UNIVERSITÀ DEGLI STUDI DELL'INSUBRIA

Dipartimento di Scienze Teoriche e Applicate

Corso di Laurea in Informatica

Insegnamento di Progettazione del Software

Prof. Sandro Morasca

ESERCIZI SU TESTING E ANALISI DEL SOFTWARE

For each of the following methods

- determine all Definition-Use-Annulment expressions
- build (if possible) three sets of test data in such a way as to cover
  - all statements, but not all branches
  - all branches, but not all conditions
  - all conditions

EXERCISE 1

```
public static int factorial( int n )
{
        int f = 1;

        for(int i = 1; i <=n; i++)
        {
                f = f*i;
        }
        return f;
}
```

EXERCISE 2

```
public static int fibonacci( int threshold )
{
        int first = 0, second = 1;

        while ( second < threshold )
        {
                int sum = first + second;
                first = second;
                second = sum;
        }
        return first;
}
```

EXERCISE 3

```java
public static boolean isSorted( int [] array )
{
	int i = 0;
	boolean check = true;
	while( i < array.length-1 )
	{
		if( array[i] > array[i+1] )
		{
			check = false;
		}
		i++;
	}
	return check;
}
```

EXERCISE 4

```java
public static boolean isSorted( int [] array )
{
	for(int i = 0; i < array.length-1; i++)
	{
		if( array[i] > array[i+1] )
		{
			return false;
		}
	}
	return true;
}
```

EXERCISE 5

```java
public static void sort( int array[] ) {
    int i, j, temp;
    for(i = 1; i < array.length; i++) {
        for(j = 0; j < array.length-i; j++) {
            if(array[j] > array[j+1]) {
              temp = array[j];
              array[j] = array[j+1];
              array[j+1] = temp;
            }
        }
    }
}
```

EXERCISE 6

```java
public static int makeNotDecreasing( int number )
{
	int previousDigit = 9;
```

```
        int powerOfTen = 1;
        int result = 0;

        while( number != 0 )
        {
                int currentDigit = number % 10;
                if( currentDigit >= previousDigit )
                {
                        currentDigit = previousDigit;
                }

                result = result + currentDigit*powerOfTen;
                powerOfTen = powerOfTen*10;
                previousDigit = currentDigit;
                number = number/10;
        }

        return result;
}
```

EXERCISE 7

```
public static boolean isIncreasing( int number )
{

        int previousDigit = 9;

        while( number != 0 )
        {
                int currentDigit = number % 10;
                if( currentDigit > previousDigit )
                {
                        return false;
                }
                previousDigit = currentDigit;
                number = number/10;
        }

        return true;
}
```

EXERCISE 8

```
public static int method( int x, int y, int z )
{
  int a = 2;

  if( a < x && a > y )
  {
    x = x - a;
    z = x + y;

    while ( a + y < x )
    {
```

```
    a = z + y;
    }
  }
  return( ++a );
}
```

EXERCISE 9

```java
public static int arrayProducts( int array[] )
{
    int product = 1;
    for( int i = 0; i < (array.length+1)/2; i++ )
    {
        int currentSum = array[i]+array[array.length-1-i];
        if( currentSum != 0 && currentSum%4 != 1 )
        {
            product *= currentSum;
        }
    }
    return product;
}
```

EXERCISE 10 (Final Exam 2021-12-20)

```java
/*
 * Metodo che conta il numero di elementi nella prima metà di 'array'
 * il cui valore è maggiore della media degli elementi
 * di 'array' incrementata di 'upperDistance'
 */
public static int outlierCount( double array[], double upperDistance )
{
    double avg = 0;
    int nOutliers = 0;
    for(int i = 0; i < array.length; i++)
    {
      avg += array[i];
    }
    avg /= array.length;
    for(int j = 0; j < array.length/2; j++)
    {
      if( array[j] > avg + upperDistance )
      {
            nOutliers++;
      }
    }
    return nOutliers;
}
```

EXERCISE 11 (Exam 2022-01-10)

```java
/*
 * Metodo che trova il secondo valore più grande in 'array'
 */
public static int secondToMax( int array[] )
{
    int max = array[0];
    if( array.length == 1 )
    {
      return max;
    }

    int secondMax  = array[1];
    if( secondMax > max )
    {
      max = array[1];
      secondMax = array[0];
    }
    for(int i = 2; i < array.length; i++)
    {
        if( array[i] > secondMax )
        {
            if( array[i] > max )
            {
                  secondMax = max;
                  max = array[i];
            }
            else
            {
                  secondMax = array[i];
            }
        }
    }
    return secondMax;
}
```

EXERCISE 12 (Exam 2022-01-31)

```
/*
 * Metodo che conta il numero di elementi (firstHalf) nella prima metà
 * di 'array' il cui valore è divisibile per l'elemento
 * simmetrico nella seconda metà
 * e il numero di elementi (secondHalf) nella seconda metà di 'array'
 * il cui valore è divisibile per l'elemento simmetrico nella prima
 * e restituisce secondHalf se firstHalf > secondHalf oppure
 * secondHalf - firstHalf
 */
public static int divisiblePairs( int array[] )
{
    int firstHalf = 0;
    int secondHalf = 0;
    for(int i = 0; i < array.length/2; i++)
    {
      if( array[i]%array[array.length-1-i] == 0 )
      {
            firstHalf++;
      }
      else if( array[array.length-1-i]%array[i] == 0 )
      {
            secondHalf++;
      }
    }
    return secondHalf - firstHalf;
}
```

EXERCISE 13 (Exam 2022-02-16)

```java
/*
 * Metodo che conta i numeri pari positivi contenuti in 'array'
 * fino alla posizione 'index'
 */
public static int method( int[] array, int index )
{
	int ret=0;

	if( index>=array.length )
	{
		return ret;
	}
	else
	{
		int i=0;
		while( i<=index )
		{
			if( array[i] > 0 & array[i]%2==0 )
			{
				ret++;
			}
			i++;
		}
		return ret;
	}
}
```

EXERCISE 14 (Mid-term Exam 2022-11-09)

```java
    /*
       * Mid-term Exam 2022-11-09
       * Metodo che calcola una somma pesata degli elementi positivi in 'array'.
       * Se un elemento è seguito da un elemento di valore minore, l'elemento viene
       * moltiplicato per un peso e sommato. Il peso viene poi aggiornato.
Altrimenti
       * l'elemento viene sottratto e il peso viene aggiornato.
       */
      public static int sumIncreasingDecreasingWithWeights( int[] array )
      {
            int sum=0;
            int weight = 1;

            for(int i = 0; i < array.length-1; i++)
            {
                  if( array[i] > array[i+1] && array[i] > 0)
                  {
                        sum = sum + weight*array[i];
                        weight = weight+array[i+1];
                  }
                  else
                  {
                        sum = sum - array[i];
                        weight = weight+array[i];
                  }
            }

            return sum;
      }
```

EXERCISE 15 (Mid-term Exam 2022-11-09)

```
/*
 * Mid-term Exam 2022-11-09
 * Metodo che scambia un elemento di 'array' se il suo prodotto
 * con l'elemento simmetrico a partire dal fondo è negativo e conta
 * il numero di scambi avvenuti.
 * Il metodo esamina gli elementi di 'array' solo fino al
 * primo elemento nullo incontrato.
 */
public static int swapNegativeProducts( int array[] )
{
    int count = 0;
    int i = 0;
    while( i < array.length/2 & array[i] != 0 )
    {
      if( array[i]*array[array.length-1-i] < 0 )
      {
            int temp = array[i];
            array[i] = array[array.length-1-i];
            array[array.length-1-i] = temp;
            count++;
      }
      i++;
    }
    return count;
}
```

Please note that black-box test data are shown in some of the solutions for illustration completeness. Black-box test data generation is NOT a part of the written exam exercises, though it may be the topic for questions.

EXERCISE 1: SOLUTION

Definition-Use-Annulment expressions

```
n: duu*
f: d(ud)*u
i: du(uudu)*
```

White-box test data

Input	n	Coverage
A	10	All statements and branches and conditions covered

Sets of inputs

1) all statements, but not all branches: impossible
2) all branches, but not all conditions: impossible
3) all statements and all branches and all conditions: {A}

Black-box test data and goals

Input	n	Goals
A	10	Regular case. Exactly n = 10 loop iterations.
B	1	Boundary case. Exactly n = 1 loop iteration.
C	0	Boundary case. Exactly n = 0 loop iterations
D	-1	????

EXERCISE 2: SOLUTION

Definition-Use-Annulment expressions

```
threshold: duu*
first: d(ud)*u
second: du(uudu)*
sum: (du)*
```

White-box test data

Input	n	Coverage
A	10	All statements and branches and conditions covered

Sets of inputs

1) all statements, but not all branches: impossible
2) all branches, but not all conditions: impossible
3) all statements and all branches and all conditions: {A}

Black-box test data and goals

Input	threshold	Goals
A	10	Regular case. Exactly n = ? loop iterations?
B	1	Boundary case. Exactly n = 0 loop iteration.
C	0	Boundary case. Exactly n = 0 loop iterations.
D	2	Exactly n = 1 loop iteration.
E	13	Case in which threshold is a Fibonacci number.
D	-1	????

EXERCISE 3: SOLUTION

Definition-Use-Annulment expressions

```
array: du(uuu)*
i: du(uuudu)*
check: d((d+ε))*u <=> dd*u
```

White-box test data

Input	array	Coverage
A	{5,1}	All statements covered
B	{1,5,3}	"if-else" branch covered

Sets of inputs

1) all statements, but not all branches: {A}
2) all branches, but not all conditions: impossible
3) all statements and all branches and all conditions: {B}

Black-box test data and goals

Try several cases:

- perfectly sorted array
- unsorted array
- pairs of equal values
- array sorted except for the last pair
- array sorted except for the first pair
- …

EXERCISE 4: SOLUTION

Definition-Use-Annulment expressions

```
array: du(uuu)*
i: du(uuudu)*
```

White-box test data

Input	array	Coverage
A	{1}	return true; covered
B	{1,5,3}	All conditions and, therefore, all branches and, therefore, all statements of the for loop covered

Sets of inputs

1) all statements, but not all branches: impossible (there is no way to execute the i++ statement in the for loop heading without executing the “if-else” branch)
2) all branches, but not all conditions: impossible
3) all statements and all branches and all conditions: {A,B}

Black-box test data and goals

Try several cases:

- perfectly sorted array
- unsorted array
- pairs of equal values
- array sorted except for the last pair
- array sorted except for the first pair
- …

EXERCISE 5: SOLUTION

Definition-Use-Annulment expressions

```
array: du(u(uu(uudd+ε)u)*u)*
i: adu(uu*udu)*
j: a(du(uu(uuuu+ε)udu)*)*
temp: a(((du+ε))*)*
```

White-box test data

Input	array	Coverage
A	{5,1}	All statements covered
B	{1,5,3}	All branches covered

Sets of inputs

1) all statements, but not all branches: {A}
2) all branches, but not all conditions: impossible
3) all statements and all branches and all conditions: {B}

Black-box test data and goals

Try several cases:

- perfectly sorted array
- unsorted array
- pairs of equal values
- array sorted except for the last pair
- array sorted except for the first pair
- …

EXERCISE 6: SOLUTION

Definition-Use-Annulment expressions

```
number: du(uudu)*
previousDigit: d(u(u+ε)d)*
powerOfTen: d(uud)*
result: d(ud)*u
currentDigit: (du(d+ε)uu)*
```

White-box test data

Input	Number	Coverage
A	9	All statements covered
B	53	All branches covered

Sets of inputs

1) all statements, but not all branches: {A}
2) all branches, but not all conditions: impossible
3) all statements and all branches and all conditions: {B}

Black-box test data and goals

Try several cases:

- totally "decreasing" number
- totally "increasing" number
- number in which all digits are the same
- number with only one digit
- number 9
- number 0
- …
- how about negative numbers?

EXERCISE 7: SOLUTION

Definition-Use-Annulment expressions

```
number: du(uudu)*(ε+u)
previousDigit: d(ud)* (ε+u)
currentDigit: (duu)*(ε+du)
```

White-box test data

Input	Number	Coverage
A	345	All statements covered, except `return false;`
B	543	All statements covered, except `return true;`

Sets of inputs

1) all statements, but not all branches: impossible
2) all branches, but not all conditions: impossible
3) all statements and all branches and all conditions: {A,B}

Black-box test data and goals

Try several cases:

- totally "decreasing" number
- totally "increasing" number
- number in which all digits are the same
- number with only one digit
- number 9
- number 0
- …
- how about negative numbers?

EXERCISE 8: SOLUTION

Definition-Use-Annulment expressions

Without taking into account lazy evaluation (considering `&&` and `||` as if they were `&` and `|` instead)

```
x: du(uduuu*+ε)
y: du(uu(uu)*+ε)
z: d(du*+ε)
a: duu(uu(du)*+ε)udu
```

Taking into account lazy evaluation

```
x: du(uduuu*+ε)
y: d(ε+u(uu(uu)*+ε))
z: d(du*+ε)
a: du(ε+u(uu(du)*+ε))udu
```

White-box test data

Input	x	y	z	Coverage
A	5	0	0	All statements covered
B	0	0	0	"else" branch covered
C	5	3	0	a < y made false

Sets of inputs

1) all statements, but not all branches: {A}
2) all branches, but not all conditions: {A,B}
3) all statements and all branches and all conditions: {A,B,C}

Black-box test data and goals

They should be based on the meaning of the method.

EXERCISE 9: SOLUTION

Definition-Use-Annulment expressions

Without taking into account lazy evaluation (considering `&&` and `||` as if they were `&` and `|` instead)

```
array: du(uuuu)*
product: d((ud+ε))*u <=> d(ud)*u
i: du(uuudu)*
currentSum: (duu(u+ε))*
```

Taking into account lazy evaluation

```
array: du(uuuu)*
product: d((ud+ε))*u <=> d(ud)*u
i: du(uuudu)*
currentSum: (du(ε+u(u+ε))*
```

White-box test data

Input	`array`	Coverage
A	{5,2}	All statements covered
B	{5,3,6,2}	"else" branch covered
C	{5,3,-1,1,6,2}	`currentSum != 0` made false

Sets of inputs

1) all statements, but not all branches: {A}
2) all branches, but not all conditions: {B}
3) all statements and all branches and all conditions: {C}

Black-box test data and goals

Try several cases:

- array with an even number of elements
- array with an odd number of elements
- array with only one element
- array with only zero sums
- array with only sums of the kind 4n+1
- array with only even sums
- array with only sums divisible by 4
- …
- how about negative numbers?

EXERCISE 10: SOLUTION

Definition-Use-Annulment expressions

```
array: du(uu)*uu(uu)*
upperDistance: du*
avg: d(ud)*udu*
nOutliers: d((ud+ε))*u ⇔ d(ud)*u
i: du(uudu)*
j: du(uudu)*
```

White-box test data

Input	array	upperDistance	Coverage
A	{7,6,2,1}	1	All statements covered
B	{7,1,2,6}	1	All branches and conditions covered

Sets of inputs

1) all statements, but not all branches: {A}
2) all branches, but not all conditions: impossible
3) all statements and all branches and all conditions: {B}

EXERCISE 11: SOLUTION

Definition-Use-Annulment expressions

```
array: duu(ε+u(uu+ε)u(u(uu+ε)u)*)
max: d(u+u(u+ε)((u(ud+ε)+ε))*) <==> d(u+u(u+ε)(u(ud+ε))*)
secondMax: ε+du(d+ε)(u(d+ε))*u
i: ε+du(u(uu+ε)udu)*
```

White-box test data

Input	`array`	Coverage
A	{1}	All statements covered down to **`return`** `max;`
B	{4,10,6,14}	All statements covered (except **`return`** `max;`). Covered implicit "else" branch in **`if`**`( array.length == 1 )`; covered then branch in **`if`**`( secondMax > max )`; covered then branch in **`if`**`( array[i] > secondMax )`; covered both branches in **`if`**`( array[i] > max )`
C	{10,4,2}	Covered implicit "else" branch in **`if`**`( secondMax > max )`; covered implicit "else" branch in **`if`**`( array[i] > secondMax )`

Sets of inputs

1) all statements, but not all branches: {A, B}
2) all branches, but not all conditions: impossible
3) all statements and all branches and all conditions: {A, B, C}

EXERCISE 12: SOLUTION

Definition-Use-Annulment expressions

```
array: du(uuu(ε+uuu)u)*
firstHalf: d((ud+ε))*u <==> d(ud)*u
secondHalf: d((ε+(ud+ε)))*u <==> d(ud)*u
i: du(uu(ε+uu)udu)*
```

White-box test data

Input	array	Coverage
A	{8,3,6,4}	All statements covered; implicit "else" branch in `else if( array[array.length-1-i]%array[i] == 0 )` not covered
B	{8,3,5,7,6,4}	All branches covered.

Sets of inputs

1) all statements, but not all branches: {A}
2) all branches, but not all conditions: impossible
3) all statements and all branches and all conditions: {B}

EXERCISE 13: SOLUTION

Definition-Use-Annulment expressions

```
array:      du (ε + (uu)*)
index:      du (ε + (u u*))
ret:        d  (u + ( (ud + ε)* u))
i:          ε + (du (uuudu)*)
```

White-box test data

Input	array	index	Coverage
A	{2,4}	5	Copro il primo then
B	{2,4}	1	Copro il then dentro while, ma non l'else implicito
C	{2,3}	1	Copro l'else implicito nel while
D	{2,3,-1, 2}	3	Copro tutte le condizioni dell'if interno al while, sia come unione delle 2 componenti che come single component logiche

Sets of inputs

1) all statements, but not all branches: {A, B}
2) all branches, but not all conditions: {A, C} (copro il then e l'else dell'if interno, ma non array[i]<0)
3) all statements and all branches and all conditions: {A, D}

EXERCISE 14: SOLUTION

Definition-Use-Annulment expressions

Without taking into account lazy evaluation (considering `&&` and `||` as if they were `&` and `|` instead)

```
array: du(uuu(uu+uu)u)* ⇔ du(uuuuuu)*
sum: d((ud+ud))*u ⇔ d (ud+ud)*u ⇔ d (ud)*u
weight: d ((uud+ud))* ⇔ d (uud+ud)*
i: du(uuu(uu+uu)udu)* ⇔ du(uuuuuudu)*
```

Taking into account lazy evaluation

```
array: du(uu(u(uu+uu)+ε(uu)))* ⇔ du(uu(u(uu)+ε(uu)))* ⇔ du (uu(u+ε)uu)*}
sum: d((ud+ud))*u ⇔ d (ud+ud)*u ⇔ d (ud)*u
weight: d ((uud+ud))* ⇔ d (uud+ud)*
i: du(uu(u(uu+uu)+ε(uu))udu)* ⇔ du(uu(u(uu)+ε(uu))udu)* ⇔ du (uu(u+ε)uuudu)*}
```

White-box test data

Input	array	Coverage
A	{1,3,2}	While loop executed twice: - when comparing array[0] and array [1] the else branch is covered - when comparing array[1] and array [2] the else branch is covered
B	{-1,-2}	Condition array[0] > array [1] is true, but condition array[0] > 0 is false
C	{-1,1}	Condition array[0] > array [1] is false and condition array[0] > 0 is false

Sets of inputs

1) all statements, but not all branches: impossible
2) all branches, but not all conditions: {A} (all branches are covered, but condition array[i] > 0 is always true)
3) all statements and all branches and all conditions: {A,B,C}

EXERCISE 15: SOLUTION

Definition-Use-Annulment expressions

```
array: duu(uuu(uuudud+ε)uu)*
count: d(ud)*u
i: duu(uu(uuuu+ε)uduu)*
temp: ((du))* ⇔ (du)*
```

White-box test data

Input	array	Coverage
A	{1,-1}	While loop is executed once and then branch is executed
B	{1,2,3,-1}	While loop executed twice - the first time the then branch is executed - the second time the "implicit else" branch is executed
C	{1,0,3,-1}	The second time the while loop predicate is evaluated, - condition 1 < array.length/2 is true - condition array[0] != 0 is false
D	{1,3,0,-1}	The third time the while loop predicate is evaluated, - condition 1 < array.length/2 is false - condition array[0] != 0 is false

Sets of inputs

1) all statements, but not all branches: {A} ("implicit else" branch not covered)
2) all branches, but not all conditions: {B} (condition array[i] != 0 is never false)
3) all statements and all branches and all conditions: {B,C,D}