

Esercizio 1) [10 punti]

Marcare le affermazioni che si ritengono vere. Ogni domanda può avere un qualunque numero naturale di affermazioni vere. Vengono **assegnati** 0.5 punti sia per ogni affermazione *vera che viene marcata* che per ogni affermazione *falsa che viene lasciata non marcata*. Analogamente vengono **sottratti** 0.5 punti sia per ogni affermazione *falsa che viene marcata* che per ogni affermazione *vera che viene lasciata non marcata*.

1. ...
 - a. Una classe *deferred* non può ereditare da una classe *effective*
 - b. Una classe *C* non può ereditare da due classi differenti *B1* e *B2*, se sia *B1* che *B2* hanno una stessa classe *A* come antenato comune
 - c. Ad un'entità di tipo statico *C* si può assegnare un oggetto di un tipo che è un antenato di *C*
 - d. In una classe *C* una feature *f* ereditata dalla classe *A* può essere ridefinita soltanto se *f* è *deferred* in *A*
2. Una query ...
 - a. ... può essere usata come procedura di creazione
 - b. ... può essere implementata come una *routine*
 - c. ... può apparire nelle precondizioni e postcondizioni di una qualunque routine
 - d. ... può apparire nell'invariante di classe
3. ...
 - a. Una funzione è una query implementata mediante memorizzazione
 - b. Un attributo è una query implementata mediante memorizzazione
 - c. Una routine è l'implementazione di una feature mediante computazione
 - d. Un attributo è l'implementazione di una feature mediante computazione
4. ...
 - a. Il risultato ritornato da una query è fornito sempre da una funzione
 - b. Il risultato ritornato da una query è fornito sempre da un attributo
 - c. Il risultato ritornato da una query è fornito da una funzione o da un attributo
 - d. Il risultato ritornato da un comando è fornito da una procedura o da un attributo
5. ...
 - a. Se *C* è una classe *deferred* allora non possono esistere nel programma entità di tipo statico *C*
 - b. Se *C* è una classe che usa la *feature* di un'altra classe *S* allora *C* è *supplier* (fornitore) di *S*
 - c. La classe *C* può essere contemporaneamente un *supplier* (fornitore) ed un *client* (cliente)
 - d. Se il tipo di una variabile è un tipo espanso allora il valore della variabile a tempo di esecuzione (*run-time*) è un oggetto

SOLUZIONE:

1. ...
 - a. Una classe *deferred* non può ereditare da una classe *effective*
 - b. Una classe *C* non può ereditare da due classi differenti *B1* e *B2*, se sia *B1* che *B2* hanno una stessa classe *A* come antenato comune
 - c. Ad un'entità di tipo statico *C* si può assegnare un oggetto di un tipo che è un antenato di *C*
 - d. In una classe *C* una feature *f* ereditata dalla classe *A* può essere ridefinita soltanto se *f* è *deferred* in *A*
2. Una query ...
 - a. ... può essere usata come procedura di creazione
 - b. ... può essere implementata come una *routine*
 - c. ... può apparire nelle precondizioni e postcondizioni di una qualunque routine
 - d. ... può apparire nell'invariante di classe

3. ...
 - a. Una funzione è una query implementata mediante memorizzazione
 - b. Un attributo è una query implementata mediante memorizzazione
 - c. Una routine è l'implementazione di una feature mediante computazione
 - d. Un attributo è l'implementazione di una feature mediante computazione
4. ...
 - a. Il risultato ritornato da una query è fornito sempre da una funzione
 - b. Il risultato ritornato da una query è fornito sempre da un attributo
 - c. Il risultato ritornato da una query è fornito da una funzione o da un attributo
 - d. Il risultato ritornato da un comando è fornito da una procedura o da un attributo
5. ...
 - a. Se C è una classe *deferred* allora non possono esistere nel programma entità di tipo statico C
 - b. Se C è una classe che usa la *feature* di un'altra classe S allora C è *supplier* (fornitore) di S
 - c. La classe C può essere contemporaneamente un *supplier* (fornitore) ed un *client* (cliente)
 - d. Se il tipo di una variabile è un tipo espanso allora il valore della variabile a tempo di esecuzione (*run-time*) è un oggetto

Esercizio 2) [10 punti]

La classe *INT_LINKABLE* modella un elemento di una lista che può rappresentare valori interi. La sua implementazione è la seguente:

```

class
  INT_LINKABLE
create
  make

feature -- accesso
  value : INTEGER
    -- L'intero memorizzato in questo elemento

  next : INT_LINKABLE
    -- Il successivo elemento della lista

feature -- operazioni fondamentali
  make (i : INTEGER)
    -- crea l'elemento
  do
    value := i
  end

  link_to (other: INT_LINKABLE)
    -- collega questo elemento con `other`
  do
    next := other
  ensure
    next = other
  end

  insert_after (other: INT_LINKABLE)
    -- inserisce questo elemento dopo `other` conservando quello che c'era dopo di esso
  require
    other /= Void
  do
    link_to (other.next)
    other.link_to (Current)
  ensure
    other.next = Current
    other.next.next = old other.next
  end

end

```

La classe *INT_LINKED_LIST* modella una lista di interi. Una parte della sua implementazione è fornita qua sotto:

```

class
  INT_LINKED_LIST

feature -- accesso
  first_element: INT_LINKABLE
    -- Il primo elemento della lista

  last_element: INT_LINKABLE
    -- L'ultimo elemento della lista

```

count: *INTEGER*

-- Il numero di elementi della lista

feature -- operazioni fondamentali

has (*a_value*: *INTEGER*): *BOOLEAN*

-- La lista contiene `a\_value`?

local

temp, *pre_temp*: *INT_LINKABLE*

do

from

temp := *first_element*;

pre_temp := **Void**

invariant

not Result implies (*pre_temp* /= **Void** **implies** *pre_temp.value* /= *a_value*)

until

(*temp* = **Void**) or **Result**

loop

if *temp.value* = *a_value* **then**

Result := **True**

end

pre_temp := *temp*

temp := *temp.next*

end

end

get_item (*a_value*: *INTEGER*): *INT_LINKABLE*

-- Ritorna l'elemento contenente `a\_value`, se esiste

local

temp, *pre_temp*: *INT_LINKABLE*

do

from

temp := *first_element*;

pre_temp := **Void**

invariant

Result = **Void** **implies** (*pre_temp* /= **Void** **implies** *pre_temp.value* /= *a_value*)

until

(*temp* = **Void**) or (**Result** /= **Void**)

loop

if *temp.value* = *a_value* **then**

Result := *temp*

end

pre_temp := *temp*

temp := *temp.next*

end

ensure

(**Result** /= **Void**) **implies** **Result.value** = *a_value*

end

invariant

count >= 0

last_element /= **Void** **implies** *last_element.next* = **Void**

count = 0 **implies** (*first_element* = *last_element*) **and** (*first_element* = **Void**)

count = 1 **implies** (*first_element* = *last_element*) **and** (*first_element* /= **Void**)

count > 1 **implies** (*first_element* /= *last_element*) **and** (*first_element* /= **Void**) **and** (*last_element* /= **Void**) **and**

then (*first_element.next* /= **Void**)

end

Aggiungere alla classe *INT_LINKED_LIST* una feature *insert_multiple_after* (*new*, *target*: *INTEGER*) che inserisce un nuovo elemento di valore *new* dopo tutte le occorrenze dell'elemento di valore *target*, se quest'ultimo è presente. Altrimenti il nuovo elemento viene inserito alla fine della lista.

feature -- operazioni fondamentali

insert_multiple_after (*new*, *target*: *INTEGER*)

- Inserisce elemento con valore '*new*' dopo tutti gli element con valore '*target*', se ne esistono
- Altrimenti inserisce elemento con valore '*new*' alla fine della lista

local

do

ensure

end

SOLUZIONE:**feature** -- operazioni fondamentali*insert_multiple_after* (*new*, *target*: *INTEGER*)-- Inserisce elemento con valore *new* dopo tutti gli elementi con valore *target*, se ne esistono-- Altrimenti inserisce elemento con valore *new* alla fine della lista**local***new_item*, *temp*: *INT_LINKABLE**target_exist*: *BOOLEAN***do****if** *has(target)* **then****from***temp* := *first_element***until***temp* = **Void****loop****if** *temp.value* = *target* **then****create** *new_item.make* (*new*)*new_item.insert_after* (*temp*)*count* := *count* + 1**if** *temp* = *last_element* **then***last_element* := *new_item***end***temp* := *new_item.next***else***temp* := *temp.next***end****end****else** -- la lista non contiene *target***create** *new_item.make* (*new*)**if** *count*=0 **then***first_element* := *new_item**last_element* := *new_item***else***new_item.insert_after* (*last_element*)*last_element* := *new_item***end***count* := *count* + 1**end****ensure***count* > **old** *count*not (**old** *has(target)*) **implies** *last_element.value* = *new*;**old** *has(target)* **implies** *get_item(target).next.value* = *new*;**end**

Nel testo dell'esercizio con "inserisce ... dopo tutte le occorrenze" si intendeva "inserisce ... dopo ogni occorrenza": la soluzione sopra descritta si riferisce a questa interpretazione.

Si noti che l'ultima post condizione garantisce soltanto che sia stato fatto l'inserimento di *new* dopo la prima occorrenza di *target* nella lista.

Esercizio 3) [10 punti]

Completare i contratti (pre-condizioni, post-condizioni, invarianti di classe) della classe *CRUISE_CONTROL* sotto descritta che modella un sistema per il controllo della velocità di crociera di un veicolo in modo da rispecchiare la seguente specifica informale:

1. Il sistema di controllo della velocità di crociera (cruise control = CC) è in uno dei seguenti stati: *spento*, *acceso*, *abilitato* (cioè acceso con controllo della velocità), *disabilitato* (cioè acceso ma senza controllo della velocità).
2. Il comando di abilitazione (*enable_CC*) richiede di aver impostato una velocità di crociera > 0 .
3. La velocità può essere impostata o re-impostata (*set_speed*) in qualunque momento CC è acceso. Lo spegnimento di CC causa la de-impostazione della velocità di crociera.
4. In stato *abilitato* la pressione sui freni (*brake*) o la pressione dell'acceleratore (*increase_gas*) causano la disabilitazione di CC.
5. In stato *disabilitato* il rilascio dell'acceleratore (*decrease_gas*) o il comando riprendi velocità (*recover_speed*) causano l'abilitazione di CC.

Non c'è relazione tra il numero di righe vuote ed il numero di contratti da scrivere. Non è necessario conoscere altro codice all'infuori di quello fornito.

deferred class

CRUISE_CONTROL

feature {ANY} -- accesso

cruise_speed : *INTEGER*

-- La velocità di crociera che è stata impostata. Vale **0** quando non è stata impostata.

feature {ANY} -- stato

is_CC_on : *BOOLEAN*

-- Il sistema CC è acceso?

is_CC_enabled : *BOOLEAN*

-- Il sistema CC è abilitato?

require

is_CC_on

deferred**ensure**

Result = (*cruise_speed* $\neq$ **0**)

end**feature** {ANY} -- operazioni sul veicolo

brake : *BOOLEAN*

-- Sono premuti i freni

require**deferred****ensure****end**

increase_gas : *BOOLEAN*

-- Viene premuto l'acceleratore

require

deferred

ensure

end

decrease_gas : *BOOLEAN*

-- Viene rilasciato l'acceleratore

require

deferred

ensure

end

feature {ANY} -- operazioni sul sistema

set_speed (*a_speed* : *INTEGER*)

-- Assegna la velocità di crociera.

require

deferred

ensure

end

enable_CC

-- Abilita il sistema CC.

require

deferred

ensure

end

disable_CC

-- Disabilita il sistema CC.

require**deferred****ensure****end***switch_on*

-- Accende il sistema CC.

require**deferred****ensure****end***switch_off*

-- Spegne il sistema CC.

require**deferred****ensure****end****invariant****SOLUZIONE:****deferred class***CRUISE_CONTROL***feature** {ANY}-- accesso*cruise_speed* : *INTEGER*-- La velocità di crociera che è stata impostata. Vale **0** quando non è stata impostata.**feature** {ANY}-- stato*is_CC_on* : *BOOLEAN*

-- Il sistema CC è acceso?

is_CC_enabled : *BOOLEAN*

-- Il sistema CC è abilitato?

require*is_CC_on***deferred****ensure****Result** = (*cruise_speed* /= **0**)**end**

feature {ANY}-- operazioni sul veicolo

brake

-- Sono premuti i freni

require

-- l'operazione non ha prerequisiti

deferred

ensure

old is_CC_enabled implies not is_CC_enabled

end

increase_gas

-- Viene premuto l'acceleratore

require

-- l'operazione non ha prerequisiti

deferred

ensure

old is_CC_enabled implies not is_CC_enabled

end

decrease_gas

-- Viene rilasciato l'acceleratore

require

-- l'operazione non ha prerequisiti

deferred

ensure

old not is_CC_enabled implies is_CC_enabled

end

feature {ANY}-- operazioni sul sistema

set_speed (a_speed : INTEGER)

-- Assegna la velocità di crociera

require

is_CC_on

a_speed /= 0

deferred

ensure

cruise_speed = a_speed

end

```

enable_CC
  -- Abilita il sistema CC
  require
    not is_CC_enabled
    cruise_speed  $\neq$  0
  deferred
  ensure
    is_CC_enabled
  end

disable_CC
  -- Disabilita il sistema CC
  require
    is_CC_enabled
  deferred
  ensure
    not is_CC_enabled
  end

switch_on
  -- Accende il sistema CC
  require
    not is_CC_on
  deferred
  ensure
    is_CC_on
    not is_CC_enabled
  end

switch_off
  -- Spegne il sistema CC
  require
    is_CC_on
  deferred
  ensure
    not is_CC_on
    cruise_speed = 0
    not is_CC_enabled
  end

invariant
  current_speed  $\geq$  0
  is_CC_enabled implies is_CC_on

```

Si noti che per le tre feature relative ad operazioni sul veicolo (*brake*, *increase_gas*, *decrease_gas*) non ha senso inserire pre-condizioni dal momento che tali operazioni sul veicolo devono comunque poter essere effettuate indipendentemente dai valori delle precondizioni.