**Проект: Конзолен симулатор за цифрови интегрални схеми**

Проектът представлява симулатор за дефиниране и изпълнение на логически вериги, които извършват операции върху променливи, като AND (&), OR (|) и NOT (!). Целта е да се предостави инструмент за дефиниране на логически изрази, оценка на тяхната стойност, генериране на таблица на истинността и конюнктивна/дизюнктивна нормална форма (CNF/DNF), както и за извършване на детайлна обработка на грешки при въвеждане на логически изрази.

### **Обработка на грешки, които са покрити**

Проектът включва няколко проверки, които осигуряват коректността на въведените логически изрази:

1. Нереализирани променливи: Ако потребителят дефинира схема с неизвестна променлива (например define i(a, b): &(a, c)), програмата сигнализира за грешка.
2. Невалиден брой аргументи за оператора NOT: Операторът NOT (!) трябва да приема само един аргумент. В случай на повече от един, например define i(a, b): &(a, b, !(b, a)), се генерира грешка.
3. Недостатъчен брой аргументи за операциите AND и OR: Операциите AND (&) и OR (|) трябва да имат минимум два аргумента. Пример за грешка: |(a, &(c), !(a)).
4. Проверка на броя на скобите: Изрази с несъответстващи скоби (например &(a, !((b) или &(a, !(b)) се отбелязват като грешни.
5. Коректност на началния оператор: Всеки логически израз трябва да започва с една от операциите AND (&), OR (|) или NOT (!).
6. Проверка за дублиране на имената на схемите: При опит за дефиниране на схема със същото име, която вече съществува, се генерира грешка.
7. Коректност на аргументите при функцията run: Функцията run трябва да получава точно толкова стойности, колкото са дефинирани в схемата. Ако бъдат подадени повече или по-малко стойности, или стойности различни от 0 и 1, се изхвърля грешка.
8. Проверка на съществуването на схема при изпълнение на run/all: Програмата проверява дали съществува схема с зададеното име, преди да изпълни командата run или all.
9. Коректност на стойностите при run: При използване на командата run, ако потребителят въведе стойности различни от 0 или 1 за променливите в логическия израз, се генерира грешка. Например, ако въвеждането е run i(2, 0) или run i(a, x), програмата ще сигнализира за грешка, тъй като само стойности 0 и 1 са валидни.

### **Допълнителни възможности**

Проектът предлага следните функции за интерактивно взаимодействие с потребителя:

* Четене на таблица на истинността от файл: Функцията readTruthTableFromFile(const char\* fileName) чете таблица от посочен файл и обработва данните в структура TruthTable. При неуспешно отваряне на файла се хвърля изключение.
* Интерактивно дефиниране на таблица на истинността: Чрез функцията defineTable() потребителят може да въведе таблица на истинността чрез конзолата. Всеки ред е разделен със запетаи и двоеточия, като двоеточието отделя колоната за изход.
* Старт на симулатора: Функцията start() показва приветствено съобщение и списък с наличните команди, като предоставя примери за всяка от тях (например, define, run, all, find).
* Изпълнение на симулатора: Функцията startProgram(Simulator simulator) инициира симулатора и очаква вход от потребителя, като обработва командите define, run, all и find.

### **Класове и функции**

1. Circuit Class:
   * Конструктор: Инициализира обект от тип Circuit с име на веригата, променливи и логически израз.
   * Деструктор: Освобождава динамично разпределената памет за чл. променливи (името, променливите и израза).
   * getName, getExpression, getVariables: Връщат съответно името на схемата, логическия израз и променливите.
   * evaluate: Рекурсивно оценява логическия израз, като обработва оператори & (AND), | (OR), и ! (NOT).
2. Функции за валидация:
   * isInString: Проверява дали символът се съдържа в низ.
   * wrongInputHasCommonLetters: Проверява дали всички символи от по-дългия низ са в по-късия.
   * validateRecursive: Рекурсивна функция за валидация на логическите изрази с поддръжка на операциите AND, OR, и NOT.
   * isValidExpression: Проверява дали изразът е валиден.
   * exprOperation: Премахва част от израза, съдържаща само променливите.
3. Основни функции за работа с логическите вериги:
   * getCircuit и isCircuitInArr: Проверяват дали дадена схема вече съществува.
   * defineCircuit: Дефинира нова схема, като извършва всички проверки и създава нов обект от тип Circuit.
   * runCircuit: Изпълнява схема, като приема стойности за променливите и извършва логическите изчисления.
   * allCircuit: Генерира всички възможни комбинации от стойности за променливите и изпълнява логическите операции.
   * findCircuit: Генерира CNF или DNF израз, в зависимост от изходната стойност на схемата, използвайки 2D масив за представяне на таблица на истинността.