

**THE AUSTRALIAN NATIONAL UNIVERSITY**

*First Semester 2000*

**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 the 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]

This question is about implementing expression trees that can be used for evaluating and printing simple arithmetic expressions. The designer has written the following deferred class. Each different type of expression will implement this interface, so that client code can build and traverse a polymorphic tree structure representing any arithmetic expression.

```
deferred class
  EXPRESSION
  -- Parent class for all types of expressions.

feature {ANY} -- Queries

  value : INTEGER is
    -- The value.
  deferred
  end

  to_string : STRING is
    -- String representation in infix order.
  deferred
  end

end -- class EXPRESSION
```

So far engineers have only written one descendant of class *EXPRESSION*, for handling integer constant expressions.

```
class
  CONSTANT
  -- Real constant expressions.

inherit
  EXPRESSION

creation
  make

feature {NONE} -- Private features

  make (v : INTEGER) is
    -- Initialise value to v.
  do
    value := v
  ensure
    value = v
  end
```









## QUESTION 2 [20 marks]

- (a) The following excerpt from a version of class *CHARACTER\_DATA* of the WebText program contains an attempted implementation of the *replace\_entities* feature. It contains at least ten (10) defects.

The effect of this code is supposed to be that a call to *replace\_entities* replaces each valid “character entity” in *contents* with the character it represents (usually a special character).

Recall that character entities are sequences of characters that start with ‘&’ and end with ‘;’. The characters in between are either a sequence of letters and digits which can be looked up in *dictionary*, or the character ‘#’ followed by an integer.

```
feature {ANY} -- Attributes.  
  
    contents : STRING  
        -- The current version of this data.  
5  
feature {ANY} -- Commands.  
  
    replace_entities is  
        -- Replace every character entity in contents  
10        -- with the appropriate character.  
    local  
        i : INTEGER  
    do  
        from i = 1 until i > contents.count loop  
15        if content.item (i) = '&' then  
            replace_one_entity_at (i)  
        end  
        i := i + 1  
    end  
20 end  
  
feature {NONE} -- Private auxiliary commands.  
  
    replace_entity_at (i : INTEGER) is  
25        -- Replaces a possible character entity in  
        -- contents starting at position i.  
    require  
        contents.item (i) = '&'  
    local  
30        j : INTEGER  
        entity : STRING  
        c : CHARACTER  
        ok : BOOLEAN  
    do
```

```

35     from  $j := i + 1$  until  $contents.item(j) = \text{'\#}'$  loop
         $entity.add\_last(contents.item(j))$ 
    end
    if  $entity.first = \text{'\#}'$  then
         $entity.remove\_first(1)$ 
40     if  $entity.is\_integer$  then
         $c := s.to\_integer.to\_character$ 
         $ok := True$ 
    end
    else if  $dictionary.has(s)$  then
45      $c := dictionary.at(s).to\_character$ 
         $ok := true$ 
    end
    if  $ok$  then
         $contents.remove\_between(i, j)$ 
50      $contents.insert(c, i)$ 
    end
end

dictionary : DICTIONARY[CHARACTER, STRING] is
55     -- Lookup table for entities: keys are strings,
        -- stored values are characters.

```

**Perform a careful review of this code, and locate all ten defects.** For each defect, write the number of the line on which it occurs, together with a clear, concise statement of what is wrong, in the answer box on the next page.

For the purposes of this question, missing require or ensure clauses and missing implementation comments do not count as defects. Note that the implementation of *dictionary* has been omitted deliberately—this is also not a defect.

You may find the following excerpts from the interface documentation of class *STRING* useful for checking the correctness of some of the library calls.

```

remove_between (low, up: INTEGER)
    -- Remove characters between positions low and up.
require
     $valid\_index(low)$  and  $valid\_index(up)$ 
     $low \leq up$ 
ensure
     $count = \mathbf{old} \ count - (up - low + 1)$ 
...
insert (ch: CHARACTER; index: INTEGER)
    -- Insert ch after position index.
require
     $0 \leq index$  and  $index \leq count$ 
ensure
     $item(index + 1) = ch$ 

```









(c) Outline the roles of and the 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 3(c)[i]	[2 marks]

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

QUESTION 3(c)[ii]	[2 marks]

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

QUESTION 3(c)[iii]	[2 marks]













## QUESTION 5 [20 marks]

Willow's Widgets is building a new computer system to access student information over the web at Sunnydale University. Willow Rosenberg — lead engineer at Willow's Widgets — is something of a hacker at heart, and so she has decided to implement the entire system in C. So far Willow has built the following C modules:

**student:** This module implements an abstract data-type of students.

**html doc:** This module implements an abstract data-type of HTML documents.

Each of these modules has an implementation given in a `.c` file, and a signature given in a `.h` file. Willow's policy is for each module implementation file to **#include** the corresponding header file.

To test the system so far, Willow has written a program `harness.c`, which she compiles to produce an executable file called `harness`. When run, this program performs the following actions:

1. Information describing a student is read in from the standard input.
2. A value of the student data-type is constructed from this information.
3. A value of the HTML data-type is then constructed to present a nicely formatted view of the data held in the student data structure.
4. The HTML document is printed on the standard output.

- (a) Draw a graph showing the dependencies between the various source and object code files in the system just described.

QUESTION 5(a)	[5 marks]
---------------	-----------





(d) Consider the following short answer questions

- (i) Give an example of a situation in which the UNIX make tool is more useful than an automated compiler, like the one used for Eiffel, that is capable of determining the dependencies between files by itself.

QUESTION 5(d)[i]	[1 mark]

- (ii) Give one example of why a software company might need to maintain various versions of its product using a version/configuration management tool like RCS.

QUESTION 5(d)[ii]	[1 mark]

- (iii) Give one possible use for inserting a string like `$Revision: 1.5 $` into a program maintained using the RCS tool.

QUESTION 5(d)[iii]	[1 mark]





Additional answers to QUESTION —(—)[—]