Mock exam
1 Terminology

• 1. Objects and classes

✓ All types are either reference or expanded.
  o If an object is of an expanded type, its fields cannot be modified at runtime.
  o Suppliers of class $C$ can use all the features of class $C$.

✓ A class can be both a supplier and a client.
  o If $C$ is a deferred class, then no entity can exist in a program with static type $C$. 
1 Terminology

• 2. About loops:
  o A loop must always define an invariant, otherwise the program will not compile.
  o The variant of the loop must increase with every loop iteration.
✓ The variant of the loop must decrease with every loop iteration and must always be >= 0.
✓ It is possible that a loop will never terminate.
1 Terminology

• 3. Information hiding is the technique of presenting programmers with an interface that...
  o ... only contains the public features of a class.
  o ... includes only features that have built-in security controls.
  o ... includes a superset of the properties of a software element.
  ✓ ... includes only a subset of the properties of a software element.
1 Terminology

• 4. Inheritance and polymorphism
  o A deferred class cannot inherit from an effective class.
  o A class $C$ cannot inherit from two different classes $A1$ and $A2$, if both $A1$ and $A2$ have a common ancestor class.
  ✓ An instruction $o.f$ at runtime can result in executing different routines.
  o An entity of static type $C$ can only be attached to an object of a type that is an ancestor of $C$.
  o In class $C$ a feature $f$ inherited from class $A$ can only be redefined if $f$ is deferred in $A$. 
1 Terminology

• 5. Design by Contract
  o The creation procedure only needs to ensure that the invariant of the created object holds at the end of the procedure body.
  ✓ Every procedure ensures that the postcondition True holds.
  o The class invariant needs to hold before every procedure call.
  ✓ A procedure pp, that redefines another procedure p, needs to ensure the postcondition of procedure p.
  o A procedure pp, that redefines another procedure p, can provide a precondition that is stronger than the one given by procedure p.

Precondition weakening and Postcondition strengthening!
require else... ensure then...
2 Design by Contract
2 Design by Contract

class PERSON
create make
feature {NONE} -- Creation
   make (n: STRING)
      -- Create a person with a name ‘n’.
         require
         n_nonempty: n/= Void and then not n.is_empty
   do
      -- Create a copy of the argument and assign it to ‘name’
         name := n.twin
      ensure
         name_set: name = n
         name_set: name ~ n
end
2 Design by Contract

84 feature -- Basic operations

86  marry (p: PERSON)
    -- Marry ‘p’.
88    require
90
90  p_exists: p /= Void
92  p_not_current: p /= Current
94  current_not_married: not is_married
94  target_not_married: not p.is_married
96
98    do
100       spouse := p
102       p.accept_marriage (Current)
104    ensure
106
106  current_spouse_is_p: spouse = p
108  is_married: is_married
110
113  end
3 Digital Root
The digital root (Quersumme) of a number is found by adding together the digits that make up the number. If the resulting number has more than one digit, the process is repeated until a single digit remains.

Example input and output

<table>
<thead>
<tr>
<th>Input</th>
<th>Digital root</th>
<th>Computation process</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>6</td>
<td>$1 + 2 + 3$</td>
</tr>
<tr>
<td>5720</td>
<td>5</td>
<td>$1 + 4 \leftarrow 14 = 5 + 7 + 2 + 0$</td>
</tr>
<tr>
<td>9999999</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Your task in this problem is to implement a function that, given a non-negative number, calculates the digital root and returns it as the result. Fill in the body of function digital_root below. Your implementation should work with INTEGER objects only. You might find the following two operators of class INTEGER useful: $\text{\\mbox{\textbackslash\,\textbackslash}}$ (modulo) and $\text{\textbackslash\,\textbackslash\,\textbackslash}$ (integer division).

There exists a closed-form solution to this problem:

$$
\text{digital_root}(n) = \begin{cases} 
0 & \text{if } n = 0, \\
9 & \text{if } n \neq 0 \text{ and then } n \text{\,\textbackslash\,\textbackslash\,\textbackslash} 9 = 0, \\
\frac{n - 9\text{\,\textbackslash\,\textbackslash\,\textbackslash}}{9} & \text{otherwise.}
\end{cases}
$$

You are not allowed to use this to solve this programming exercise!
3 Digital Root

Wiederholt die Quersummenberechnung bis das Resultat einstellig ist.

Berechnet die Quersumme von 'number'
4 Doubly linked lists
4 Doubly linked list

INTEGER_LIST

INTEGER_LIST_CELL

value

next

previous

first

last

count

3
4 Doubly linked list - extend

Gegeben:

Ziel:
Zelle mit Wert 9 am Ende einfügen.
4 Doubly linked list - extend

Gegeben:

**INTEGER_LIST**

```
first
last
count 3
```

create el.set_value(a_value)

**INTEGER_LIST**

```
first
last
count 3
```
4 Doubly linked list - extend

Gegeben:

Gegeben:

create el.set_value(a_value)

last.set_next(el)
4 Doubly linked list - extend

Gegeben:

```
| 4 | 2 | 6 |
```

```
create el.set_value(a_value)
```

```
last.set_next(el)
el.set_previous(last)
```

```
INTEGRAL_LIST
```

```
first
last
count 3
```

```
| 4 | 2 | 6 |
```

```
el |
```

```
INTEGRAL_LIST
```

```
first
last
count 3
```

```
| 4 | 2 | 6 | 9 |
```

4 Doubly linked list - extend

Gegeben:

4 Doubly linked list - extend

create el.set_value(a_value)

last.set_next(el)
el.set_previous(last)
last := el
4 Doubly linked list - extend

Gegeben:

```
create el.set_value(a_value)
last.set_next(el)
el.set_previous(last)
last := el
count := count + 1
```

```java
el.set_value(a_value);
last.set_next(el);
el.set_previous(last);
last = el;
count = count + 1;
```
4 Doubly linked list - extend

Gegeben:

Achtung! Was, wenn die Liste leer ist?

last.set_next(el)
last.set_previous(last)
last := el
count := count + 1
4 Doubly linked list - extend

Gegeben:

Ziel:
Zelle mit Wert 9 einfügen.
4 Doubly linked list - extend

Gegeben:

```
Gegeben:

first
last
count 0

INTEGER_LIST
```

create el.set_value(a_value)

```
first
last
count 0

INTEGER_LIST
```
4 Doubly linked list - extend

Gegeben:

\[
\begin{align*}
\text{first} & := \text{el} \\
\text{create el.set_value(a_value)} \\
\text{first} & := \text{el}
\end{align*}
\]
4 Doubly linked list - extend

Gegeben:

```
create el.set_value(a_value)
first := el
last := el
```
4 Doubly linked list - extend

Gegeben:

```
INTEGRAL_LIST
```

```
first
last
count
```

```
create el.set_value(a_value)
first := el
last := el
count := count + 1
```

```
INTEGRAL_LIST
```
4 Doubly linked list - has

cursor: INTEGER_LIST_CELL

from
until
loop

end
4 Doubly linked list - has

cursor: INTEGER_LIST_CELL
from
  cursor := first
until
loop

end
4 Doubly linked list - has

cursor: INTEGER_LIST_CELL

from
cursor := first
until
cursor = Void or Result
loop

end
cursor: INTEGER_LIST_CELL

from
  cursor := first
until
  cursor = Void or Result
loop
  if cursor.value = a_value then
    Result := True
  end
end
4 Doubly linked list - has

```
cursor: INTEGER_LIST_CELL

from
cursor := first
until
cursor = Void or Result
loop
  if cursor.value = a_value then
    Result := True
  end
end
cursor := cursor.next
end
```
I DON'T WANT TO GO TO SCHOOL! I HATE SCHOOL!
I'D RATHER DO ANYTHING THAN GO TO SCHOOL!