extends Expression {
121         public Expression left;
122         public Expression right;
123         public Addition(Expression l, Expression r) {
124             left = l;
125             right = r;
126         }
127         public void accept(ExpressionVisitor e) {
128             e.visitAddition(this);
129         }
130     } // end class Addition
131     static class Multiplication extends Expression {
132         // similar to Addition
133         ... } // end class Multiplication
134
135     static class ratorCounter implements ExpressionVisitor {
136         public int count = 0;
137
138         public void visitConstant(Constant c){
139             // do nothing - a constant is not an operator
140         }
141         public void visitAddition(Addition c){
142             c.left.accept(this);
143
144             

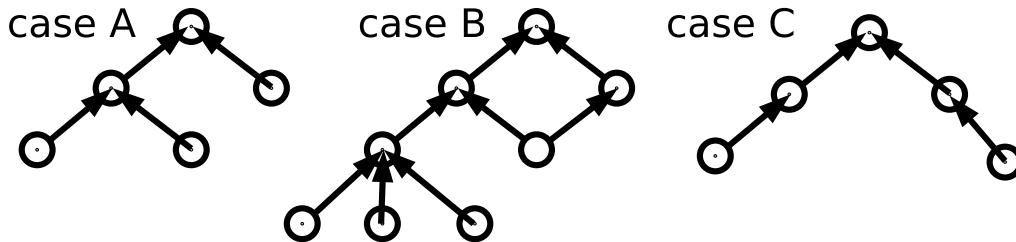
145                 // MISSING: one or more statements to implement
146                 // ratorCounter when visiting an Addition object
147


148         }
149         public void visitMultiplication(Multiplication c){
150             ... // similar to visitAddition
151         }
152     }
153     static class evaluator implements ExpressionVisitor {... }
154
155     public static void main(String[] args) {
156         Expression a,b,c,d,e;
157         // initialisation of these variables: same as other example
158
159         ratorCounter cc = new ratorCounter();
160         e.accept(cc);
161         System.out.println("constant count = "+ cc.count);
162
163         exprEval evaluator = new exprEval();
164         e.accept(evaluator);
165         System.out.println("value = " + evaluator.value);
166     }

```

QUESTION 3 [21 marks]

- (a) List which one or more of the dependency graphs A, B, C below can be expressed as Makefile dependency rules (the arrows in the graph indicate that the node at the point of the arrow depends on the node at the root of the arrow).



[4 marks]

- (b) Draw a dependency graph to represent the following Makefile rules

```
A: B N S
    transform -d5 S B N > A
```

```
Z: X N
    mash X N | modulate > Z
```

```
N: X Y
    intermunge X Y > N
```

[5 marks]

- (c) Assume that the files named X, N, Y, B, and S are all modified at the same time, and following these changes, the user types the command *make A*. State what commands will be executed, showing them in the order they will be executed. If the order is indeterminate (i.e. if the order is not fully determined by these rules), say so. [6 marks]
- (d) Explain why these commands and no others will be executed, and why they will be executed in the order you state. [6 marks]

QUESTION 4 [18 marks]

- (a) A small team of software developers is developing new software by making small changes to a large existing set of programs. Their development method is to make small experimental changes to the features of the software, writing code, testing, and keeping these changes or trying again from their previous version or from some previous version. They expect that many of the changes made by the programmers will be in a small number of program source files.

The team expects to create several slightly different versions of the software for different operating system environments. A possibly different team will make more changes a year later.

Explain how using a build tool such as `make` or `ant` has benefits for this development team (hint: consider what they would have to do without any build tool). [9 marks]

- (b) In a version control system such as SVN a distinction is made between a user's *working copy* of a file and the *repository copy* of the file. Either of these copies can be changed after the working copy is made from the repository, as a result of the user editing the working copy, or by another user committing changes to the repository copy. The user has the `svn` commands `update`, `commit` and `resolved` to control the versions.

We call the original repository copy R_0 . Consider the case: starting from a state where both working copy (now W_1) and repository copy (now R_1) have been changed since the working copy was made

(assume that the changes are made on different lines in the two copies).

Briefly describe each step of what the user sees, and what happens to the working copy and what happens to the repository, **using the terminology of "states" of the files in working copy and in repository**, when the user gives these commands in order

1. `svn status`
2. `svn update`
3. `svn status`
4. `svn commit`

[9 marks]

QUESTION 5 [23 marks]

- (a) Briefly explain what is the difference between **black box testing** and **white box testing**.
[4 marks]

For the remaining parts of this question, consider the task of unit testing the method `printMonthCal` in a newly invented class:

```
/**
 * print a month layout table for one month given length and starting day
 * @param days number of days in this month
 * @param dayOne day number of the first of the month,
 *             based on day number(Sunday)=0, Monday=1, ..., Saturday=6
 */
public static void printMonthCal(int days,
                                int dayOne,
                                String monthLabel,
                                String yearLabel) {
    ...
}
```

For example, `printMonthCal(31, 0, ``March``, ``2015``);` produces

```
      March 2015
-----
 S  M  T  W  T  F  S
-----
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

- (b) Describe the kinds of **test selection strategies** that are best applied to each one of the three arguments to `printMonthCal`
- (i) `days`
 - (ii) `dayOne`
 - (iii) `monthLabel`

[8 marks]

question continued over the page...

(c) Two test cases have been designed:

1. `printMonthCal with days=31 dayOne=0 monthLabel='March' yearLabel='2015'`

Expected result: a month layout table starting 1st of the month on Sunday, and a total of 31 days, with the label March 2015.

2. `printMonthCal with days=39 dayOne=0 monthLabel='March' yearLabel='2015'`

Expected result: an error message: number of days out of range

Design the minimum number of additional test cases that are needed for us to be reasonably confident that the implementation of *printMonthCal* is correct. Explain your reasons for including each test case, and relate them to your test selection strategies.

[11 marks]

5 questions, total marks 100