# Logikai modulok szimulálása konzolon

## Tartalomjegyzék

# Specifikáció:

Ez a program logikai kapukból álló modulok szimulálására szolgál. Lehet benne létrehozni modulokat. Szimulálni inputok változását, és a program kiírja az inputok és outputok értékeit a szimulálás folyamán. Le lehet menteni egy fájlba, hogy milyen modulokat hozott létre a felhasználó, milyen inputokat adott meg, meddig futtatta a szimulációt, valamint milyen korábbi mentéseket olvasott be.

## Utasítások=inputok:

### modul létrehozása:

forma: <név>:<leírás>

1.1. egyszerű modul:
leírás: <érzékenység><hozzárendelés>,<érzékenység><hozzárendelés>,…
név megadása, minden outputra: inputoktól függésének megadása, ez a függés tartalmazhat érzékenységi listát: csak akkor változzon az output, ha az ebben a listában megadott inputok egyike változott, elhagyás esetén bármelyik a hozzárendelésben szereplő input változásakor lefut a hozzárendelés kiértékelése, és módosulhat az output.

1.2. komplex modul:
leírás: <modulnév>(<inputok>,<outputok>)<modulnév>(<inputok>,<outputok>)…
név megadása, használni kívánt modulok és bekötéseik

### végrehajtás:

forma: <inputok><futtatásszám>< módosítók>
három részből áll melyek mindegyike opcionális, de sorrendjük kötött

1.3. inputok állítása: kis betű: az adott betűvel jelölt _main modulba vezető jel 0-ra (hamis) állítása, nagy betű: az adott betűvel jelölt _main modulba vezető jel 1-re (igaz) állítása. A be nem állított értékek bizonytalan állapotban, ’?’-vel jelölve jelennek meg.

1.4. futtatásszám megadása: egy pozitív integer. ha egy végrehajtás parancsnak van ilyen eleme, akkor ennyiszer fogja lefuttatni az éppen aktuális várakozó modulokat. Mindig, amikor egy modul outputjának értéke változik/ "inputok állítása" történik az adott outputra kötött modulok/_main modul felkerül a várakozó modulok listájára. Lényegében ekkor történik a szimuláció.

1.5. módosítók megadása:
1.5.1.’+’-jel: minden lefutásnál kiírja a _main modul portjainak értékeit,
1.5.2.’-’-jel: bármennyire nagy a lefutások száma, csak a végén írja ki a _main modul portjainak értékeit.
1.5.3.’%’-jel: konzol törlése, csak az átláthatóság érdekében, a háttérben semmit sem töröl, csak a konzolt tisztítja meg.
1.5.4.’!’-jel: kilép a programból.

### fájlból beolvasás:

forma: <<fájlnév+kit>
beolvasandó fájl neve + kiterjesztése

A fájl ugyan olyan parancsokat tartalmazhat, mint amiket a konzolon adunk meg, a fájlba írás kivételével, amit nem tartalmazhat, valamint nem tartalmazhatja önmaga beolvasását, de más fájlokét igen (körkörös beolvasást sem). Ha nem sikerül megnyitni a fájlt kiírja, hogy nem sikerült.

### fájlba írás:

forma: ><fájlnév+kit>
"ki" fájl neve + kiterjesztése.

Minden utasítás, amit kiadtunk/ fájlból hajtódott végre, (fájlba írások kivételével) egyesével megkérdezésre kerülnek, hogy bekerüljenek-e a "ki" fájlba 'I'(nagy i): igen, bármi más: nem.

## Használat:

A fent felsorolt utasításokat lehet kiadni a konzolon, amik végrehajtódnak, vagy kiírja milyen hiba miatt nem hajthatók végre. Hibák kiírásánál használt nyelvezet: text_module: egyszerű modul, comp_module: komplex modul

### modul hibák:

1. nincs : jel

   Minden modul definiálásánál a név után kell egy ':'-jel, hogy honnan kezdődik a modul viselkedésének leírása

2. nem _ val kezdodo nev

   minden modul névnek '_'-vel kell kezdődnie

3. rossz karakter a nevben

   A név a következő karaktereket tartalmazhatja: „!#$%*+-./<=>?@_{}" valamint az angol abc ki és nagybetűit plusz a számokat

4. mar foglalt nev

   Egy nevet csak egy modulnak lehet adni

5. masodik kivezetes comp_module -ban

   Minden kivezetés legyen az output vagy belő vezetékezés, csak egy helyről kaphatja az értékét

6. nem jo bekotes comp_module -ban

   Az inputok az egymás után következő angol kisbetűivel vannak jelölve, valamint az outputok az első nem csak inputként használt kisbetűtől vannak sorban. Nem lehet egy modul inputjai 'a' és 'g' és ugyanakkor outputja 'b'; Ez helyesen 'ab' input és 'c' output.

   Ez a hiba jelentkezik akkor is, ha a nagy betűknél (belső vezetékezésben) van hiba: csak inputként van használva az egyik vezeték, csak outputként van használva (ha nem akarjuk használni egy modul outputját, azt '-' karakterrel jelezhetjük), nincs minden betű felhasználva a legnagyobb betű előtt (ABD van használva, de C nem).

7. nem letezo modul comp_module -ban

   Nem definiált modult próbál használni egy komplex modulban

8. rossz inputszam comp_module-ban

   Az egyik használni kívánt modulnak nem egyezik meg az input száma a '(' és a ',' közé írt karakterek számával

9. rossz outputszam comp_module-ban

Az egyik használni kívánt modulnak nem egyezik meg az output száma a ’,’ és a ’)’ közé írt karakterek számával

10. nem jo bekotes text_module-ban

Az inputok az egymás után következő angol kisbetűivel vannak jelölve, ’abd’ használata rossz mert kimarad a ’c’

11. nem jo forma text_module-ban

Az érzékenységi listában csak inputok(angol kisbetűk) állhatnak, az után viszont érték (input vagy konstans) és művelet csak felváltva állhat. Ez alól kivétel a ’~’ ami állhat érték elé és művelet elé is, de önmaga elé nem. pl.: ~~a hibás, ab&c hibás, de ~a~&~b jó

12. rossz karakterek a text_module-ban

az egyszerű modulban csak ’[]’-jelek (érzékenységi lista megadása), angol abc kisbetűi (inputok), ’()’ zárójelek(műveleti sorrend állítása), ’01’ (konstansok) és ’&|^~ ’ karakterek(és, vagy, kizáró vagy, nem), valamint ’,’-jel (outputok szétválasztása) használhatók.

### További tudnivalók:

Az utasítással definiált modulok csak template-ek, pl.: egy komplex modul tartalmazhat egy adott modulból többet, nem kell hozzá többször definiálni egy-egy modult. Ez alól kivétel a _main modul, ami rögtön definiálás után példányosodik.

A program teljes használatára le van tiltva a space, nem szabad használni se moduloknál se sehol.

Fontos, hogy komplex modul csak olyan modulokat tartalmazhat, amik már definiálva lettek. A _main nevű modul lesz a szimuláció alapja, neki állíthatjuk az inputjait és neki látjuk az outputjait, így tehát végrehajtás utasítást a _main modul létrehozása előtt nincs nagyon értelme kiadni, de lehetséges.

* kiírások még változhatnak, de tartalmilag nem változtatok rajtuk, maximum szóhasználatilag.

## Modul példák:

### egyszerű modul érzékenység nélkül:

_and:a&b

Definiálja a _and nevű modult, hogy az első outputja: az első és második inputján kapott jelek és kapcsolatát adja ki.

_half_adder:a^b,a&b

Definiálja a _half_adder nevű modult, hogy az első outputja: az első és második inputján kapott értékek kizáró vagy kapcsolatát adja ki, és második outputja: az első és második inputján kapott értékek és kapcsolatát adja ki.

### érzékenységgel:

_d_flipflop:[b]a

Definiálja a _d_flipflop nevű modult, hogy az első outputja: felvegye az első inputja értékét, amikor a második input megváltozik (fel és leszálló ágban is változásként van érzékelve).

## komplex modul

tegyük fel, hogy a _and:a&b és a _xor:a^b utasításokat már kiadták ekkor a _half_adder modul így is kinézhet:

_half_adder:_and(ab,d)_xor(ab,c)

ekkor az _and modul első inputjára bekötjük a _half_adder első inputját(a) és második inputjára, a másodikat(b), valamint az első outputjára a _half_adder második outputját(d) az _xor modul inputjait ugyan úgy kötjük be, mint az _and modul esetében, viszont ennek az outputját a _half_adder első outputjára(c) kötjük

valamint a _full_adder, függetlenül attól, hogy hogy lett a _half_adder definiálva:

_full_adder:_half_adder(ab,AB)_half_adder(Ac,dC)_xor(BC,e)

Ekkor az első _half_adder modul első két inputja a _full_adder modul első két inputjára van kötve, outputjai pedig a belső vezetékek (nagy betűvel jelölt kívülről el nem érhető vezetékrendszer, a komplex modulokban) első kettőjére. A második _half_adder modul első inputja a belső vezetékezés első vezetékére van kötve, második bemenete pedig a _full_adder modul harmadik bemenetére. Első kivezetése a _full_adder modul első kivezetésére, második kimenete a belső vezetékezés harmadikjára van kötve. Végül a két carry-t egy or vagy xor kapuval össze kell rakni, erre szolgál a _xor modul a végén, ami a két _half_adder modul carry-ének kizáró vagy kapcsolatát kiadja a _full_adder modul második kivezetésére.

## _main modul

ami egy két 4 bites számot ad össze:

_main:_full_adder(dh0,mA)_full_adder(cgA,lB)_full_adder(bfB,kC)_full_adder(aeC,ji)

összeadja az abcd és efgh inputok értékét és kiadja az eredményt az ijklm outputokon. Az első _full_adder modul harmadik inputján egy konstans 0 van bekötve, mert csak a két számot akarjuk összeadni és nincs kezdeti carry.

## Példa végrehajtás utasításokra:

A fenti _main definiálása esetén

be: abcdefgh

minden inputot 0ra állít, de nem történik lefutás

be: 10-

tíz lefutást hajt végre, de csak a végeredményt írja ki

ki:

```
┌────────┬─────┐
|abcdefgh|ijklm|
|00000000|00000|
└────────┴─────┘
```

be: abCDeFGh10+

ki:

```
┌────────┬─────┐
|abcdefgh|ijklm|
|00110110|00000|
|00110110|00000|
|00110110|00101|
|00110110|00001|
|00110110|00001|
|00110110|01001|
|00110110|01001|
|00110110|01001|
|00110110|01001|
|00110110|01001|
└────────┴─────┘
```

be: ABEH10

ki:

```
┌────────┬─────┐
|abcdefgh|ijklm|
|11111111|01001|
|11111111|01001|
|11111111|11100|
|11111111|10110|
|11111111|11110|
|11111111|11110|
|11111111|11110|
|11111111|11110|
|11111111|11110|
|11111111|11110|
└────────┴─────┘
```

## Példa mentésre

be: >save.txt

```
_and:a&b
I/N: I
_half_adder:a^b,a&b
I/N: I
_xor:a^b
I/N: I
_full_adder:_half_adder(ab,AB)_half_adder(Ac,dC)_xor(BC,e)
I/N: I
_main:_full_adder(dh0,mA)_full_adder(cgA,lB)_full_adder(bfB,kC)_full_adder(aeC,ji)
I/N: I
abcdefgh
I/N: N
10-
I/N: N
abCDeFGh10+
I/N: I
ABEH10
I/N: N
```

ekkor a save.txt tartalma:

```
_and:a&b
_half_adder:a^b,a&b
_xor:a^b
_full_adder:_half_adder(ab,AB)_half_adder(Ac,dC)_xor(BC,e)
_main:_full_adder(dh0,mA)_full_adder(cgA,lB)_full_adder(bfB,kC)_full_adder(aeC,ji)
abCDeFGh10+
```

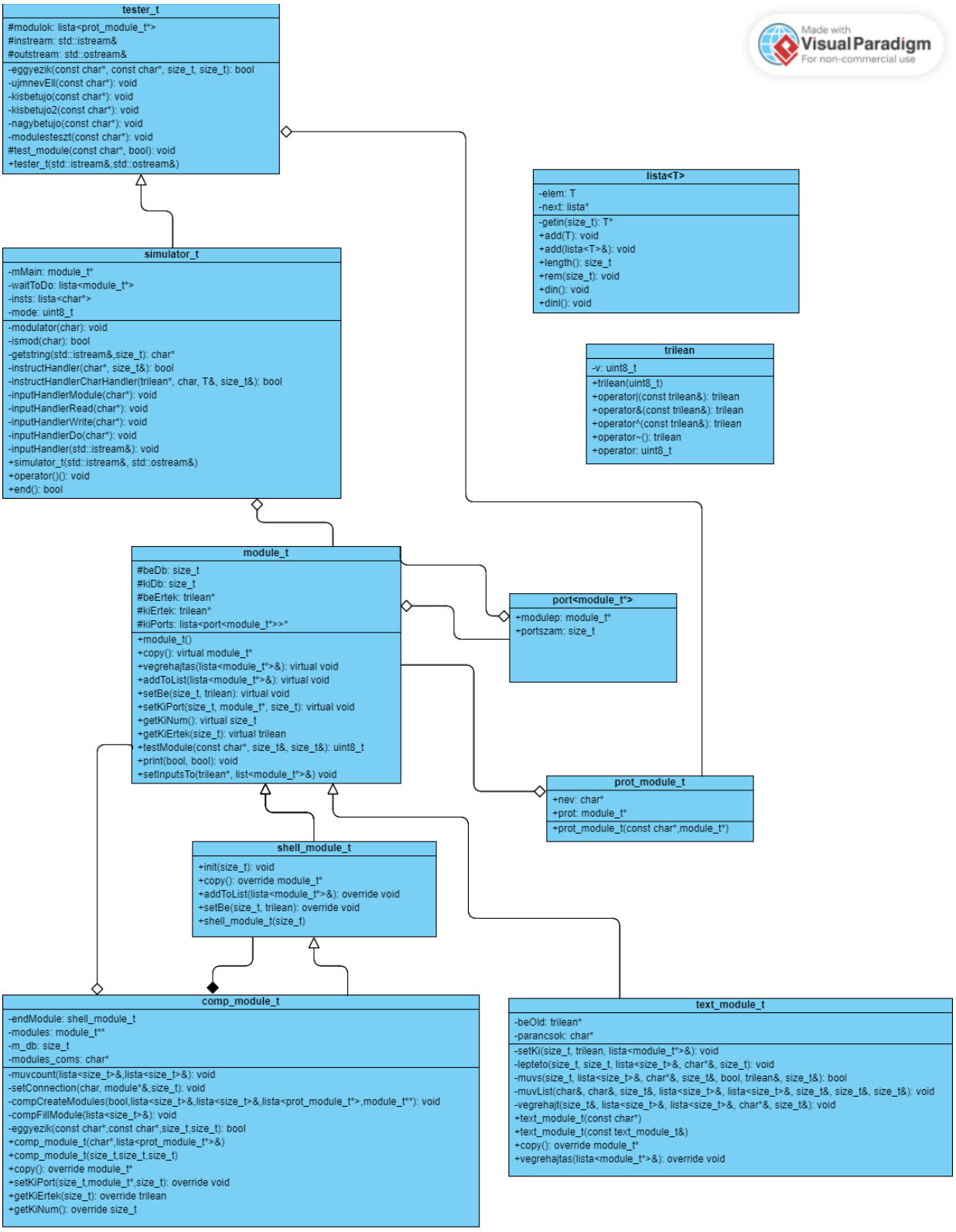
## Példa beolvasásra

amikor megnyitjuk a programot beolvashatjuk a "mentést"

be: <save.txt

```
sikeres megnyitas
main setted
┌────────┬─────┐
|abcdefgh|ijklm|
|00110110|?????|
|00110110|?????|
|00110110|????1|
|00110110|0???1|
|00110110|0?001|
|00110110|0?001|
|00110110|01001|
|00110110|01001|
|00110110|01001|
|00110110|01001|
└────────┴─────┘
```

# Terv:




A trilean és a templates lista a programom alapszókészletét képezik, mint az int. Minden class ismeri ezeket az elemeket. Illetve a lista meg van valósítva a size_t, module_t*, port_t*, prot_module_t* és a char* typeokra.

# Classok és fontosabb(nem maguktól érthetődő) függvényeik:

## 1. lista:

láncolt lista, aminek lezáró elemének next-je nullptr, ebben már nem lesz használt adat.

1.1. din: a listában tárolt elemek törlése
1.2. dinl: a listában tárolt tömbök törlése
1.3. rem: remove i-edik elem

## 2. trilean

ez egy saját változótipus, a booleanhoz hasonló, de előrre definiált low, high és undet értékeket vehet fel. Értelmezve vannak rá az alapvető bináris operátorok(& | ^ ~). És lehet uint8_t ként használni.

## 3. module_t:

tárolja az input és output értékeket, valamint, hogy milyen modulok vannak rákötve melyik kivezetésére

3.1. copy: visszatér egy a modulról készült másolat pointerével
3.2. vegrehajtas: semmit nem csinál
3.3. addToList: hozzáadja magát a paraméterként kapott listához.
3.4. setBe: a megadott indexű inputra beírja a megadott értéket.
3.5. setKiPort: a megadott indexű outputra rákötí a kapott modul kapott indexű bemenetét
3.6. getKiNum: visszatér, hogy hány outputja van a modulnak
3.7. getKiErtek: visszatér az indexedik output értékével
3.8. testModule: leteszteli, hogy a kapott string k és v köztirésze tartozhat e hozzá, megfelelő számú input/output van megadva
3.9. print: kiírja a modul bemeneti és kimeneti értékeit
3.10. setInputsTo: beállítja az inputok értékét a kapott listának megfelelően

## 4. text_module_t:

ez az a modul, ahol számolások is folynak

4.1. végrehajtás: lefutatja az outputokhoz tartozó kiértékeléseket és ha változott az output értéke, akkor a hozzá kapcsolt modulokat hozzáadja a várakozók listájához:
4.1.1. Egy outputhoz tartozó szöveg kimásolása, inputok betűi helyére értékek behelyettesítése.(vegrehajt)
4.1.2. Műveleti lista létrehozása, műveleti sorrenddel.(muvList)
4.1.3. Majd ebben a sorrendben, műveletek elvégzése, szintén segédfüggvényekkel(muvs), a kimásolt szöveg folyamatosan változik, minden végrehajtott művelet helyére az eredmény kerül (lepteto), és így a végén kijön egy 1 karakteres végeredmény.
4.1.4. Ezek után, ha változott az output akkor a hozzá kapcsolt moduloknak megváltoztatja a bemeneti adatait és a modulokat hozzáadja a várakozók listájához(setKi)

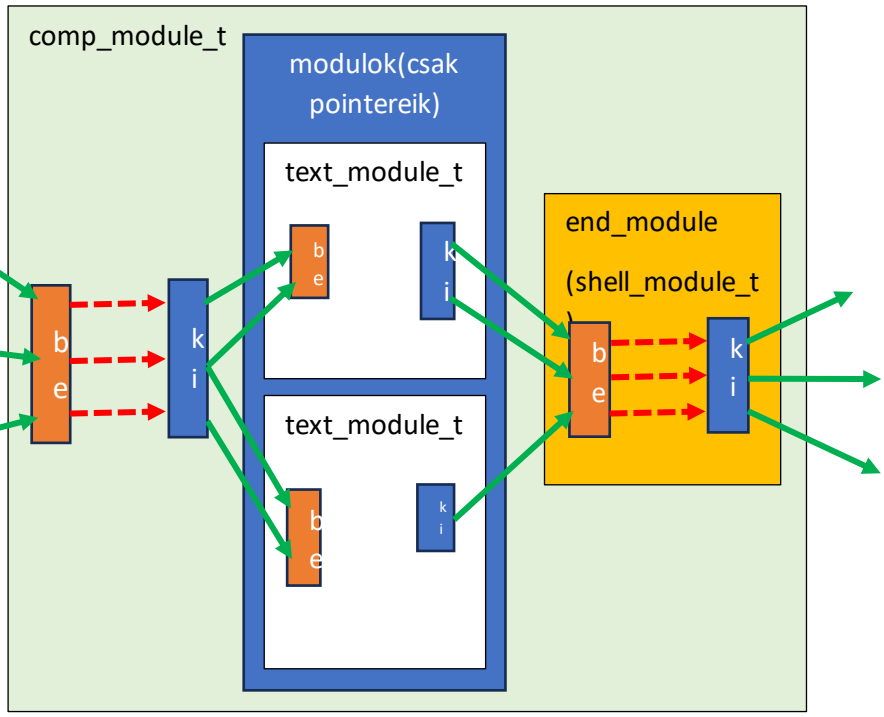
## 5. shell_module_t:

ez a modul vázat biztosít további moduloknak, ami beérkezik változás, egyből továbbítja a rákötött moduloknak

5.1. init: portok számának megadása, létrehozza a tömböket
5.2. addToList: átírva, a kapott indexű outputjára kötött modulokat adja hozzá a várakozók listájához, legalábbis meghívja azok addToList függvényét.
5.3. setBe: a saját be és kimenetét beállítja az adott indexen az adott értékre, valamint a hozzá kapcsolt moduloknak megváltoztatja a bemeneti adatait(a cél modul setBe hívásával)

## 6. comp_module_t:

ez a modul, ami modulokat tartalmaz, és a kapcsolások leírását, valamint lezárásként egy shell_module_t-t, amibe a belső modulok kivezetései futhatnak




6.1. comp_module_t létrehozásakor: Feltérképezi mettől meddig tartanak a modulok(muvCount)
  6.1.1. Létrehoz minden használt modulból egyet (compCreateModules)
  6.1.2. Majd elvégzi a bekötéseket (compFillModule, setConnection)
6.2. setKiPort: az endModuljának hívja meg a setKiPort függvényét
6.3. getKiErtek: az endModuljának hívja meg a getKiErtek függvényét
6.4. getKiNum: az endModuljának hívja meg a getKiNum függvényét

## 7. prot_module_t:

7.1. modulok és neveik tárolására való struct, konstruktorral és destruktorral a char* kezelése miatt

## 8. port:

8.1. generikus struct modulhivatkozás és portjának tárolására

## 9. tester_t:

9.1. tesztelésekre használható, tárolja a ki és bemeneti streamet, valamint a már jó modulokat
9.2. eggyezik: stringek összehasonlítására használt függvény
9.3. ujmnevEll: megnézi, hogy megfelelő-e az új modul név: nem volt még használva, nem tartalmaz tiltott karaktereket, '_'-vel kezdődik
9.4. kisbetujo: text module esetén megnézi, hogy jó-e a szintaxis, nem tartalmaz tiltott karaktereket, jó-e a bekötés
9.5. kisbetujo2: comp module esetén megnézi, hogy jó-e a szintaxisa az io-nak, jó-e a bekötésük
9.6. nagybetujo: comp module esetén megnézi, hogy jó-e a szintaxisa a belső vezetékezésnek, jó-e a bekötésük
9.7. modulesteszt: megnézi, hogy a használni kívánt modulok léteznek e már és meghívja az adott module testModule függvényét
9.8. test_module: a fentebbi függvényekkel teszteli, hogy létrehozható-e a kívánt új modul, és ha igen hozzáadja a modulok listájához (képes hozzáadás nélkül is tesztelni)

## 10. simulator_t:

10.1. getstring: egy adott istreamról olvas be enterig, vagy end_of_file-ig
10.2. modulator: megváltoztatja a modulátorok beállítását a kapott karakter alapján

10.3. ismod: megnézi, hogy a karakter egy a végrehajtásnál használható karakter-e
10.4. `inputHandlerDo`: végrehajtás utasítást hajt végre, először értelmezi (instructHandlerDo, instructHandlerCharHandler) segítségével
10.4.1. ha a lefutások száma > 0 akkor végigmegy a várakozó modulok listáján és meghívja azok végrehajtás függvényeit, az ezek miatt triggerelt modulok egy új várakozó modulok listába mentődnek. Amint a régi listáról minden végrehajtódik, az új lista veszi át a régi helyét
10.4.2. ez a folyamat a lefutások számaszor hajtódik végre.
10.5. inputHandlerModule: létrehozza az új modult, ha lehet(testModule), és ha az új modul neve '_main' akkor példányosítja a mMain változóban
10.6. inputHandlerRead: fájlból hajtja végre az utasításokat
10.7. inputHandlerWrite: végigmegy az insts listán(kiadott utasítások) és megkérdezi, hogy melyeket akarjuk menteni a fájlba, és közben menti azokat.
10.8. `inputHandler`: az előző négy függvényt kezelő függvény,
10.8.1. ha az utasítás '_'-vel kezdődik: inputHandlerModule,
10.8.2. '<'-vel kezdődik: inputHandlerRead,
10.8.3. '>'-vel kezdődik: inputHandlerWrite,
10.8.4. különben inputHandlerDo
10.9. end: ha end modulátor volt, akkor true-val tér vissza különben false-al
10.10. operator(): az inputHandler-t hívja meg a default(konstruktorral definiált) istreammel

### Változások az eredeti specifikációhoz, és Tervhez képest:

- Hibaüzenetek, csak névlegesen, tartalmuk nem változott
- Már képes a negációkat értelmesen kezelni
- Bejött egy új osztály(tipus) trilean, valamint módosultak a függvények (kevesebb get, objektum orientáltabb tagfüggvények)
- a const függvények továbbra sincsenek feltüntetve az UML-diagrammon, mert sok esetben ez egy függvény kettéválasztását jelnetette.

# Használati útmutató:

A specifikáció alapján lehet használni. Az egész úgy kezdődik, hogy modulokat kell definiálni, vagy beolvasni egy fájlt, ami modulok definiálásait tartalmazza. Ha már minden a _main modulhoz kellő modul definiálva van, lehet definiálni a _main modult. Ezek után megadhatjuk, a _main modulnak melyik portjára milyen értéket kapcsolunk. Nem kötelező minden portot minden parancsnál beállítani, hiányuk esetén nem változnak. És hogy hány itteráció fusson le. A program pedig kiírja az iterációk végén, mi van a _main modul outputján.

# Tesztelések:

A program tesztelésére a gtest_lite-ot használtam, amit a tantárgy keretein belül is használtunk.

A tesztek a program használatát immitálják, pontosabban a szimulátor objektumból hoznak létre példányokat (valódi lefutás esetén egy ilyen objektum keletkezne csak). A tesztprogramok, különböző _main modulokat használnak, többször előzetes modulok definiálásával, majd ezeknek input értékeket adva, vizsgálják, hogy megfelelő outputot adott-e a szimuláció. Az első teszt a hibaüzeneteket teszteli, majd sorra: egyszerű text_module, kétkivezetésű text_module, komp_module-t tartalmazó komp_module, érzékenységi llistát tartalmazó text_module, negációk, fájlba írás, fájlból olvasás.

Hibakezelésre memtrace-t használtam.