Einführung in die Programmierung

ETH Zurich

Datum: 19.08.2013

Nachname, Vorname: ....

Leginummer: ....

Ich bestätige mit meiner Unterschrift, dass ich die Prüfung unter regulären Bedingungen schreiben konnte und dass ich die untenstehenden Anweisungen gelesen und verstanden habe.

Unterschrift: ....

Anweisungen:

- Prüfungsdauer: 120 Minuten.
- Ausser einem Deutsch-Englisch Wörterbuch sind keine Hilfsmittel erlaubt.
- Es darf *nicht* mit Bleistift geschrieben werden.
- Bitte schreiben Sie Ihre Leginummer auf *alle* Blätter.
- Alle Lösungen können direkt auf die Aufgabenblätter geschrieben werden. Falls Sie mehr Platz für die Lösungen benötigen, bitten Sie die Aufsicht um zusätzliches offizielles Papier. Sie dürfen *nicht* Ihr eigenes Papier benützen.
- Es darf nur eine Lösung pro Aufgabe abgegeben werden. Ungültige Lösungsversuche sind klar durchzustreichen.
- Bitte schreiben Sie leserlich! Es können nur lesbare Lösungen korrigiert werden.
- Teilen Sie sich die Zeit gut ein. Beachten Sie dazu die Anzahl der Punkte, die für die einzelnen Aufgaben vergeben werden.
- Bitte melden Sie sich *sofort*, falls Sie sich während der Prüfung gestört fühlen.

Viel Erfolg!

Frage	Anzahl möglicher Punkte	Punkte
1	10	
2	14	
3	12	
4	13	
5	16	
Gesamt	65	

1 Multiple choice (10 Punkte)

Kreuzen Sie die korrekten Aussagen an. Richtig angekreuzte/nicht-angekreuzte Aussagen werden mit je 0.5 Punkten bewertet. Falsch angekreuzte/nicht-angekreuzte Aussagen werden mit je 0 Punkten bewertet.

Beispiel:

Welche der folgenden Aussagen sind korrekt?

a.	Die Sonne ist eine Masse von glühendem Gas.	☒	0.5 points
b.	$2 \times 4 = 8$	☐	0 points
c.	“Rösti” ist eine Wurst.	☒	0 points
c.	C ist eine object-orientierte Programmiersprache.	☐	0.5 points

---

1. Objekte und Klassen.
   - a. Ein Programm kann eine unlimitierte Anzahl Objekte eines bestimmten Typs erstellen. ☐
   - b. Der Zugriff auf Erzeugungsprozeduren (Creation procedures) kann auf eine limitierte Anzahl Klienten eingeschränkt werden. ☐
   - c. Jede Funktion in Eiffel kann auch eine Erzeugungsprozedur sein. ☐
   - d. Eine generische Klasse *C* [*G*] repräsentiert genau einen Typ. ☐
   - e. Das öffentliche Interface einer Klasse beinhaltet alle Features, welche allen Klienten zur Verfügung stehen. ☐

2. Kontrollstrukturen und Rekursion.
   - a. Wenn wir wissen, dass eine Schleife die Schleifenvariante verkleinert und diese niemals weniger wie 5 beträgt, dann wissen wir, dass diese Schleife terminiert. ☐
   - b. Die Schleifeninvariante darf während der Initialisierung einer Schleife verletzt werden (bevor die Schleife betreten wird). ☐
   - c. Die Schleifeninvariante sagt aus, wie oft eine Schleife ausgeführt wird. ☐
   - d. Eine Eiffel-Prozedur darf sich auf dem gleichen Objekt beliebig oft selber aufrufen. ☐
   - e. Eine Schleife kann immer als eine endliche Sequenz von Konditional- und Verbund-Instruktionen ausgedrückt werden. ☐

3. Vererbung und Polymorphismus.
   - a. Alle Klassen in Eiffel erben implizit von der Klasse *OBJECT*. ☐
   - b. Eine Variable kann zur Laufzeit an ein Objekt gebunden sein, dessen dynamischer Typ vom statischen Typ der Variable erbt. ☐
   - c. Eine Variable kann zur Laufzeit an ein Objekt gebunden sein, dessen dynamischer Typ gleich dem statischen Typ der Variable ist. ☐
   - d. Eine Variable kann zur Laufzeit an ein Objekt gebunden sein, dessen dynamischer Typ ein Vorfahre des statischen Typs der Variable ist. ☐
   - e. Für ein Object *obj* kann der Aufruf *obj.is_equal*(*obj*) das Resultat **False** haben. ☐

4. Entwurf gemäss Vertrag (Design by Contract).
   - a. In einer Erzeugungsprozedur ist die Klasseninvariante zum Beginn und Ende der Prozedur efüllt. ☐
   - b. Eine Klasseninvariante kann im Rumpf einer Prozedur ohne Konsequenzen verletzt werden. ☐
   - c. Die Invariante einer Klasse wird mit den Invarianten seiner Vorfahren verUNDet. ☐
   - d. Bei einer redefinierten Funktion kann man eine stärkere Vorbedingung angeben. ☐
   - e. Bei einer redefinierten Prozedur kann man eine schwächere Nachbedingung angeben. ☐

2 Vererbung und Polymorphismus (14 Punkte)

Die Klassen *SCIENTIST*, *COMPUTER_SCIENTIST*, *BIOLOGIST*, und *PET* (siehe unten) sind Teil eines Programms zur Verwaltung des Soziallebens von Wissenschaftern.

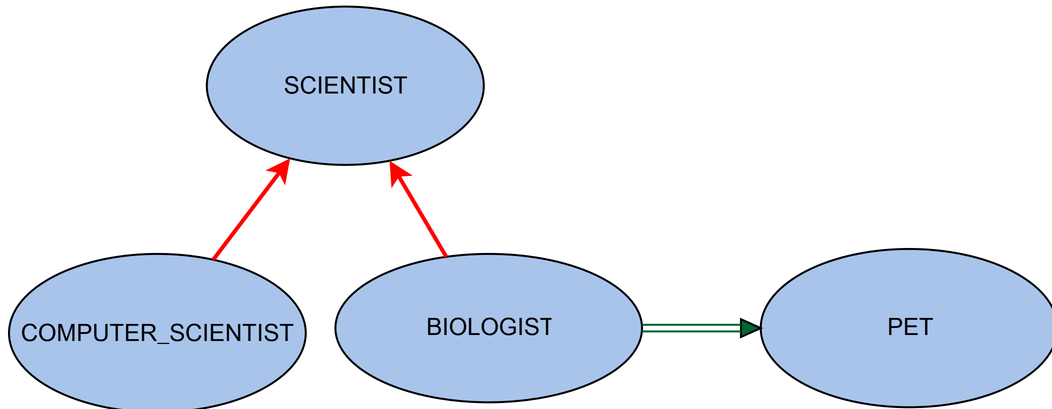

Figure 1: BON Diagram

```
deferred class
  SCIENTIST

feature {NONE} -- Initialization

  make (a_name: STRING)
      -- Initialize Current with 'a_name'.
    require
      a_name_exists: a_name /= Void and then not a_name.is_empty
    do
      name := a_name
    ensure
      name_set: name = a_name
    end

feature -- Access

  name: STRING
      -- Current's name.

feature -- Basic operations

  introduce
      -- Print info about self.
    do
      io.put_new_line
      print ("My name is " + name + "; ")
    end
end
```

```
class
  COMPUTER_SCIENTIST

inherit
  SCIENTIST
    redefine
      introduce
    end

create
  make

feature -- Basic operations

  introduce
      -- Print info about self.
    do
      Precursor
      print ("I am a computer scientist.")
    end
end
```

```
class
  BIOLOGIST

inherit
  SCIENTIST
    rename
      introduce as express
    redefine
      express
    end

create
  make_with_pet

feature {NONE} -- Initialization

  make_with_pet (a_name: STRING; a_pet: PET)
      -- Initialization for 'Current'.
    require
      name_exists: a_name /= Void and then not a_name.is_empty
      pet_exists: a_pet /= Void
    do
      make (a_name)
      pet := a_pet
    ensure
      name_set: name = a_name
      pet_set: pet = a_pet
    end

feature -- Access

  pet: PET
      -- Current biologist's pet.
```

```
feature -- Basic operations

  express
      -- Print info about self.
    do
      Precursor
      print ("I am a biologist. ")
      print ("I have a pet. Its name is " + pet.name + ".")
    end
end
```

```
class
  PET

create
  make

feature {NONE} -- Initialization

  make (pet_name: STRING)
      -- Initialization for 'Current'.
    require
      pet_name_exists: pet_name /= Void and then not pet_name.is_empty
    do
      name := pet_name
    ensure
      pet_name_set: name = pet_name
    end

feature -- Access

  name: STRING
      -- Current pet's name.

feature -- Basic operations

  introduce
      -- Print info about self.
    do
      io.put_new_line
      print ("My name is " + name + " and I tend to be afraid.")
    end
end
```

Geben Sie für die Code-Fragmente unten an ob sie kompilieren oder nicht, indem Sie das entsprechende Kästchen ankreuzen. Falls das Code-Fragment nicht kompiliert, erklären Sie warum und markieren Sie die Linie die nicht kompiliert. Falls es kompiliert, geben Sie den Text an, welchen die Konsole ausgibt sobald das Code-Fragment ausgeführt wird.

Gegeben sind die Deklaration folgender Variablen:

```
a_scientist: SCIENTIST
a_computer_scientist: COMPUTER_SCIENTIST
a_biologist: BIOLOGIST
```

Example 1:

```
(create {PET}.make (“Bob”)).introduce
```

Kompiliert der Code? ☒ Ja ☐ Nein
Ausgabe/Fehlerbeschreibung My name is Bob and I tend to be afraid.

Example 2:

```
Bob.introduce
```

Kompiliert der Code? ☐ Ja ☒ Nein
Ausgabe/Fehlerbeschreibung Der Code kompiliert nicht, weil ”Bob” ein unbekannter Bezeichner ist.

Task 1

```
create a_scientist.make (“Theo”)
a_scientist.introduce
```

Kompiliert der Code? ☐ Ja ☐ Nein
Ausgabe/Fehlerbeschreibung

. . . .

. . . .

Task 2

```
create a_computer_scientist.make (“Heidi”)
a_computer_scientist.introduce
```

Kompiliert der Code? ☐ Ja ☐ Nein
Ausgabe/Fehlerbeschreibung

. . . .

. . . .

Task 3

```
a_scientist := create {COMPUTER_SCIENTIST}.make (“Helen”)
a_scientist.introduce
```

Kompiliert der Code? ☐ Ja ☐ Nein
Ausgabe/Fehlerbeschreibung

. . . .

. . . .

Task 4

```
a_scientist := create {COMPUTER_SCIENTIST}.make ("Hal")
a_computer_scientist := a_scientist
a_computer_scientist.introduce
```

Kompiliert der Code? ☐ Ja ☐ Nein
Ausgabe/Fehlerbeschreibung

....

....

Task 5

```
create a_biologist.make_with_pet ("Reto", create {PET}.make ("Toby"))
a_biologist.express
```

Kompiliert der Code? ☐ Ja ☐ Nein
Ausgabe/Fehlerbeschreibung

....

....

Task 6

```
create a_biologist.make_with_pet ("Kandra", create {PET}.make ("Tom"))
a_computer_scientist := a_biologist
a_computer_scientist.introduce
```

Kompiliert der Code? ☐ Ja ☐ Nein
Ausgabe/Fehlerbeschreibung

....

....

Task 7

```
a_biologist := create {BIOLOGIST}.make_with_pet ("Elmo", create {PET}.make ("
    Hex"))
a_scientist := a_biologist
a_scientist.pet.introduce
```

Kompiliert der Code? ☐ Ja ☐ Nein
Ausgabe/Fehlerbeschreibung

....

....

3 Softwarespezifikation durch Verträge (12 Punkte)

Die Stadt *Lake* hat viele Bibliotheken. Jede dieser Bibliotheken besitzt Bücher und ermöglicht es diese Bücher auszuleihen. Jedes Buch gehört zu einer Bibliothek. Ein Leser kann beliebig viele Bücher einer Bibliothek ausleihen, aber jedes Buch kann jeweils nur von einem Leser aufs Mal ausgeliehen werden.

Die Programmstücke 1 bis 3 zeigen Ausschnitte des Programm-Codes der drei Klassen *LIBRARY*, *READER* und *BOOK* des Software Systems, das zur Verwaltung dieser Bibliotheken verwendet wird. Ihre Aufgabe ist es, die Verträge (Contracts) des Programm-Codes zu schreiben. Dabei soll sowohl die informale Beschreibung oben und auch die Kommentare der jeweiligen Features berücksichtigt werden.

Bitte beachten Sie:

- Zuerst sollten sie alle Kommentare der einzelnen Features durchlesen.
- Der Eiffel-Kompiler stellt sicher, dass Entitäten des Typs **attached** *T* nie **Void** sein können. Zum Beispiel kann das Attribut *books*: **attached** *ARRAYED_SET* [**attached** *BOOK*] nie **Void** sein, und auch keine **Void**-Elemente enhalten, und daher sind Verträge der Form *books* /= **Void** für diese Typen unnötig und sollten für diese Entitäten auch nicht geschrieben werden.
- Die explizite Verwendung von **attached** und **detachable** sowie die entrsprechenden Regeln gelten ausschliesslich fr dieses Aufgabe. (In regulrem Eiffel sind Typen standardmssig **attached**, also wird **attached** normalerweise nicht angegeben. Es wird in dieser Aufgabe zur Klarheit eingesetzt.)
- Wenn Sie für einen Ort denken, dass kein expliziter Vertrag nötig ist, müssen sie dies mit dem Vertrag **True** angeben. Falls eine Vorbedingung, Nachbedingung oder Klasseninvariante leer gelassen wird, gibt es für diese 0 Punkte, auch wenn kein expliziter Vertrag nötig wäre.
- Die Anzahl der gepunkteten Linien ist kein Indikator für die Anzahl fehlender Vertrags-Klauseln.
- Die folgenden Features der Klasse *ARRAYED_SET* könnten hilfreich sein:

```
class ARRAYED_SET [G]

feature
    has (v: G): BOOLEAN
            -- Does current include 'v'?

    is_subset (other: like Current): BOOLEAN
            -- Is current set a subset of 'other'?

    -- Other features omitted.
end
```

Listing 1: Class *LIBRARY*

```
deferred class
    LIBRARY

feature -- Book

    books: attached ARRAYED_SET [attached BOOK]
            -- Set of books the library owns.

    has_book (a_book: attached BOOK): BOOLEAN
            -- Does the library own 'a_book'?
        require

            ......................................................................

            ......................................................................

            ......................................................................
        deferred
        ensure

            ......................................................................

            ......................................................................

            ......................................................................
        end

    acquire (a_book: attached BOOK)
            -- Add 'a_book' to the library.
        require

            ......................................................................

            ......................................................................

            ......................................................................
        deferred
        ensure

            ......................................................................

            ......................................................................

            ......................................................................
        end

    borrower (a_book: attached BOOK): detachable READER
            -- Reader who has borrowed but not yet returned 'a_book'.
            -- Return Void if none.
        require

            ......................................................................

            ......................................................................
```

```
            ..........................................................
        deferred
        ensure
            ..........................................................
            ..........................................................
            ..........................................................
        end

feature -- Operation

    lend (a_book: attached BOOK; a_reader: attached READER)
            -- Lend 'a_book' to 'a_reader'.
            -- Applicable only to books owned by 'Current'.
        require
            ..........................................................
            ..........................................................
            ..........................................................
            ..........................................................
        deferred
        ensure
            ..........................................................
            ..........................................................
            ..........................................................
            ..........................................................
        end

    recover (a_book: attached BOOK)
            -- Recover 'a_book' from its borrower.
            -- Applicable only to books owned by 'Current'.
            -- This is the reverse operation of 'lend'.
        require
            ..........................................................
            ..........................................................
            ..........................................................
            ..........................................................
        deferred
        ensure
            ..........................................................
            ..........................................................
```

```
            ...........................................................................

            ...........................................................................
        end

-- Other features omitted.
invariant

  .....................................................................................

  .....................................................................................

  .....................................................................................

  .....................................................................................
end
```

Listing 2: Class *READER*

```
deferred class
    READER

feature -- Access

    borrowed_books (a_lib: attached LIBRARY): attached ARRAYED_SET [attached
        BOOK]
            -- Set of books borrowed from 'a_lib'.
            -- Return empty set if 'Current' has not currently borrowed
            -- any book from 'a_lib'.
        require

            ...........................................................................

            ...........................................................................

            ...........................................................................
        deferred
        ensure

            ...........................................................................

            ...........................................................................

            ...........................................................................
        end

-- Other features omitted.

invariant

  .....................................................................................

  .....................................................................................

  .....................................................................................
end
```

Listing 3: Class *BOOK*

```
deferred class
    BOOK

feature -- Access

    library: detachable LIBRARY assign set_library
            -- Library to which the book belongs. Immutable once set.

feature{LIBRARY} -- Set

    set_library (a_lib: attached LIBRARY)
            -- Set 'library' to 'a_lib'.
        require

            ..........................................................................

            ..........................................................................
        deferred
        ensure

            ..........................................................................

            ..........................................................................
        end

-- Other features omitted.

invariant

  ..................................................................................

  ..................................................................................

  ..................................................................................
end
```

4 Rekursion (13 Punkte)

Aufgabe 1

Die Funktion *n_th_element* (siehe unten) sollte einen rekursiven Algorithmus implementieren, der für eine Liste *a* das n-te Element einer sortierten Liste (in aufsteigender Reihenfolge) berechnet, welche die gleichen Elemente wie die Liste *a* hat. Beachten Sie, dass die Liste *a* selber nicht sortiert sein muss. Das Beispiel aus Aufgabe 2 zeigt eine korrekte Ausgabe der Funktion *n_th_element*. *Vervollständigen Sie die Implementation, indem sie die fehlenden Ausdrücke einfüllen. Beachte dass die erwartete Implementierung Rekursion verwendet.*

```
n_th_element (a: ARRAYED_LIST [INTEGER]; n: INTEGER): INTEGER
    require
        a /= Void and 1 <= n and n <= a.count
    local
        i, element, pivot: INTEGER
        not_greater, greater: ARRAYED_LIST [INTEGER]
    do
        print ("n = " + n.out + "%N")
        pivot := a [n]

        -- Split up the elements of list 'a', such that 'not_greater' contains all elements
        -- that are less than or equal to 'pivot' (except for the pivot itself),
        -- and 'greater' contains all elements that are greater than 'pivot'.
        create not_greater.make (0)
        create greater.make (0)
        from
            i := 1
        until
            a.count < i
        loop
            element := a [i]
            if element <= pivot and i /= n then
                not_greater.extend (element)
            elseif pivot < element then
                greater.extend (element)
            end
            i := i + 1
        end

        if (.................................................................) then

            Result := (...........................................................)

        elseif (............................................................) then

            Result := (...........................................................

                       ...........................................................)

        elseif (............................................................) then

            Result := (...........................................................

                       ...........................................................)
        end
    end
```

Aufgabe 2

In den folgenden Codefragmenten wird die Funktion *n_th_element* mit verschiedenen Eingaben aufgerufen. *Schreiben sie die Ausgabe auf, wie sie beim Aufruf der Funktion auf der Konsole ausgegeben wird, wenn die Funktion n_th_element korret implementiert wäre.* Beachten Sie, dass die Funktion *n_th_element* bei jedem Aufruf das Argument *n* ausgibt.
Nehmen Sie für diese Aufgabe an, dass die Variable *a* wie folgt deklariert ist:

```
local
    a: ARRAYED_LIST [INTEGER]
```

Beispiel

```
create a.make (0) -- Create an empty list.
a.extend (1)
a.extend (2)
a.extend (−2)
print ("result = " + n_th_element (a, 1).out)
```

Ausgabe:

```
n = 1
n = 1
result = -2
```

Fragment 1

```
create a.make (0)
a.extend (0)
print ("result = " + n_th_element (a, 1).out)
```

Ausgabe:

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

Fragment 2

```
create a.make (0)
a.extend (5)
a.extend (1)
a.extend (3)
a.extend (2)
a.extend (1)
print ("result = " + n_th_element (a, 5).out)
```

Ausgabe:

....

....

....

....

....

....

5 Datenstrukturen (16 Punkte)

In dieser Aufgabe müssen sie mehrere Funktionen für eine generische Klasse *SET* [*G*] (Menge) implementieren.

Eine Menge ist eine Kollektion von unterschiedlichen Objekten. Jedes Element muss einmalig sein, keine zwei Elemente dürfen gleich sein. Alle Operationen müssen diese Eigenschaft erhalten. Die Reihenfolge, in der die Elemente einer Menge dargestellt werden, ist nicht relevant (nicht wie bei einem Tupel oder einer Sequenz). Daher sind die beiden Mengen $\{5, 10, 12\}$ und $\{10, 12, 5\}$ identisch. Mehrere fundamentale Mengen-Operationen können für die Konstruktion einer neuen Menge verwendet werden.

- Vereinigungsmenge: Die Vereinigungsmenge von A und B, geschrieben als $A \cup B$, ist die Menge aller Elemente, die entweder in A oder B vorkommen.
- Schnittmenge: Die Schnittmenge von A und B, geschrieben als $A \cap B$, ist die Menge aller Elemente, die in A und B vorkommen.
- Differenzmenge: Die Differenzmenge von A in B (umgangssprachlich auch A *ohne* B), geschrieben als $A \backslash B$ (oder $A - B$), ist die Menge der aller Elemente, welche in A aber nicht in B vorkommen.

Der Jaccard-Index (oder Koeffizient) misst die Gemeinsamkeit zweier Mengen, und ist definiert als die Kardinalität der Schnittmenge dividiert durch die Kardinalität der Vereinigungsmenge zweier Mengen. Wenn beide Mengen leer sind, ist der Jaccard Koeffizient als 1.0 definiert.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Figure 2: Definition des Jaccard-Index für zwei nicht-leere Mengen A und B.

Ihre Aufgabe ist es, die Lücken in der Klasse *SET* [*G*] einzufüllen. Bitte beachten Sie:

- Ihr Code sollte die gegebenen Verträge (Contracts) erfüllen, und wenn nötig sollten Sie neue Verträge dazu schreiben.
- Die Menge sollte nie **Void** enthalten.
- Die Anzahl der gepunkteten Linien ist kein Indiz für die Anzahl Klauseln oder Codezeilen, die Sie einfüllen sollen.
- Die Implementation der Klasse *SET* [*G*] basiert auf einer Array-Liste. Diese Array-Liste ist so aufgesetzt, dass für Vergleiche von Elementen der Liste der Objekt-Vergleich verwendet wird. Die Funktionen wie *has* und *prune* verwenden also Objekt-Vergleiche und nicht Referenz-Vergleiche um die Elemente in der Liste zu vergleichen. Die folgenden Features der Klasse *ARRAYED_LIST* könnten von Nutzen sein:

```
class ARRAYED_LIST [G]

feature
  has (v: G): BOOLEAN
    -- Does current include 'v'?

  start
    -- Move cursor to first position if any.

  extend (v: G)
    -- Add 'v' to the end.
```

```
    prune (v: G)
        -- Remove first occurrence of 'v', if any, after cursor position.
        -- Move cursor to right neighbor.

-- Other features are omitted.
end
```

```
class
  SET [G]

create
  make_empty

feature {NONE} -- Initialization

  make_empty
      -- Create empty Current.
    do
      create content.make (0)
      content.compare_objects
    ensure
      empty_content: content.is_empty
    end

feature -- Access

  count: INTEGER
      -- Cardinality of the current set.
    do
      Result := content.count
    end

  is_empty: BOOLEAN
      -- Is current set empty?
    do

      ..........................................................................

      ..........................................................................

      ..........................................................................

    end

  has (v: G): BOOLEAN
      -- Does current set contain 'v'?
    require

      ..........................................................................

      ..........................................................................

    do

      ..........................................................................

      ..........................................................................

      ..........................................................................

      ..........................................................................

    end
```

```
add (v: G)
        -- Add 'v' to the current set.
    require

        ...............................................................................

        ...............................................................................

    do

        ...............................................................................

        ...............................................................................

        ...............................................................................

        ...............................................................................

    ensure

        ...............................................................................

        ...............................................................................

        ...............................................................................

    end

remove (v: G)
        -- Remove 'v' from the current set.
    require

        ...............................................................................

        ...............................................................................

    do

        ...............................................................................

        ...............................................................................

        ...............................................................................

        ...............................................................................

    ensure

        ...............................................................................

        ...............................................................................

        ...............................................................................

    end
```

```
  duplicate: like Current
      -- Deep copy of Current.
    do

      create Result.make_empty
      across content as c
      loop
        Result.add (c.item)
      end
    ensure

      same_size: Result.count = count
      same_content: across content as c all Result.has (c.item) end
    end

feature -- Set operations.

  union (another: like Current): like Current
      -- Union product of the current set and 'another' set.
    require

      ..........................................................................

      ..........................................................................

    do

      ..........................................................................

      ..........................................................................

      ..........................................................................

      ..........................................................................

      ..........................................................................

      ..........................................................................

      ..........................................................................

      ..........................................................................

      ..........................................................................

      ..........................................................................

    ensure

      ..........................................................................

      ..........................................................................

    end
```

```
intersection (another: like Current): like Current
    -- Intersection product of the current set and 'another' set.
  require

    ..........................................................................

    ..........................................................................

  do

    ..........................................................................

    ..........................................................................

    ..........................................................................

    ..........................................................................

    ..........................................................................

    ..........................................................................

    ..........................................................................

    ..........................................................................

    ..........................................................................

  ensure

    ..........................................................................

    ..........................................................................

  end

difference (another: like Current): like Current
    -- Set-theoretic difference of the current set and 'another' set.
  require

    ..........................................................................

    ..........................................................................

  do

    ..........................................................................

    ..........................................................................

    ..........................................................................

    ..........................................................................

    ..........................................................................
```

```
        ...........................................................................
        ...........................................................................
        ...........................................................................
        ...........................................................................
    ensure
        ...........................................................................
        ...........................................................................
    end

feature -- Set metrics.

  jaccard_index (another: like Current): REAL_64
      -- Jaccard similarity coefficient between current set and 'another' set.
    require
        ...........................................................................
        ...........................................................................
    do
        ...........................................................................
        ...........................................................................
        ...........................................................................
        ...........................................................................
        ...........................................................................
        ...........................................................................
        ...........................................................................
        ...........................................................................
        ...........................................................................
    ensure
        ...........................................................................
        ...........................................................................
    end
```

```
feature {NONE} -- Implementation

  content: ARRAYED_LIST[G]
      -- Items of the set.

invariant

  content_exists: content /= Void
  content_object_comparison: content.object_comparison
  non_negative_cardinality: count >= 0

end
```