# 1. Practical part

## 1.1. Static analysis of source code

1. **Static analysis**

   The following code snippet is given, where the function `ioread32()` reads a 32 bit signed integer.

```
   int64 a, b
1: a = ioread32()
2: if (a < 0) {
3:     b = -a
4: } else {
5:     b = a
6: }
```

   (a) (2 points) Using abstract interpretation, describe the values of the program variables in each line ($X_1, X_2, \ldots, X_6$)! If no concrete value is known, compute symbolically (using previous values)!

   (b) (2 points) Using abstract interpretation with intervals, propagate the range of the program variables from the entry of the program ($X_1$)! What is the possible range of variable $b$ at line 6 ($X_6$)?

## 1.2. Specification-based testing

2. **Combinatoric testing**

   Our application is available for the iOS, Android and Windows platforms. For Windows it can run also on PC, for the other two only smart phone and tablet is supported. The application is localized to Hungarian and English.

   (a) (1 point) How many different configurations can be tested for the application ______________

   (b) (2 points) We do not need all the possible combinations, pair-wise coverage is enough. Give a set of configurations that is minimal and satisfies the pair-wise criterion.

| # | Platform | Devise | Language |
|---|---|---|---|

3. **Decision tables**

   We are developing an application to calculate car insurance prices. The price is based on engine power (500 Ft / kW). A 5% percent discount is given if the owner has a family or if he or she is public servant (but only one of the discounts can be applied). A further 5% percent is given if the owner did not have an accident in the last year (this discount is applied after the first one). However, after taking into account all discounts, the price cannot be lower than 25 000 Ft.

   (a) (3 points) Draw a decision table to support testing the above specification.

   (b) (1 point) What concrete test cases would you select based on the decision table?

## 1.3. Structure-based testing

4. **Structure-based testing**

Given the following source code fragment:

```
int function13(int a, bool b){
  int c = 10;
  if (b) c = -c;

  for (int i = 0; i < a; i++){
    c++;
  }

  return c;
}
```

(a) (2 points) Draw the control flow graph (CFG) of the function next to the code.

(b) (2 points) Give a test suite that reaches 100% decision coverage.

5. **Structure-based testing**

Given the following function (left) and a set of test inputs to test the function (right)

```
void checkParameters(bool a, int b, int c, bool d){
  if (!a || (b > 0)){
    error();
    return;
  }

  if ( (c == 0) && d && (b < -100) ){
    warning();
    return;
  }

  info();
  return;
}
```

| # | a | b | c | d |
|---|---|---|---|---|
| T1 | false | 0 | 0 | true |
| T2 | true | 0 | 1 | false |

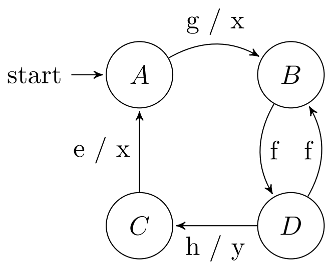
(a) (2 points) What decision coverage do the test achieve? (Give the calculations also.)

(b) (2 points) What condition coverage do the test achieve? (Give the calculations also.)

## 1.4. Test generation

6. **Model-based test generation**

   Given the following finite state machine:




   (a) (3 points) Give a test suite that reaches 100% transition coverage!

7. **Symbolic execution**

   Given the following program:

```
public int Handler(State s, int v) {
  switch(s) {
    case Active:
      return 0;
    case Inactive:
      return -1;
    default:
      v = v + 1;
  }
  if(v > 10) throw new Exception();
  return v;
}
```

   (a) (3 points) Draw the symbolic execution tree of the `Handler` function. Mark the path(s) raising an exception.

   (b) (1 point) What path condition belongs to the path(s) raising an exception?

## 1.5. Regression testing

8. **Regression test selection**

   We have a web shop application with a JavaScript front-end, a cloud-hosted back-end consisting of the inventory management and product recommendation modules, and a database. Unit tests test independently the inventory (UT1) and recommendation modules (UT2). Integration tests use the API of the back-end. The first integration test (IT1) checks a basic purchase, the second one (IT2) checks buying a recommended product. The system test (ST1) is a test driving the web UI.

   Our team performed a modification on the recommendation module.

   (a) (1 point) Give the mappings from tests to tested components.

   (b) (2 points) Classify each test with respect to the given modification.

## 1.6. Dependability analysis

9. **Event trees**

An automated train has two on-board computers. If the main computer fails, then the backup computer takes over the control, and to reach a safe state it tries to stop the train. If the backup computer fails also, then the train switches to manual control, and it is the responsibility of the driver to stop the train. However, it is possible that the manual brake fails also, in this case a dangerous state arises. Moreover, a dangerous state can arise even if the computer or the driver tries to stop the train, but due to failure of the brake subsystem, it is not possible.

The probability of the failure of the main computer is $P_1$, of the backup computer is $P_2$, of the manual brake is $P_3$, and of the brake subsystem is $P_4$. All events are independent.

(a) (3 points) Draw an **event tree** starting from the failure of the main computer (initial state)!

(b) (1 point) Give the probability of arriving in a dangerous state!

10. **Reliability block diagram**

In our distributed system the writing of a data request is the following. The request can arrive through two redundant network switch. First a load balancer forwards the request to one of the three redundant web servers. Next an other load balancer forwards the request to one of the two redundant database servers. The component-level asymptotic availability values ($k$) are the followings: switch $0,95$; load balancer $0,99$; web server $0,9$; database server $0,9$. h $0,95$; terheléselosztó $0,99$; webszerver $0,9$; adatbázis szerver $0,9$.

(a) (3 points) Give the *reliability block diagram* of the whole system's availability.

(b) (1 point) Give the asymptotic availability of the whole systems. How many hours is the system out of service in a year?