# **Верификатор за прости свойства**

**Абстракт**

В този проект осъществяваме верификацията на прости свойства за цели числа, числови масиви, низове и булеви променливи.

# **1 Въведение**

# 1.1 Предмет

# Проекта разглежда прости свойства на типовете: цяло число, низ, масив от цели числа, булеви променливи. Низовете се бележат с единични кавички „' '“, а масивите с фигурни скоби „{}“ и запетаи между отделните елементи. Празният символ се игнорира освен ако не е част от низ.

# Пример: „'а а'“ различно от „'аа'“, но „а = {5, 6}“ е еднакво с „а = {5,6}“.

## 1.1 Цел

## Целта на проекта е да се верифицират прости свойства за различните горепосочени типове:

## <, >, =, ==, <=, >=, !=, &&, ||, !, като всеки символ, включен в тези операции е запазен символ и не може да се въвеждат такива променливи. Валидни са всякакви входове, използващи еднократно сравнение или присвояване, докато булевите операции могат да бъдат използвани многократно, но само между булеви променливи, като още нещо специфично при булевите променливи е, че не могат да се използват операции за сравнение при тях. За всички други типове са валидни операциите за сравнение, както и операции +, \*. Възможно е използването на кръгли скоби „()“ (стандартна употреба).

## 1.2 Дейности

1. Сравняване на изрази по всички възможни начини за сравнение (<, >, <=, >=, ==, !=). За булеви променливи само „==“ и „!=“.

**Числа:** стандартни операции за сравнение

**Масиви от числа:** лексикографско сравнение

**Низове:** лексикографско сравнение

**Булеви променливи:** стандартни операции за сравнение

1. Операциите, които трябва да се поддържат от отделните типове:

**Числа:** събиране и умножение на число с друго такова или с масив от числа, като събирането на число с масив връща

**Низ:** конкатенация с друг низ (бележи се със знак +)

**Булеви променливи:** логически операции „*и*“ и „*или*“ с други булеви променливи

**Масив от числа:** събиране и умножение на масив от числа с друг такъв или с число, като операциите на число с масив представляват събиране или умножение на всеки елемент от масива с даденото число.

# **2 Методи и материали**

За да се извърши проверка дали въведения от потребителя низ е валиден използваме граматика. Най-удачно е да се ползва граматика, при която за всеки следващи *k* символа еднозначно е определено кое правило трябва да се използва. Такъв тип граматика се нарича *LL(k)* граматика, където първото L означава *left-to-right*, т.е. от ляво надясно се интерпретират символите във въведения низ, второто L означава *leftmost derivation*, което означава, че следващото правило за изпълнение е най-лявото в получения запис, а с k означаваме колко символа напред се проверяват на една итерация. Нашата граматика е LL(1). С такъв тип граматика се избягва backtracking и се използва при контекстно-свободни езици.

В проектът се имплементира следната граматика:

* Γ = (V, Σ, P, M)
* V — променливи (нетерминали) = {M, I, I1, S, S1, A, A1, B, B1, R}
* Σ — терминали = {**(**,**)**, **{**, **}**, **,**,**0-9**, **a-z**, **A-Z**, **‘**, **&**, **|**, **+**, **\***,**<**,**>**,**=**,**!**}
* P = V x (V ∪ Σ)\*— правила
* M ∈ V — начална променлива

1. M → (M) | I | S | A | B
2. I → (I) I1 | integer I1 | integer variable I1
3. I1 → +R | \*R | Ɛ
4. S → (S) S1 | string S1 | string variable S1
5. S1 → +S | Ɛ
6. B → (B) B1 | boolean B1 | boolean variable B1 | !B
7. B1 → &&B | ||B | Ɛ
8. A → (A) A1 | array A1 | array variable A1
9. A1 → +R | \*R | Ɛ
10. R → A | I | (R)

Правилото (всички правила са с главни букви) *М* е правилото, от което започва изчислението. Ако началния символ от входа е число или числова променлива, се преминава към правилото *I*. Ако е низ или променлива от тип низ, се преминава към правилото *S*. Ако е масив или променлива от тип числов масив, се преминава към правилото *А*. Възможно е първият елемент от входа да е скоба и тогава оставаме в правилото *М*, преминавайки на следващия символ. Това се повтаря, докато не открием някой от валидните типове. Ако бъде открита дума (поредица от символи между два празни символа), която не отговаря на валидните типове, изпълнението на проверката приключва и се извежда съобщение за невалиден вход. Останалите правила използват *терминал* (с малки букви), което прави еднозначно определянето на по-нататъшните действия, като не трябва да забравяме, че веднага щом срещнем правило, влизаме в него (подобно на depth first search). Символът Ɛ се използва, когато не е необходимо да има последващ символ съвпадащ с някоя от операциите за съответното правило. При заместване на всички възможни правила, ако не са останали „висящи“ терминали (такива, които не могат да бъдат заместени с някоя дума от входа) или не са останали думи от входа, а правилата да са свършили, тогава приемаме, че въведеният вход е бил верен.

За изчислението на самите изрази от двете страни на операторите за присвояване и сравнение се използва обратен полски запис. В обратния полски запис операторите следват своите операнди. Например, за да съберем 3 и 4, ще напишем "3 4 +", вместо "3 + 4". Ако има множествени операции, операторът е даден веднага след втория операнд, така че ако изразът е написан "3 - 4 + 5" в обратен полски запис ще бъде написан по следния начин "3 4 - 5 +". Предимството на обратния полски запис е това, че премахва нуждата от скоби, които се изискват в инфиксната нотация. Докато "3 - 4 \* 5" също може да бъде написано "3 - (4 \* 5)", което означава нещо много по-различно от "(3 - 4) \* 5. Предходният израз също може да бъде написан "3 4 5 \* -", което недвусмислено означава "3 (4 5 \*) -", което след това се изчислява като "3 20 -".

При сравнителни тестове на обратния полски запис с алгебричен запис се установява, че обратният полски запис води до по-бързи изчисления поради две причини. Изчисленията на обратния полски запис нямат нужда от изрази, за да бъдат параметризирани, по-малко операции трябва да бъдат вкарани, за да се извършат обичайните изчисления. Като допълнение, при изчисленията с обратен полски запис се правят по-малко грешки, отколкото при другите типове на изчисления. По-късно проучване показва, че увеличената скорост от обратния полски запис може да е сведена до по-малкия брой на натискане на клавишите необходим да се въведе тази нотация, отколкото по-малкия познавателен товар на неговите потребители. Въпреки това, много доказателства показват, че обратният полски запис е по-труден за научаване от потребителите, отколкото алгебричната анотация.

Алгоритъмът за изчисляване на резултат на израз, записан с обратен полски запис, е сравнително лесен. За целта се използва структурата от данни Стек (Stack).

* За всеки пореден символ се извършва една от следните операции:
  + Ако символът е число:
    - добавяме числото в стека
  + Ако символът е оператор (операторът може да включва и функции):
    - Знае се, че операторът използва **n** аргумента - за стандартните математически операции събиране, изваждане, умножение, деление и степенуване се използват два аргумента - **n = 2**
    - Ако в стека има по - малко от **n** числа:
      * **Грешка** – потребителят е въвел непълен израз - пример: "3 + "
      * В нашия проект първо се извършва операция по валидация, т.е. не може да се стигне до случая грешка
    - Ако има достатъчно числа, изваждаме **n** на брой от стека
    - Извършваме операцията
    - Добавяме в стека резултата от операцията
* След като изпълним горните операции за всеки символ в записа:
  + Ако има останало само едно число в стека:
    - числото е резултатът от израза
  + Ако има повече от едно число
    - **Грешка** – потребителят е въвел непълен израз

# **3 Резултати и дискусии**

# В проекта успяхме да реализираме интерпретация на LL(1) граматика, валидираща прости операции. Реализирахме преобразуването на израз в обратен полски запис, както и пресмятането му след това, за да верифицираме подаден от потребителя израз. Проектът поддържа желаните операции за сравнение, присвояване, събиране, умножение, конкатенация, логически операции.

# **Литература**

# [1] [Predictive parsing](https://www.youtube.com/watch?v=0wbDm2q5dH4)

# [2] [FMI lecture - grammars](https://learn.fmi.uni-sofia.bg/mod/resource/view.php?id=47255)

# [3] [FMI lecture – RPN](https://learn.fmi.uni-sofia.bg/mod/resource/view.php?id=46721)

# [4] [Wikipedia - RPN](https://bg.wikipedia.org/wiki/Обратен_полски_запис)