UNIVERSITETET I BERGEN

KANDIDAT

205

PRØVE

INF222 0 Programmeringsspråk

Emnekode	INF222
Vurderingsform	Skriftlig eksamen
Starttid	23.05.2023 07:00
Sluttid	23.05.2023 10:00
Sensurfrist	--
PDF opprettet	03.05.2024 11:02

Introduction

Oppgave	Tittel	Oppgavetype
i	Exam advice	Informasjon eller ressurser
i	Background – some geometrical objects	Informasjon eller ressurser

Assertions 15% – about 30 minutes

Oppgave	Tittel	Oppgavetype
1	The assertion mechanism 5%	Langsvar
2	Geometry assertions 10%	Programmering

Procedures 25% – about 50 minutes

Oppgave	Tittel	Oppgavetype
i	Background - procedures	Informasjon eller ressurser
3	Parameter passing semantics 10%	Langsvar
4	Passing three distinct variables 5%	Langsvar
5	Passing the same variable 5%	Langsvar
6	Parameter passing vulnerabilities 5%	Langsvar

Mars Agriculture Planning 35% – about 70 minutes

Oppgave	Tittel	Oppgavetype
i	Background - Mars Agriculture Planning	Informasjon eller ressurser
7	Sum-of-products 5%	Programmering

8	Sum-of-products in object-oriented languages 5%	Programmering
9	Geometric objects as an AST in Haskell 5%	Programmering
10	Geometric objects as an AST in OO – case-based methods 5%	Programmering
11	Geometric objects as an AST in OO – overriding methods 5%	Programmering
12	Language extension: another geometric figure 10%	Programmering

Seksjon 5

Oppgave	Tittel	Oppgavetype
i	Multiway Dataflow Constraint Systems (MDCS) 15% – about 30 minutes	Informasjon eller ressurser
13	Computing the geometric relationships for the figures 5%	Programmering
14	Computing with the geometry system 10%	Langsvar

Results from compulsory assignments 10% - 0 minutes

Oppgave	Tittel	Oppgavetype
15	Results from compulsory assignments	Muntlig

1 The assertion mechanism 5%

1. Explain what an assertion mechanism in a programming language is.
2. Discuss how assertions can be used in software development for algorithms and APIs (signatures, e.g., for abstract data types or classes).

Fill in your answer here

1: an assert is a type of test where we can check that certain assumptions are held. this can be that a function gave the right output, that a given variable is of the correct type etc..

2: we can use assertions in APIs to make sure that only the correct types where inputted, by checking the input type vs the expected input type. we can also assert parameters for the input like for example that an int must be greater then 0. for an algorithm we can assert that the expected output for a given input was returned.

Ord: 99

2 Geometry assertions 10%

Consider the geometric figures (circle, square, equilateral triangle) from the background.

Task:

1. Write object language code (e.g., PIPL, Java or Python pseudocode) asserting the properties (relationships between the parameters) for each of these geometric figures. Name the procedures circleProperty, squareProperty, triangleProperty, respectively.

Fill in your answer here

```


public bool squareProperty(float s1, float s2, float s3, float s4, float a, float p)
    assertTrue(s1 == s2 == s3 == s4);
    assertEquals((s1 + s2 + s3 + s4), p);
    assertEquals(a, s1 * s2);

    // the asserts will throw an error if they are not true.
    //if we instead want to return false we can run it in a try catch.
    //or just check it our selfs
    return true;
}

public bool circleProperty(float r, float a, float p){
    assertEquals((π * r2), a);
    assertEquals((2 * π * r), p);
    return true;
}

public bool triangleProperty(float s1, float s2, float s3, float a, float p){
    assertEquals((sqrt(3)/4 * s * s), a);
    assertEquals(3 * s, p);
    return true;
}
```

3 Parameter passing semantics 10%

1. Define the semantics of passing arguments by *reference*, *copy-in, copy-out* and *copy-in/out*.
2. Explain which of these can be used with which parameter passing mode (**obs** **out** **upd**).

Fill in your answer here

1:

passing by reference means you pass in a pointer that points to the original value, you can therefore overwrite the original value.

2:

in copy-in you instead copy the value to a new place in memory and pass a pointer to that new address. the original value will not be overwritten.

in copy-out you get a pointer to the original value as input, and can change its content inside the function. but the return value will be a new pointer / value that will over write the original.

copy-in/out means that you get a copy of the value as input, and return a new copy as output.

Ord: 108

4 Passing three distinct variables 5%

Here we call the two procedures with 3 distinct variables.

```
procedure use3p1 (out x:Real, out y:Real, out z:Real ) {
  x := 1;
  y := 1;
  z := 1;
  call p1 ( x, y, z );
}
procedure use3p2 (out x:Real, out y:Real, out z:Real ) {
  x := 1;
  y := 1;
  z := 1;
  call p2 ( x, y, z );
}
```

Tasks:

1. Using *reference semantics*: what are the values of the three variables after calling use3p1 and after calling use3p2 .
2. Using *copy-in/out semantics*: what are the values of the three variables after calling use3p1 and after calling use3p2 .

Fill in your answer here

1:
use3p1: x=3, y=1, z=1
use3p2: x=4, y=1, z=1

2:
use3p1: x=3, y=1, z=1
use3p2: x=4, y=1, z=1

Ord: 18

5 Passing the same variable 5%

Now we call the two procedures with the same variable in all three argument positions.

```
procedure use1p1 (out x:Real ) {
  x := 1;
  call p1 ( x, x, x );
}
procedure use1p2 (out x:Real ) {
  x := 1;
  call p2 ( x, x, x );
}
```

Tasks:

1. Using *reference semantics*: what are the values of the three variables after calling use1p1 and after calling use1p2 .
2. Using *copy-in/out semantics*: what are the values of the three variables after calling use1p1 and after calling use1p2 .

Fill in your answer here

1:
use1p1: x=24
use1p2: x=(16*16)

2:
use1p1: x=1
use1p2: x=1

Ord: 10

6 Parameter passing vulnerabilities 5%

1. Discuss the pros and cons of the various parameter passing semantics. Which semantics do you consider the “best”, considering both efficiency and vulnerabilities?
2. Formulate guidelines for calling procedures that can eliminate the parameter passing vulnerabilities.

Fill in your answer here

reference:

pro: you do not need to copy the value when passing it in.

con: it is easier to cause a bug where you overwrite a value that has larger implications outside the function you are writing. so you need more experience to work with pointers.

it is not possible, or at least hard to write pure functions, because any value you update inside the function will also update it outside the function.

copy in/out:

pro: functions are more encapsulated and you can use this to write pure functions that have no side effects.

it is also easier to intuitivly understand what is happening.

con: copying large data can be slow.

Ord: 111

7 Sum-of-products 5%

Tasks:

1. **Explain the sum-of-products data structure.**
2. **Code MBTL in Haskell as an example of an abstract syntax tree (AST) with a sum-of-products structure.**

The minimal BTL (MBTL) is an expressions only language in the style of BTL. Its expressions are the constant π, the multiplication of two expressions, and taking the square root of an expression.

Fill in your answer here

```
1: the parent of the leaf nodes multiply the leaf nodes together, and the other node
    parents.

2:
data MBLT = Lit i
    | Add MBLT MBLT
    | Mult MBLT MBLT

eval :: MBLT -> Integer
eval (Lit t) = i
eval (Sqrt a) = sqrt $ eval a
eval (Mult a b) = eval a * eval b


ast = sum (Mult (Lit 3) (Lit 4)) (sum (Mult (Lit 31) (Lit 54)) (Mult (Lit 5) (Lit 8)
```

8 Sum-of-products in object-oriented languages 5%

1. Explain how to use the subclass mechanism in object-oriented (OO) languages to encode sum-of-products.
2. Code the MBTL AST using the subclass mechanism in Java.

Fill in your answer here

```
1: each class should only add together the result calculated by its two children.
the children are resposable for calculating the product.

class A {
    A corner = new Corner();
    A fence = new Fence();

    public int totalCost(int fenceLength, cornerLength){
        retrun fence.getCost(fenceLength) + corner.getCost(cornerLength);
    }
}

class Corner extends A {
    int cost 9000;

    public int getCost(int length){
        return cost * length;
    }
}

class Fence extends A {
    int cost 1000;

    public int getCost(int length){
        return cost * length;
    }
}
```

9 Geometric objects as an AST in Haskell 5%

Tasks:

1. Implement a Haskell sum-of-products data type (AST) Land for keeping track of the data for a circle, a square, and an equilateral triangle.
2. Implement a Haskell function (interpreter) fence :: Land −> MKR which computes the cost of a fence along the perimeter of land of each selected geometrical form.

Here **type** MKR = Double;

Fill in your answer here

```haskell


data Land = square (size i)
    | circle (size i)
    | triangle (size i)
    | corner (size i)
    | line (size i)
    | size i
    | mult (size i) (size i)
    | modulo (size i) (size i)

fence :: Land -> MKR
fence (size i) = i
fence (line s) = 1000 * fence s
fence (corner s) = 9000 * fence s

fence (square s) = fence (corner (size 4)) + fence (line s * (size 4))
fence (circle s) = fence (corner (fence (modulo s (size 15))))
                    + fence ((mult (line s)  (size 4)))
fence (triangle s) = fence ((corner (size 3)) + fence (line s )

fence (mult (size s1) (size s2)) = fence s1 * fence s2
fence (modulo (size s1) (size s2)) = fence s1 %  fence s2

```

10 Geometric objects as an AST in OO – case-based methods 5%

For OO we have two strategies for implementing the fence interpreter. First we use Java's **instanceof** , or its equivalent in other OO languages, to case on each of the relevant subclasses.
This is very similar to the Haskell version, which uses pattern matching for each case.

Tasks:

1. Implement an OO sum-of-products data type (AST) CBLand for keeping track of the data for a circle, a square, and an equilateral triangle.
2. Implement a case-based method (interpreter) **public static double** fence (CBLand p) which computes the cost of a fence along the perimeter of land of each selected geometrical form.

Fill in your answer here

```
class CBLand {
    public static double fence(CBLand p){
        if (p instanceOf circle) {
            return p.corners * 9000 + p.size * 1000
        }
        if (p instanceOf square) {
            return 4 * 9000 + p.size * 1000
        }
        if (p instanceOf triangle) {
            return 3 * 9000 + p.size * 1000
        }
    }
}

class circle extends CBLand {
    int size;
    public circle (int size){
        this.size = size;
    }
}
class square extends CBLand {
    int size;
    public triangle (int size){
        this.size = size;
    }
}
class triangle extends CBLand {

    int size;
    public triangle (int size){
        this.size = size;
    }
}
```

11 Geometric objects as an AST in OO – overriding methods 5%

Here we investigate overriding abstract methods in each of the relevant subclasses.

Tasks:

1. Implement an OO sum-of-products data type (AST) OMLand for keeping track of the data for a circle, a square, or an equilateral triangle.
2. Declare an abstract method **public abstract double** fence () in the superclass OMLand .
3. In each of the subclasses, implement an overriding method fence which computes the cost of a fence along the perimeter of land for that geometrical form.

Fill in your answer here

```
class CBLand {
    public abstract double fence ();
}

class circle extends CBLand {
    public double fence(double size){
        int corners = size % 15;
        return corners * 9000 + size * 1000
    }
}
class square extends CBLand {
     public double fence(double size){
        int corners = 4;
        return corners * 9000 + size * 1000 * 4
    }
}
class triangle extends CBLand {
     public double fence(double size){
        int corners = 3;
        return corners * 9000 + size * 1000 * 4
    }
}
```

12 Language extension: another geometric figure 10%

MPT has noticed that regular hexagons, as constructed by bees for millions of years, may be a very efficient land structure. A regular hexagon has six corners, six sides of the same length s, area $a = \frac{3\sqrt{3}}{2}s^2$ and perimeter p = 6 $*$ s. It can be thought of as 6 equilateral triangles all packed together.

Tasks:

1. Case-based OO: Add the new subclass for the regular hexagon, and update the case-based fence method.
2. Override OO: Add the new subclass for the regular hexagon, and provide the override for the fence method.
3. Discuss which of these two OO approaches (case-based versus overriding) is safest for adding a new geometric figure.

Assume a setting with a large OO based system, thousands of classes in complex inheritance hierarchies, spread over many files and a team of more than a dozen software developers evolving the system and fixing bugs. In your discussion take into account support from IDEs, mistakes such as not covering all cases, etc.

Fill in your answer here

13 Computing the geometric relationships for the figures 5%

Tasks:

1. Implement PIPL procedures for computing the parameters for a circle: procedures circle_r2ap (from radius to area and perimeter) circle_a2rp (from area to radius and perimeter) circle_p2ra (perimeter to radius and area).
2. Write the declarations for the PIPL procedures that compute the parameters for a square: procedures square_s2ap (from side to area and perimeter) square_a2sp (from area to side and perimeter) square_p2sa (perimeter to side and area).
3. Write the declarations for the PIPL procedures that compute the parameters for an equilateral triangle: procedures triangle_s2ap (from side to area and perimeter) triangle_a2sp (from area to side and perimeter) triangle_p2sa (perimeter to side and area).

Be careful to get the parameter passing modes correct, and make certain to call the related property procedure to check the computations before returning.

Fill in your answer here

```
1:
procedure circle_r2ap(obs r:Real, out a:Real, out p:Real){
    a := pi * r * r
    p := 2 * pi * 2
    call circleProperty(r, a, p)
}

procedure circle_a2rp(out r:Real, obs a:Real, out p:Real){
    r := sqrt (a / pi)
    p := 2 * pi * r
    call circleProperty(r, a, p)
}

procedure circle_p2ra(out r:Real, out a:Real, obs p:Real){
    r := p / (2 * pi)
    a := pi * r * r
    call circleProperty(r, a, p)
}

2:

procedure square_s2ap(obs s:Real, out a:Real, out p:Real){
    a := s * s
    p := s * 4
    call squareProperty(s, a, p)
}

procedure square_a2sp(out s:Real, obs a:Real, out p:Real){
    s := sqrt a
    p := s * 4
    call squareProperty(s, a, p)
}

procedure square_p2sa(out s:Real, out a:Real, obs p:Real){
    s := p / 4
    a := s * s
    call squareProperty(s, a, p)
}

procedure triangle_s2ap(obs s:Real, out a:Real, out p:Real){
    a := s * s / 2
    p := s * 3
    call triangleProperty(s, a, p)
}

procedure triangle_a2sp(out s:Real, obs a:Real, out p:Real){
    s := sqrt (a * 2)
    p := s * 3
    call triangleProperty(s, a, p)
}

procedure triangle_p2sa(out s:Real, out a:Real, obs p:Real){
    s := p / 3
    a := s * s / 2
    call triangleProperty(s, a, p)
}

```

14 Computing with the geometry system 10%

Tasks:

1. Write the sequence of procedures calls (with their arguments) needed to consistently update the system when ca receives a new value.
2. Write the sequence of procedures calls (with their arguments) needed to consistently update the system when ts receives a new value.
3. Explain how MDCS systems solve such tasks for every allowable global system variable update.
4. Discuss the pros and the cons of using MDCS code for such problems.

Fill in your answer here

```
1: procedure updateFromCA(obs ca:Real, out cr:Real, out cp:Real,
   out ss:Real, out sa:Real, out sp:Real
   out ts:Real, out ta:Real, out tp:Real
){
   assign cr := 0
   assign cp := 0
   call circle_a2rp(ca, cr, cp)

   assign ss := 0
   assign sa := 0
   assign sp := cp
   square_p2sa(sp, ss, sa)

   assign ts := 0
   assign ta := 0
   assign tp := cp
   triangle_p2sa(tp, ts, ta)
}

2:
procedure updateFromTS(out ca:Real, out cr:Real, out cp:Real,
   out ss:Real, out sa:Real, out sp:Real
   obs ts:Real, out ta:Real, out tp:Real
){

   assign ta := 0
   assign tp := cp
   triangle_p2sa(tp, ts, ta)

   assign cs := 0
   assign cr := 0
   assign cp := tp
   call circle_a2rp(ca, cr, cp)

   assign ss := 0
   assign sa := 0
```

```
    assign sp := tp
    square_p2sa(sp, ss, sa)
}
```

3: for every shape, we know how to calculate the other variables of that shape if we get one value. and every shape is connected to at least one other shape by one variable they have in common. so for any given variable that is updated, we just have to calculate the rest of the values of that shape. and then propagate the new state to the rest of the shapes by what ever shared values we have.

4:
pros:
-there is relatively little code we have to write. we just have to write the conventions for any variable to the others in each shape. then the system can calculate all shapes.
-it is easily extendable and modular. adding new shapes entails just writing the functions to calculate all variables from some input. and then to plug it into the system we just have to define its relation to any of the other shapes.

cons:
-if the system is a bit more complex, and we have functions for calculating all variables from all other combinations of variabels. then we might end up with an update that can not propegate to the rest of the system. finding out how many / which convertions we need to write can be difficult. (by convertions i mean triangle_p2sa etc..)
-it can be slow since we might have to update a shape multiple times as other shapes gets updated.
-it is possible to set up an unsolvable system, where the program will run forever.

Ord: 377

15 Results from compulsory assignments

Your points for A5C A6C A8C will automatically be entered here.