

**THE AUSTRALIAN NATIONAL UNIVERSITY**

*First Semester 2003*

**COMP2100  
Software Construction**

*Writing Period: 3 hours*

*Study Period: 15 minutes*

*Permitted Materials: None*

*Answer all questions*

*Write your answers in the boxes provided in this booklet. Only those answers written in this booklet will be marked. 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 remove this booklet from the examination room. **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 attached to an object of class *PARENT*?

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

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

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

(iii) *parent* is Void?

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]





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

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

(g) Write one line of Bash, as you would type it interactively at the command line, that will count the number of lines in the file `foo.e` that contain at least one character other than space or tab.

You will probably want to use pipes, and some of the commands `tr -d`, `grep` and `wc -l`. The tab character has ASCII code 9, which is 011 octal.

QUESTION 1(g)	[1 mark]

(h) What is a pointer?

QUESTION 1(h)	[1 mark]

(i) Explain the difference between a function *definition* and a function *declaration* in C.

QUESTION 1(i)	[1 mark]

## QUESTION 2 [20 marks]

This question is about recursive tree data structures for representing XML documents, similar to but simpler than those studied in the assignments.

One major simplification is that in this system, XML elements do not have any attributes. Also in this question you do not have to worry about parsing, styles, lookup tables or entities.

The designer of the system hasn't decided whether to use a Visitor pattern style implementation for operations on these trees or a more conventional object-oriented approach (similar to *EXPRESSION* Version 3 from lectures), so has allowed for both.

The starting point is classes *XML\_ELEMENT* and *VISITOR*.

```
deferred class
```

```
    XML_ELEMENT
```

```
feature
```

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

```
end -- class XML_ELEMENT
```

```
deferred class
```

```
    VISITOR
```

```
feature
```

```
    visit_data (x: XML_DATA_ELEMENT) is deferred end
    visit_container (x: XML_CONTAINER_ELEMENT) is deferred end
```

```
end -- class VISITOR
```

In class *XML\_ELEMENT* the feature *to\_string* is to be implemented in the standard object-oriented way, while the presence of *accept* allows for Visitor pattern operations as well.

Class *XML\_ELEMENT* will have two subclasses: class *XML\_DATA\_ELEMENT* (for leaf nodes representing the actual document content) and class *XML\_CONTAINER\_ELEMENT* (for non-leaf nodes that separate the document into parts and give it structure). The first of these has already been written:

```

class
    XML_DATA_ELEMENT

inherit
    XML_ELEMENT

creation
    make

feature

    make (s: STRING) is
        do
            content := clone (s)
        end

    content: STRING

    to_string: STRING is
        do
            Result := content
        end

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

end -- class XML_DATA_ELEMENT

```

The second subclass, *XML\_CONTAINER\_ELEMENT*, is incomplete.

```

class
    XML_CONTAINER_ELEMENT

inherit
    XML_ELEMENT

creation
    make

feature

```

```

make (s: STRING) is
  -- Initialise to empty
  do
    name := clone (s)
    create children.make (1, 0)
  end

name: STRING
children: ARRAY [XML_ELEMENT]

add (x: XML_ELEMENT) is
  -- Add 'x' to the children
  do
    children.add_last (x)
  end

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

visit_children (v: VISITOR) is
  local
    i: INTEGER
  do
    from
      i := children.lower
    until
      i > children.upper
    loop
      children.item (i).accept (v)
      i := i + 1
    end
  end

end -- class XML_CONTAINER_ELEMENT

```

**(a)** Complete this class by writing a query *to\_string* that returns a one-line string representation of the XML element with the following properties:

1. If there are no children, this should look like '<' followed by the value of *name*, followed by '>'.



(b) The *to\_string* feature prints the entire document on one line. It's much easier to read an XML document if it is printed with each tag or data element on a separate line, and with child elements indented.

Write a new effective class *PRETTY\_PRINTER* that inherits from class *VISITOR* and has the following properties:

1. Objects of this class must print XML documents on the standard output.
2. For data elements, they should print the content, on a line by itself, in double quotes.
3. For empty container elements, they should print '<', the value of *name* and '>' (as for *to\_string*) followed by a newline.
4. For non-empty container elements, they should print '<', the value of *name* and '>', then a newline, followed by each of the child elements, indented two additional spaces, followed by '</', the value of *name* and '>', which should be lined up with the start tag.
5. There should be no creation clause.

For example, given the same input as in the previous part, this class should print:

```
<doc>
  <para>
    "Plain "
    <it>
      "italic"
    </it>
    " plain."
  </para>
</doc>
```

QUESTION 2(b)	[7 marks]



- (c) Assume that the variable  $x$  of type *XML\_ELEMENT* is attached to the root element of a tree representing an XML document. Assume also the declaration  $pp$ : *PRETTY\_PRINTER*. Write a fragment of code that will print, on the standard output, *both* versions of the output developed in the code above: first the version implemented in the *to\_string* feature, and then the version implemented in class *PRETTY\_PRINTER*.

QUESTION 2(c)	[2 marks]

- (d) For some purposes it might be useful to be able to count the nodes in a tree, and also its depth. Write a new effective class *NODE\_COUNTER* that inherits from *VISITOR* and has the following properties:

1. The class must have two attributes *count* and *depth* of class *INTEGER*.
2. There shall be no creation clause or creation routine: the only initialisation shall be the automatic setting of *count* and *depth* to their default value (zero).
3. When a *NODE\_COUNTER* traverses a tree (or subtree) representing an XML element, it must add the number of nodes in that tree to the value of *count*. In other words, if a client creates a new *NODE\_COUNTER* and runs it over a tree, the value of *count* at the end shall be the number of nodes in the tree.
4. When a *NODE\_COUNTER* traverses a tree it must set the value of *depth* to the depth of that tree.

(An empty tree has depth zero; a tree containing a single data element has depth one; a container element with a data element as its only child has depth two; and so on. The example document given above has depth four.)

QUESTION 2(d)	[7 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 a slightly simplified version of Homework 11. The requirements for the program were:

Write a program called 'treesort' that uses a binary search tree to sort a list of integers.

The program shall read a list of integers from the user. It shall prompt for each data value by printing the string '>' at the start of a line. It shall read one line of input at a time. If the line contains only a valid integer, it shall insert it in the tree, otherwise it shall ask the user to try again.

To insert a number in a binary search tree: compare it to the root; if it's smaller or equal, insert it in the left subtree; if it's larger, insert it in the right subtree. The insertion into a subtree is a recursive call on the same algorithm. If the tree or subtree is empty, create a new node and put the number there.

When the user has indicated (by entering an empty line of input) that the list is complete, the program shall then print out all the numbers in the list in ascending order.

The program must use a binary search tree for sorting, not any other method. For the purposes of this homework, a binary search tree is defined as a binary tree that stores an integer in each node, and has the following properties:

- Every node in the left subtree has a value less than or equal to the value in the root node.
- Every node in the right subtree has a value strictly greater than the value in the root node.
- Both subtrees are themselves binary search trees.

Here is an example interaction with the finished program:

```
comp2100@iwaki$ treesort
Please enter data values, one per line.
> 3
> two
Invalid input, try again.
> 2
> 4
>
```

In ascending order:

```
2
3
4
```

The attempted solution consists of two classes. The root class is class *TREESORT*.

```
1 class
2
3     TREESORT
4
5 creation
6     make
7
8 feature {ANY}
9
10    line: STRING
11    tree: BINARY_TREE
12
13    get_input: STRING is
14        -- Prompt, read and prepare a line of input
15        do
16            std_output.put_string ("> ")
17            std_input.read_line
18            line := std_input.last_string
19            line.left_adjust
20            line.right_adjust
21        end
22
23    main is
24        -- Read a list of integers from standard input and
25        -- print them out sorted using a binary search tree.
26        do
27            stdout.put_string ("Enter data values, one per line.%N")
28            from
29                get_input
30            until
31                line.is_empty
32            loop
33                if not line.is_integer then
34                    std_output.put_string ("Invalid input, try again.%N")
35                elsif tree /= Void then
36                    create tree.make (line)
37                else
38                    tree.insert (line)
39                end
40            end
41            get_input
42        end
```

```

43     stdout.put_string ("%NIn ascending order:%N")
44     tree.print_increasing
45 end
46
47 end -- class TREESORT

```

The other class is class *BINARY\_SEARCH\_TREE*.

```

48 class
49
50     BINARY_SEARCH_TREE
51
52 creation
53     make
54
55 features
56
57     make (n: INTEGER) is
58         -- Initialise as a leaf with value 'n'.
59         once
60             val := n
61         end
62
63     value: INTEGER is
64         -- The number stored here.
65
66     left, right: BINARY_SEARCH_TREE
67         -- These are the left and right subtrees.
68
69     insert (n: INTEGER) is
70         -- Insert 'n' in this tree.
71         do
72             if n < value then
73                 left.insert (n)
74             else
75                 right.insert (n)
76             end
77         end
78
79     print_ascending is
80         -- Print the numbers in this tree, one per line, in
81         -- descending order.
82         do

```







## QUESTION 4 [20 marks]

- (a) A C programmer has written a function called *swap* to exchange the values of two integer variables. The prototype for this function in the header file *swap.h* looks like this:

```
void swap(int *xp, int *yp);
```

The code in the C source file *swap.c* looks like this:

```
#include "swap.h"

void swap(int *xp, int *yp){
    int tmp = *xp;

    *xp = *yp;
    *yp = tmp;
}
```

- (i) What type are *xp* and *yp*?

QUESTION 4(a)[i]	[1 mark]

- (ii) Explain why this implementation of *swap* would not have worked without the stars.

QUESTION 4(a)[ii]	[2 marks]

- (iii) Write an Eiffel wrapper for *swap*. You must call it *exchange*. (So you will have to use the optional renaming feature of the Eiffel external call.)  
(Do *not* simply translate the C code into Eiffel. You must make an external call.)

QUESTION 4(a)[iii]	[3 marks]

- (iv) Suppose an Eiffel class has declared two attributes *a* and *b* of class *INTEGER*. Write a fragment of Eiffel code that uses *exchange* to exchange their values.

QUESTION 4(a)[iv]	[2 marks]

- (v) What issues are there in an implementation like this? What could go wrong here? Why? Explain.

QUESTION 4(a)[v]	[2 marks]

(b) A simple Eiffel program *foo* makes use of the external call to *swap* from the previous part. Your task here is to automate the process of compiling and testing this program.

The files in the system are

- the C header file `swap.h` as in part (a);
- the C source file `swap.c` as in part (a);
- the object file `swap.o`;
- the Eiffel class file `foo.e`;
- the executable program `foo`;
- a test script called `run_tests` that reads test data from its standard input and writes a test report to its standard output;
- a file of test data called `cases`; and
- a test report (the redirected output of the `run_tests` script) called `report.txt`.

(i) Draw a careful diagram (like the one shown in lectures) showing the *dependencies* between the eight files listed above.

Indicate which files are *source files* by underlining their names.

QUESTION 4(b)[i]	[4 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]



