

THE AUSTRALIAN NATIONAL UNIVERSITY

First Semester 2002

COMP2100 (Software Construction)

Writing Period: 3 hours duration

Study Period: 15 minutes duration

Permitted Materials: None

Answer all questions

*Your answers must be written in the boxes provided in this booklet. You will be provided with scrap paper for working, but only those answers written in this booklet will be marked. Do not remove this booklet from the examination room. There is additional space at the end of the booklet in case the boxes provided are insufficient. Label any answers you write at the end of the booklet to indicate which question they refer to. **Do not write in red ink anywhere in the booklet.***

Name (family name first):

Student Number:

Official use only:

Q1 (20)	Q2 (20)	Q3 (20)	Q4 (20)	Q5 (20)	Total (100)

QUESTION 1 [20 marks]

- (a) Suppose that class *CHILD* inherits from class *PARENT*, and that we have made the declarations *child: CHILD* and *parent: PARENT*. What will be the outcome of the assignment attempt *child ?= parent* if:

- (i) *parent* is Void?

QUESTION 1(a)[i]	[1 mark]

- (ii) *parent* is attached to an object of class *PARENT*?

QUESTION 1(a)[ii]	[1 mark]

- (iii) *parent* is attached to an object of class *CHILD*?

QUESTION 1(a)[iii]	[1 mark]

- (b) Give two reasons for keeping a file under version control using a system like RCS.

QUESTION 1(b)	[2 marks]

- (f) Give two reasons why object-oriented languages are not completely replacing procedural languages.

QUESTION 1(f)	[2 marks]

- (g) Explain the roles of and interactions between the three components of the Model-View-Controller architecture for graphical applications.

- (i) What is the role of the Model? How does it interact with the View and the Controller?

QUESTION 1(g)[i]	[1 mark]

- (ii) What is the role of the View? How does it interact with the Model and the Controller?

QUESTION 1(g)[ii]	[1 mark]

- (iii) What is the role of the Controller? How does it interact with the Model and the View?

QUESTION 1(g)[iii]	[1 mark]

QUESTION 2 [20 marks]

This question is about recursive tree data structures for representing algebraic expressions, similar to those studied in lectures and in Lab 4.

The designer of the system has decided to use a Visitor pattern style implementation of these trees. The starting point for this is the two deferred classes *EXPRESSION* and *VISITOR*.

```
deferred class
```

```
    EXPRESSION
```

```
feature
```

```
    accept (v: VISITOR) is deferred end
```

```
end -- class EXPRESSION
```

```
deferred class
```

```
    VISITOR
```

```
feature
```

```
    visit_constant (x: CONSTANT) is deferred end
```

```
    visit_addition (x: ADDITION) is deferred end
```

```
    visit_multiplication (x: MULTIPLICATION) is deferred end
```

```
end -- class VISITOR
```

Class *EXPRESSION* will have three effective subclasses, representing constants, sums of two expressions and products of two expressions. Only one of these, class *CONSTANT*, has been implemented.

```
class
```

```
    CONSTANT
```

```
inherit
```

```
    EXPRESSION
```

```
creation
```

```
    make
```

```

feature {NONE}

    make (n: INTEGER) is
        -- Initialise
        do
            value := n
        end

feature {ANY}

    value: INTEGER

    accept (v: VISITOR) is
        do
            v.visit_constant (Current)
        end

end -- class CONSTANT

```

(a) Write a new class *ADDITION* which inherits from class *EXPRESSION* and represents sums of two sub-expressions.

- Each object must store its two sub-expressions in attributes of type *EXPRESSION*.
- The creation routine *make* must take two arguments of class *EXPRESSION* and initialise the attributes appropriately.
- The *accept* feature must be made effective.

Your code must conform to the usual Eiffel coding standard. Include require and ensure clauses where appropriate.

QUESTION 2(a)	[6 marks]

QUESTION 3 [20 marks]

In this question you have to do a code review. The code you are to review is an attempted solution to Homework 12. The requirements for the program are:

Write an Eiffel program called 'change' that works out how to make change in coins for a given number of cents.

The program shall take exactly one command line argument, which should be a non-negative integer in the range 3–497. If the argument is out of range, the program should print an error message and stop.

If the argument is not a multiple of 5, the program shall first round it to the nearest multiple of 5.

The different types of coins you may use for making change are 5c, 10c, 20c, 50c, \$1 and \$2.

The program shall use a 'greedy' algorithm to choose between the many possible ways to make change. That is, it shall first use as many \$2 coins as it can, then as many \$1 coins as it can, then as many 50c pieces and so on.

For example:

```
comp2100@iwaki change 20
20c = 1 x 20c
comp2100@iwaki change 85
85c = 1 x 50c + 1 x 20c + 1 x 10c + 1 x 5c
comp2100@iwaki change 90
90c = 1 x 50c + 2 x 20c
comp2100@iwaki change 497
$4.95 = 2 x $2 + 1 x 50c + 2 x 20c + 1 x 5c
comp2100@iwaki change 498
Error: argument 498 is too large.
```

Output should be formatted exactly as in the examples above.

The attempted solution consists of a single class *CHANGE*. The creation procedure is *make*.

```
1 class CHANGE
2
3 feature
4
5     make is
6         -- Makes change for the amount on the command line
7         local
8             amount: INTEGER
9             i: INTEGER
10        do
```

```

11      -- Get and check argument
12      if argument_count /= 1 or else not argument (1).is_integer then
13          std_error.put_string ("Usage: change <amount>%N")
14          die_with_code (exit_failure_code)
15      end
16      amount := argument (1).to_integer
17      if amount < 3 or amount > 497 then
18          std_error.put_string ("Error: argument ")
19          std_error.put_string (argument (1))
20          std_error.put_string (" is too large.%N")
21          die_with_code (exit_failure_code)
22      end
23
24      -- Round to nearest multiple of 5c
25      if amount // 5 /= 0 then
26          inspect amount // 5
27          when 1 then amount := amount - 1
28          when 2 then amount := amount - 2
29          when 3 then amount := amount + 2
30          when 4 then amount := amount + 1
31      end
32
33      -- Print the amount
34      if amount >= 100 then
35          std_output.put_character ('$')
36          std_output.put_double_format (amount // 100, 2)
37      else
38          std_output.put_integer (amount)
39          std_output.put_character ("c")
40      end
41      std_output.put_string (" = ")
42
43      -- Loop through taking off coin values and printing
44      from
45          i := 1
46          n := 0
47      until amount = 0 loop
48          if amount > coins.item (i) then
49              n := n + 1
50              amount := amount - coins.item (i)
51          else
52              std_output.put_integer (n)
53              std_output.put_string (" x ")
54              std_output.put_string (coin_strings.item (i))

```


(c) Your final task is to write a Makefile that automates the process of compilation and testing. The files in your system are `strcnt.c`, `strcnt.o`, `harness.e`, `harness`, `cases`, `tester` and `report.txt`. The last file, `report.txt` should be the output of running the `tester` script.

(Note that the header file `strcnt.h` is not part of the system, since it is not needed when calling the C function from Eiffel.)

(i) Draw a diagram showing the dependencies between the files listed above. Indicate source files by underlining their names.

QUESTION 4(c)[i]	[2 marks]
------------------	-----------

QUESTION 5 [20 marks]

- (a) What benefits might come to you as an individual in the workplace from following the PSP?

QUESTION 5(a)	[2 marks]

- (b) What benefits might come to your organisation?

QUESTION 5(b)	[2 marks]

- (c) Explain why the PSP includes a personal code review *before* the Compile phase, rather than after.

QUESTION 5(c)	[2 marks]

(d) The Process Yield is defined as

$$\text{Yield} = 100 \times \frac{\text{defects removed before Compile}}{\text{defects injected before Compile}}$$

Explain the significance of this number, and why it is defined the way it is. What do high and low values mean? What is a reasonable yield to aim at?

QUESTION 5(d)	[4 marks]

(e) The appraisal to failure ratio is defined as

$$\text{A/FR} = \frac{\text{total time spent looking for defects}}{\text{total time spent fixing defects}}$$

Explain the significance of this number, and why it is defined the way it is. What do high and low values mean? What is a reasonable A/FR to aim at?

QUESTION 5(e)	[4 marks]

(f) Explain why there is more to software quality than defect management.

QUESTION 5(f)	[2 marks]

(g) When recording defects, why is it important to classify the defects according to type?

QUESTION 5(g)	[2 marks]

(h) What is the point of recording and summarising how you spend your time?

QUESTION 5(h)	[2 marks]

