*Софийски университет „Св. Климент Охридски“  
Факултет по математика и информатика*

КУРСОВ ПРОЕКТ ПО СТРУКТУРИ ОТ ДАННИ И АЛГОРИТМИ

зимен семестър 2021/2022г.

Проект 5: Интерпретатор за функционалния език **ListFunc**

Изготвил: **Тошко Антонов Бацанов**

Специалност: **Информационни системи**

Факултетен номер: **72076**

Група: 3

февруари 2022

гр. София

Съдържание

**Глава 1. Увод** (3 стр.)

1.1. Описание и идея на проекта……………………………………………..…………..

**Глава 2. Преглед на предметната област** (4-5 стр.)

2.1. Основни дефиниции, концепции и алгоритми…………………………………….

2.2. Дефиниране на проблеми и сложност на поставената задача……………………..

**Глава 3. Реализация, тестване** (6 стр.)

3.1. Управление на паметта.-----------------------------------------------------------------------

3.2. Планиране, описание и създаване на тестови сценарии --------------------------------

***1.1 Описание и идея на проекта***

Основната идея на проекта е реализацията на интерпретатор*,* който извършва последователен анализ на командите от изходния код, непосредствено ги превежда и изпълнява. Целта е интерпретаторът да може да работи в интерактивен режим, в който позволява на потребителя да пише ред код, който се оценява и се извежда резултат от оценката. Също така интерпретаторът трябва да може да се стартира върху файлове, които да се изпълняват и резултатът от тях да се отпечатва на изхода на вашата програма.

Приложението приема, обработва и пресмята резултата от въведения вход от потребителя, като при нужда пази необходими данни при дефиниция на нова фунцкия в подходяща структура от данни. Приложението поддържа филтрация на данни, така че при сгрешени входни данни от страна на потребителя, то отговаря с информация за съответната грешка.

***2.1. Основни дефиниции, концепции и алгоритми***

Проектът е върху основата на три класа *Reader*, *Parser* и *Interpretor, Tokenizer и Expressions*.

*Reader* е клас, който съдържа в себе си eдин много важен алгоритъм за системата. Той съдържа в себе си метод, с който приема входа на потребителя и чрез помощта на класа *Tokenizer*, го превръща в отделни *Token-и*.

Клас *Tokenizer* e класът, който се грижи за манипулацията на входните данни.

*Parser* поема задачата да приеме *Token-*ите и да ги вкара в структура от данни- дърво.

*Interpretor* е клас, който извършва изчисленията от системата. Когато той получи дървото създадено от *Parser,* проверява дали е дефиниция на нова функция или извикване на функция и извършва съответното действие. Ако е нова функция то той запазва дървото, описващо функцията, в клас UserGeneratedFunctions, който използва Хеш Таблица, за да складира функциите дефинирани от потребителя. Ако е извикване на функция, той пресмята резултата и го извежда на екрана.

Клас *Expressions* е виртуален с член-данна *Token* *operation* и виртуални функции *print()* и *getCopy().* Идеята на класа е да бъде наследен от видовете *Expression-и*, а именно *InfiniteListLiteral, FiniteListLiteral, Number, FunctionCall и Argument.* По този начин мога да използвам хетерогенни контейнери, чрез които да не правя разлика между различните Expression-и, но същевременно при нужда дефинираните виртуални функции да бъдят изпълнени според различните видове.

***2.2. Дефиниране на проблеми и сложност на поставената задача***

Един от проблемите е свързан с правилното извличане на информацията от входа на потребителя, тъй като той е цял *String*. Ако тези данни са коректни, то програмата ще ги изпълни.

Друг проблем е именно грешните потребителски данни. Трябва да се предвиди, какво ще се случи, когато потребителят въведе данни, които не се очакват от него, например различен тип или несъществуващ аргумент, команда и т.н. Също така е важно и какво ще се случи, когато потребителят не въведе нищо, или въведе по-малко , или повече аргументи.

Друго нещо е въпросът за вкарването на различните функции в дърво което в следствие да бъде изчислено.

Не на последно място излиза проблемът с изчисляването на дървото, което да спазва реда на изпълнение на функциите и коректно да използва аргументите.

**4.1. Управление на паметта. Оптимизации.**

За управлението на паметта се грижат две член-данни в класа *Interpreter –*

*junkNodeCollector и junkExpressionCollector.* При всяко алокиране на памет в *heap-а,* ако се прецени, че паметта ще бъде ненужна в бъдеще, то указател към нея бива вкаран в един от двата вектора, в зависимост дали е дърво или *Expression.*

**4.2. Планиране, описание и създаване на тестови сценарии`**

Създадох втори проект в *solution-a*, който вместо *main* изпълнява *test.cpp* файл, в който съм тествал основните функционалности и хвърлените грешки, когато е необходимо.