1. Gyakorlati rész

1.1. Specifikáció átvizsgálása

1. **Specifikáció ellenőrzése**

   A repülőjegyhez választható extrák árának kiszámításához készülő modulhoz a következő specifikációt kaptuk.

   - „Egy 15 kg-os csomag ára €10, a 20 kg-os ára pedig €20. Egy utas legfeljebb két csomagot hozhat. A reptéren vásárolt csomag felára €15."
   - „Ülések foglalásának díja €8. Prioritásos ülések (1–3 sor) díja €13."
   - „Lehetőség van Leisure csomagot választani (€15), amely tartalmaz egy 15 kg-os csomagot és egy foglalt ülést."

   (a) Átvizsgálással ellenőrizzük a kapott specifikációt. Milyen kérdéseink és észrevételeink lennének? (4 pont)

2. **Specifikáció ellenőrzése**

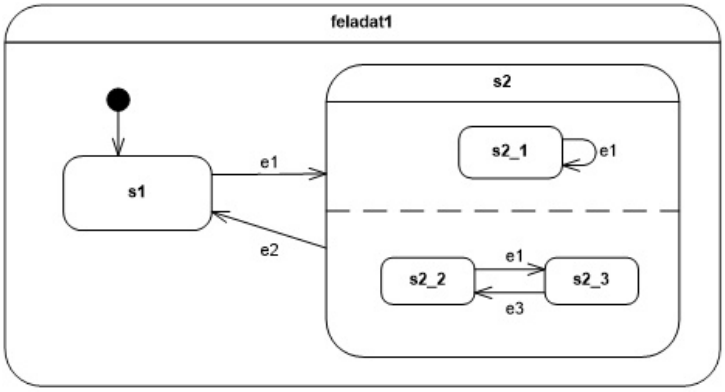
   Rajban repülni képes autonóm drónok fejlesztésén dolgozunk, és jelenleg annak a modulnak a V&V tervét készítjük, ami eldönti, hogy mikor kell egy adott drónnak önállóan vagy a raj vezetőjét követve repülnie. Rajban repülve a drón a raj vezetőjétől kap folyamatosan utasításokat, valamint a drónra szerelt kamerával is tudja követni a vezetőt. A következő követelményeket fogalmazták meg eddig a rendszermérnökök.

   - REQ1: Alapállapotban, ha más parancsot nem kap a drón, akkor önállóan kell repülnie.
   - REQ2: A kezelőtől kapott távoli üzemmódváltó parancsot követni kell.
   - REQ3: Ha a drón rajban repül és elveszti a kommunikációs és vizuális kapcsolatot is a raj vezetőjével, akkor át kell váltania önálló üzemmódba.

   (a) Átvizsgálással ellenőrizzük a kapott specifikációt. Milyen kérdéseink és észrevételeink lennének? (3 pont)

3. **Állapotgépek ellenőrzése**

   Adott a következő UML állapotgép.



(a) Teljes-e az állapotgép specifikációja (az e1, e2, e3 eseményekre nézve)? Ha nem, adjuk meg az összes problémát. (2 pont)

(b) Egyértelmű az állapotátmenetek definíciója? Ha nem, adjuk meg az összes problémát. (2 pont)

1.2. Forráskód átvizsgálása

4. **Forráskód átvizsgálása**

Adott a következő forráskód részlet.

```
float func(float a, int t){
    if (a == 0) return 0;
    switch (t)
    {
    case 0: {
        a *= 0.8;
        break;
    }
     case 1: a *= 0.7;
     case 2: a *= 0.5;
    }
      return a;
}
```

(a) Soroljon fel legalább két problémát a kód stílusával kapcsolatban! (2 pont)

(b) Soroljon fel legalább két olyan problémát, amely potenciális hibalehetőséget rejt magában! (2 pont)

5. **Forráskód átvizsgálása**

Adott a következő forráskód részlet.

```
void writeFile(String text_to_write, String fileName){
    PrintWriter pw = null;
    try{
        pw = new PrintWriter( fileName );
    } catch(Exception e) {
        System.out.println("An exception occured!");
    }
    text_to_write.replace('-','_');
    pw.write(text_to_write);
}
```

(a) Soroljon fel legalább két problémát a kód stílusával kapcsolatban! (2 pont)

(b) Soroljon fel legalább két olyan problémát, amely potenciális hibalehetőséget rejt magában! (2 pont)

1.3. Specifikáció-alapú tesztelés

6. **Specifikáció-alapú tesztelés**

Adott a következő tesztelendő függvény:

```
int functionA(int a, int b)
```

ahol az `a` egy pozitív egész szám, `b`-nek pedig 1 és 10 közé kell esnie.

(a) Mik az egyes paraméterek határértékei, és a határérték analízis technika alapján milyen teszt-bemeneteket választanánk ki ezekhez? (2 pont)

(b) A kiválasztott értékek alapján milyen tesztkészletet állítanánk elő a `functionA` függvényhez? (1 pont)

#	a	b	Elvárt eredmény

7. **Specifikáció-alapú tesztelés**

Készülő alkalmazásunk elérhető iOS, Android és Windows platformokra. Windows esetén PC-ken is, míg a másik két platformon csak okostelefonon és tableten támogatott. Jelenleg magyarra és angolra van lefordítva.

(a) Hány különböző konfigurációban lehetne tesztelni az alkalmazást? _____ (1 pont)

(b) Megelégszünk csak a páronkénti lefedettséggel. Mutasson egy olyan konfigurációhalmazt, ami minimális és kielégíti a páronkénti lefedettség kritériumát! (2 pont)

#	Platform	Eszköz	Nyelv

1.4. Struktúra-alapú tesztelés

8. **Struktúra-alapú tesztelés**

Adott a következő forráskód részlet.

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

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

  return c;
}
```

(a) Rajzoljuk fel a függvény vezérlési folyam gráfját (CFG) a függvény kódja mellé! (2 pont)

(b) Adjunk meg egy tesztesetet, mely 100%-os utasítás lefedettséget garantál! (2 pont)

9. **Struktúra-alapú tesztelés**

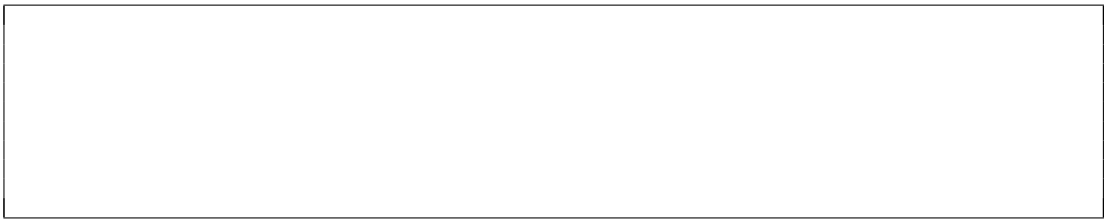
Adott a következő függvény (bal), és egy tesztbemenet-halmaz, amivel teszteljünk (jobb).

```
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) Hány százalékos döntés lefedettséget érnek el a tesztek? (A számítást is kérjük mellékelni.) (2 pont)

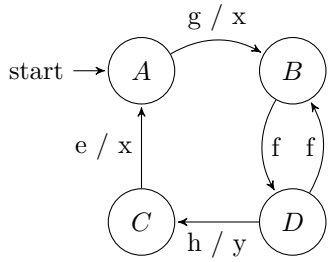
(b) Hány százalékos feltétel lefedettséget érnek el a tesztek? (A számítást is kérjük mellékelni.) (2 pont)



1.5. Tesztgenerálás

10. **Tesztgenerálás**

Adott a következő véges automata:



(a) Adjon egy olyan tesztkészletet, ami 100% átmenet lefedettséget ér el! (3 pont)

11. **Tesztgenerálás**

Adott a következő forráskód részlet.

```
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) Rajzolja fel a `Handler` függvény szimbolikus végrehajtási fáját! Jelölje a kivételt előidéző lefutási útvonala(ka)t a többitől eltérően! (3 pont)

(b) Milyen útvonal-feltétel (path condition) adódik így a kivételt előidéző útvonalon? (1 pont)

1.6. Szolgáltatásbiztonság analízise

12. **Szolgáltatásbiztonság analízise**

Egy automatikus vezérlésű vonaton két fedélzeti számítógép van. Amennyiben a fő számítógép meghibásodik, a tartalék számítógép veszi át az irányítást és a biztonság érdekében megpróbálja lefékezni a vonatot. Ha a tartalék számítógép is hibás, a vonat kézi vezérlésre vált és a vezető feladata lefékezni a vonatot. Előfordulhat azonban, hogy a kézi vezérlő is meghibásodik, ilyenkor nem lehet lefékezni a vonatot és veszélyes helyzet áll elő. Emellett akkor is veszélyes helyzet áll elő, ha a számítógép vagy a vezető megpróbálja ugyan lefékezni a vonatot, de a fékrendszer hibája miatt ez nem sikerül.

A fő számítógép meghibásodásának valószínűsége P_1, a tartalék számítógépé P_2, a kézi vezérlőé P_3 a fékrendszeré pedig P_4. Az események egymástól függetlenek.

(a) Rajzoljon fel **eseményfát** (event tree) a fő számítógép meghibásodásából (kezdeti esemény) kiindulva! (3 pont)

(b) Adja meg a veszélyes helyzet kialakulásának valószínűségét! (1 pont)

13. **Szolgáltatásbiztonság analízise**

Egy elosztott rendszerben egy lokális és két távoli adatbázis van. Amennyiben a lokális adatbázis meghibásodik, az első távoli adatbázishoz kell fordulni. Ha az első távoli adatbázis is hibás, akkor a második távoli adatbázishoz kell fordulni. Előfordulhat azonban, hogy ez is hibás, ilyenkor nem elérhető az adat. Emellett akkor sem érhető el az adat, ha működnek a távoli adatbázisok, de a lokális adatbázis hálózati kapcsolata hibás.

A lokális meghibásodásának valószínűsége P_L, az első tartalék adatbázisé P_{T_1}, a második tartalék adatbázisé P_{T_2} a hálózati kapcsolaté pedig P_H. Az események egymástól függetlenek.

(a) Rajzoljon fel **eseményfát** (event tree) a lokális adatbázis meghibásodásából (kezdeti esemény) kiindulva! (3 pont)

(b) Adja meg annak a valószínűségét, hogy az adat nem érhető el! (1 pont)