

**СОФИЙСКИ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ  
ОХРИДСКИ“**

**Факултет по математика и информатика**



**Структури от данни  
Курсов проект на тема:  
„IML“**

**Изготвила:**

**Весела Илиянова Стоянова**

**Ф.Н.**

**71949**

**2 курс**

**Бакалавър**

**Специалност:**

**Информационни системи**

**Ръководител:**

**Калин Георгиев**

**20.01.2021 г.**

## Съдържание:

|    |  |   |
|----|--|---|
| 1. | Описание и идея на проекта.....                          | 3 |
| 2. | Използвани структури от данни.....                       | 3 |
| 3. | Потребителски изисквания .....                           | 4 |
| 4. | Планиране, описание и създаване на тестови сценарии..... | 9 |

## Таблицы и графики:

|         |  |   |
|---------|--|---|
| Фиг.1:  | Методите на клас Stack .....                                       | 3 |
| Фиг.2:  | Използваният unordered_map.....                                    | 3 |
| Фиг.3:  | Методите на клас Vector .....                                      | 4 |
| Фиг.4:  | Снимка при въведен невалиден входен файл.....                      | 4 |
| Фиг.5:  | Снимка при въведен валиден вход с валидни данни .....              | 5 |
| Фиг.6:  | Член-данни и методи на клас Tag .....                              | 5 |
| Фиг.7:  | Методите на клас Parser .....                                      | 6 |
| Фиг.8:  | Функция evaluateContent (Vector<string> tagRows).....              | 7 |
| Фиг.9:  | Функция parseArgument (string operAndArgum, Operation operation) . | 8 |
| Фиг.10: | Функция parseNumbers (string numbersStr).....                      | 8 |

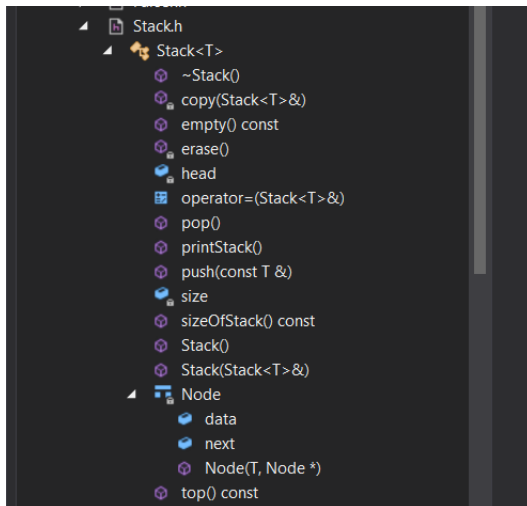
## 1. Описание и идея на проекта

В рамките на зададения проект трябва да се реализира програма, реализираща **parser** за езика **IML**. Езикът съдържа различни тагове, които могат да се влагат. Програмата извършва агрегиращи, трансформиращи и сортиращи операции върху списък от сравними данни.

## 2. Използвани структури от данни

За реализацията на проекта са използвани следните структури от данни:

- **Stack**



- Използването на стек е свързано с основната идея на проекта – първият влезнал елемент да излезне последен.

Фиг.1: Методите на клас *Stack*

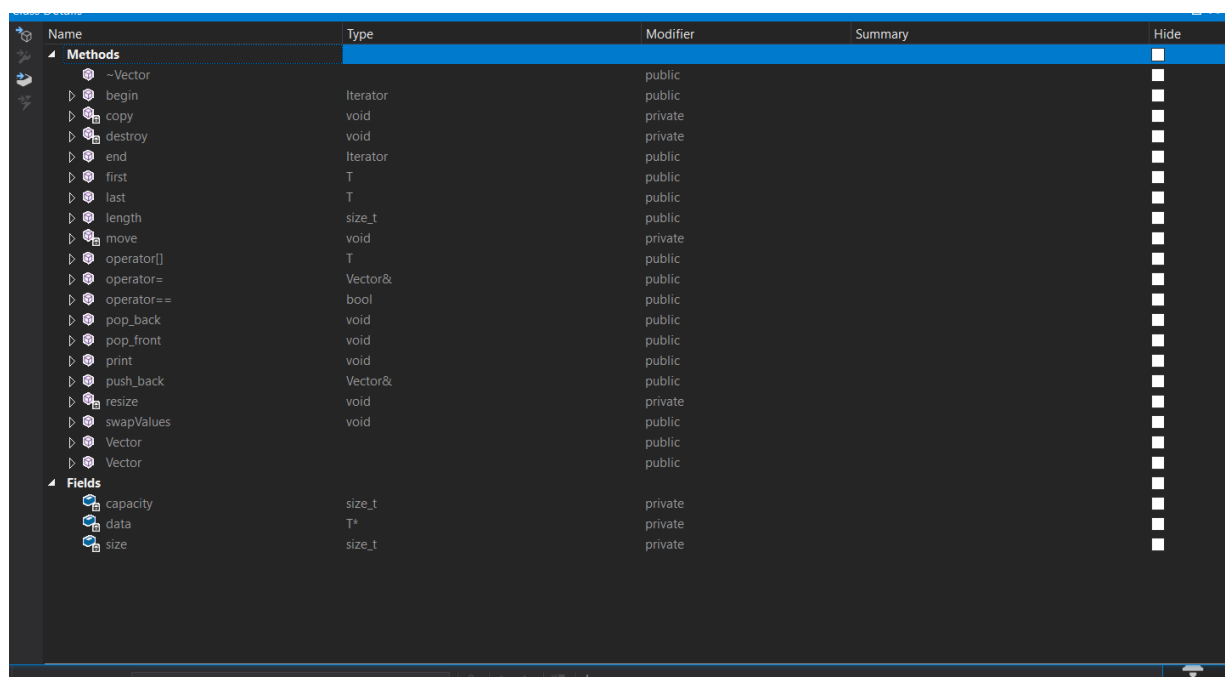
- **Unordered\_map**

```
20
21 static unordered_map<string, Operation> const operationTable =
22 {
23     {"MAP-INC", MAP_INC},
24     {"MAP-MLT", MAP_MLT},
25     {"AGG-SUM", AGG_SUM},
26     {"AGG-PRO", AGG_PRO},
27     {"AGG-AVG", AGG_AVG},
28     {"AGG-FST", AGG_FST},
29     {"AGG-LST", AGG_LST},
30     {"SRT-REV", SRT_REV},
31     {"SRT-ORD", SRT_ORD},
32     {"SRT-SLC", SRT_SLC},
33     {"SRT-DST", SRT_DST}
34 };
35
```

- Използваният **unordered\_map** има ключ, който е стрингът, който се подава, а стойността му е **Operation**.

Фиг.2: Използваният *unordered\_map*

- **Vector**



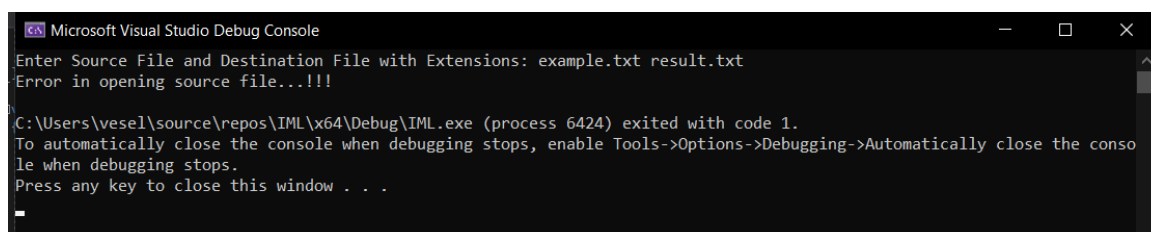
| Name           | Type     | Modifier | Summary | Hide |
|----------------|----------|----------|---------|------|
| <b>Methods</b> |          |          |         |      |
| ~Vector        |          | public   |         |      |
| begin          | Iterator | public   |         |      |
| copy           | void     | private  |         |      |
| destroy        | void     | private  |         |      |
| end            | Iterator | public   |         |      |
| first          | T        | public   |         |      |
| last           | T        | public   |         |      |
| length         | size_t   | public   |         |      |
| move           | void     | private  |         |      |
| operator[]     | T        | public   |         |      |
| operator=      | Vector&  | public   |         |      |
| operator==     | bool     | public   |         |      |
| pop_back       | void     | public   |         |      |
| pop_front      | void     | public   |         |      |
| print          | void     | public   |         |      |
| push_back      | Vector&  | public   |         |      |
| resize         | void     | private  |         |      |
| swapValues     | void     | public   |         |      |
| Vector         |          | public   |         |      |
| Vector         |          | public   |         |      |
| <b>Fields</b>  |          |          |         |      |
| capacity       | size_t   | private  |         |      |
| data           | T*       | private  |         |      |
| size           | size_t   | private  |         |      |

Фиг.3: Методите на клас **Vector**

- Векторът има различни методи, които са необходими за реализацията на проекта.
- Има метод за намиране дължината на вектор, първия елемент от вектора, последния елемент от вектора, добавяне и премахване на елементи в края на вектора и други.

### 3. Потребителски изисквания

Функционалността на програмата започва в **main** функцията, където се извиква функцията **getUserInput()**, която е свързана с потребителския вход. При стартиране на програмата потребителят трябва да въведе име на входен файл и име на изходен файл с интервал между тях. Ако не съществува такъв входен файл, на конзолата излиза съобщение за грешка.



```

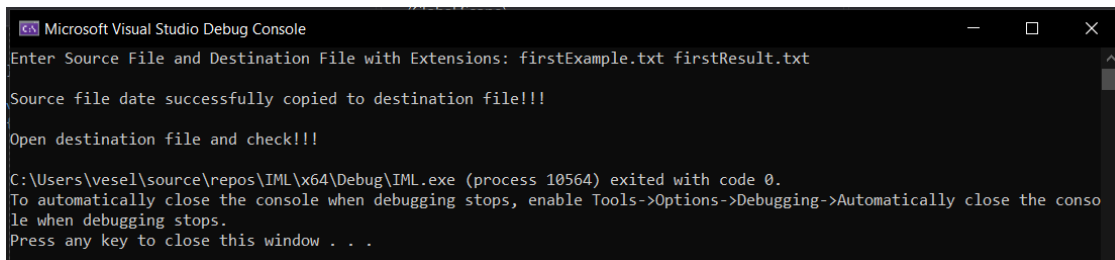
Microsoft Visual Studio Debug Console
Enter Source File and Destination File with Extensions: example.txt result.txt
Error in opening source file...!!!

C:\Users\vesel\source\repos\IML\x64\Debug\IML.exe (process 6424) exited with code 1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```

Фиг.4: Снимка при въведен невалиден входен файл

Ако съществува такъв входен файл, но има някаква синтактична грешка при въвеждането на таговете, на конзолата излиза каква е грешката. Ако съществува такъв входен файл и няма никакви синтактични грешки, то излиза съобщение, че файлът е успешно записан.

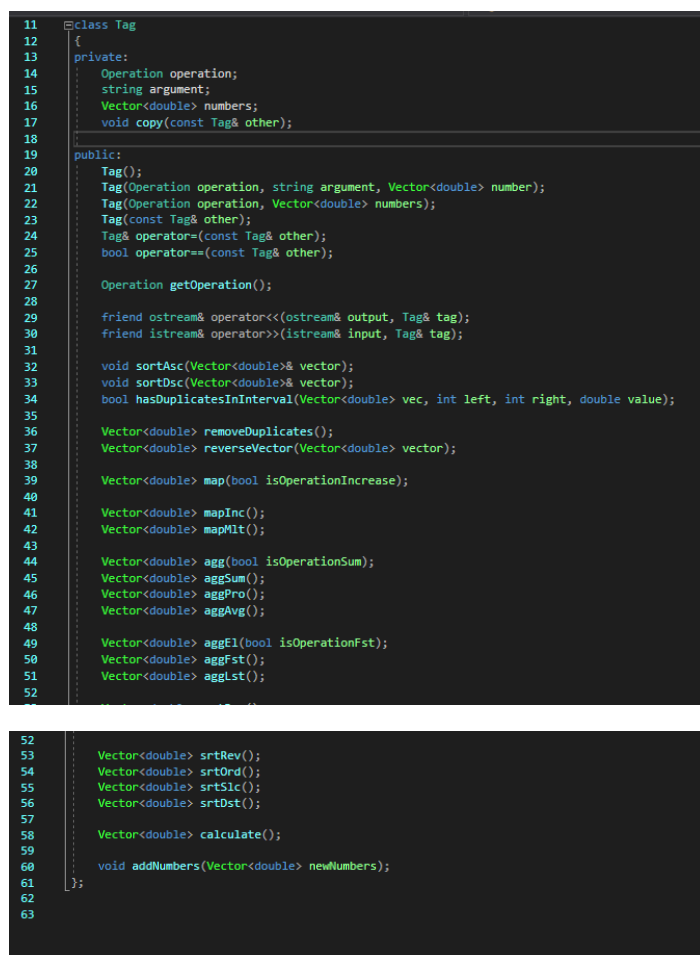


```
Microsoft Visual Studio Debug Console
Enter Source File and Destination File with Extensions: firstExample.txt firstResult.txt
Source file date successfully copied to destination file!!!
Open destination file and check!!!
C:\Users\vesel\source\repos\IML\x64\Debug\IML.exe (process 10564) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Фиг.5: Снимка при въведен валиден вход с валидни данни

В изходния файл е записана стойността на съответния таг.

За цялостната реализация на проекта е създаден клас **Tag**, който съдържа конструкции за създаването на един таг, както и съответните функции за изчисляване.



```
11 class Tag
12 {
13 private:
14     Operation operation;
15     string argument;
16     Vector<double> numbers;
17     void copy(const Tag& other);
18
19 public:
20     Tag();
21     Tag(Operation operation, string argument, Vector<double> number);
22     Tag(Operation operation, Vector<double> numbers);
23     Tag(const Tag& other);
24     Tag& operator=(const Tag& other);
25     bool operator==(const Tag& other);
26
27     Operation getOperation();
28
29     friend ostream& operator<<(ostream& output, Tag& tag);
30     friend istream& operator>>(istream& input, Tag& tag);
31
32     void sortAsc(Vector<double>& vector);
33     void sortDesc(Vector<double>& vector);
34     bool hasDuplicatesInInterval(Vector<double> vec, int left, int right, double value);
35
36     Vector<double> removeDuplicates();
37     Vector<double> reverseVector(Vector<double> vector);
38
39     Vector<double> map(bool isOperationIncrease);
40
41     Vector<double> mapInc();
42     Vector<double> mapMlt();
43
44     Vector<double> agg(bool isOperationSum);
45     Vector<double> aggSum();
46     Vector<double> aggPro();
47     Vector<double> aggAvg();
48
49     Vector<double> aggEl(bool isOperationFst);
50     Vector<double> aggFst();
51     Vector<double> aggLst();
52
53     Vector<double> srtRev();
54     Vector<double> srtOrd();
55     Vector<double> srtSlc();
56     Vector<double> srtDst();
57
58     Vector<double> calculate();
59
60     void addNumbers(Vector<double> newNumbers);
61 };
62
63
```

Фиг.6: Член-данни и методи на клас **Tag**

Също така е създаден клас **Parser**, който съдържа методите за преобразуването на тага.

```
11
12 class Parser
13 {
14 public:
15     Vector<double> parseContent(string content);
16
17 private:
18     Vector<double> evaluateContent(Vector<string> tags);
19     Operation parseClosingOperation(string operAndArgum);
20     Operation parseOperation(string operAndArgum);
21     string parseArgument(string operAndArgum, Operation operation);
22     Vector<double> parseNumbers(string numbersStr);
23 };
```

Фиг.7: Методите на клас *Parser*

В него се намират проверките за валидност на подадения таг – дали започва с '<', дали отварящият и затварящият таг са еднакви, дали завършва с '>', дали функциите част от функциите имат аргумент и други и извежда съответното съобщение за грешка при невалидност на входа. В клас **Parser** се намират и основните функции – **evaluateContent**, **parseArgument** и **parseNumbers**. Във функцията **evaluateContent** се намира основната логика на проекта – взимаме най-горния елемент от стека, пресмятаме го и го премахваме от стека. Докато имаме елементи в стека, взимаме най-горния елемент, пресмятаме го и го премахваме от стека като запазваме резултата.

```

32
33 Vector<double> Parser::evaluateContent(Vector<string> tagRows)
34 {
35     Stack<Tag> tags;
36     Vector<double> result;
37
38     for (auto it = tagRows.begin(); it != tagRows.end(); ++it)
39     {
40         istringstream iss(*it);
41         Vector<string> components = splitToWords(&iss, '>');
42
43         string operAndArgum = components[0];
44         bool isClosingTag = operAndArgum.length() > 0 && operAndArgum[0] == '/';
45
46         Operation operation;
47         string argument;
48
49         if (isClosingTag)
50         {
51             operation = parseClosingOperation(components[0]);
52         }
53
54         else
55         {
56             operation = parseOperation(components[0]);
57             argument = parseArgument(components[0], operation);
58         }
59
60         bool hasNumbers = components.length() > 1;
61         Vector<double> numbers;
62         if (hasNumbers)
63         {
64             numbers = parseNumbers(components[1]);
65         }
66
67         if (isClosingTag)
68         {
69             //Проверяваме дали имаме затварящ таг, но стекът от тагове е празен
70             if (tags.empty())
71             {
72                 cout << "There is an additional closing tag." << endl;
73                 exit(1);
74             }
75             Tag lastTag = tags.top();
76
77             if (isClosingTag)
78             {
79                 //Проверяваме дали имаме затварящ таг, но стекът от тагове е празен
80                 if (tags.empty())
81                 {
82                     cout << "There is an additional closing tag." << endl;
83                     exit(1);
84                 }
85             }
86             Tag lastTag = tags.top();
87             //Проверяваме дали операцията на затварящия таг е различна от тази на отварящия
88             if (operation != lastTag.getOperation())
89             {
90                 cout << "The closing tag is different from the opening tag." << endl;
91                 exit(1);
92             }
93
94             //Взимаме най-горния елемент на стека и го пресмятаме
95             result = lastTag.calculate();
96             //После по-ваме горния елемент
97             tags.pop();
98             //Проверяваме дали имаме още елементи в стека
99             if (!tags.empty())
100             {
101                 Tag newLastTag = tags.top();
102                 tags.pop();
103                 newLastTag.addNumbers(result);
104                 newLastTag.addNumbers(numbers);
105                 tags.push(newLastTag);
106             }
107         }
108         else
109         {
110             Tag tag = Tag(operation, argument, numbers);
111             tags.push(tag);
112         }
113     }

```

Фиг.8: Функция *evaluateContent* (*Vector<string> tagRows*)

```

129
130 string Parser::parseArgument(string operAndArgum, Operation operation)
131 {
132     //Проверяваме дали функциите, които трябва да имат аргумент, го имат
133     bool needsArgument = operation == MAP_INC || operation == MAP_MLT || operation == SRT_ORD || operation == SRT_SLC;
134
135     istringstream iss(operAndArgum);
136     Vector<string> components = splitToWords(&iss, ' ');
137
138     bool hasArgument = components.length() > 1;
139
140     //Проверка, ако трябва да има аргумент, но няма
141     if (needsArgument && !hasArgument)
142     {
143         cout << "The required argument for the operation " << operation << " is missing." << endl;
144         exit(1);
145     }
146
147     //Проверка, ако не трябва да има аргумент, но има
148     if (!needsArgument && hasArgument)
149     {
150         cout << "There is a redundant argument for the operation " << operation << " . " << endl;
151         exit(1);
152     }
153
154     //Проверка, ако трябва да има аргумент и има
155     if (needsArgument && hasArgument)
156     {
157         string argument = components[1];
158         argument.erase(remove(argument.begin(), argument.end(), '\\'), argument.end());
159         return argument;
160     }
161
162     return "";
163 }

```

Фиг.9: Функция *parseArgument (string operAndArgum, Operation operation)*

```

177
178 Vector<double> Parser::parseNumbers(string numbersStr)
179 {
180     istringstream iss(numbersStr);
181     Vector<string> components = splitToWords(&iss, ' ');
182
183     Vector<double> result;
184     for (int i = 0; i < components.length(); i++)
185     {
186         //Проверяваме дали векторът се състои само от числа
187         if (is_number(components[i]))
188         {
189             result.push_back(stod(components[i]));
190         }
191
192         else if (components[i] == "\\n")
193         {
194             return result;
195         }
196
197         else
198         {
199             cout << "A not valid digit occurred." << endl;
200             exit(1);
201         }
202     }
203
204     return result;
205 }
206
207

```

Фиг.10: Функция *parseNumbers (string numbersStr)*

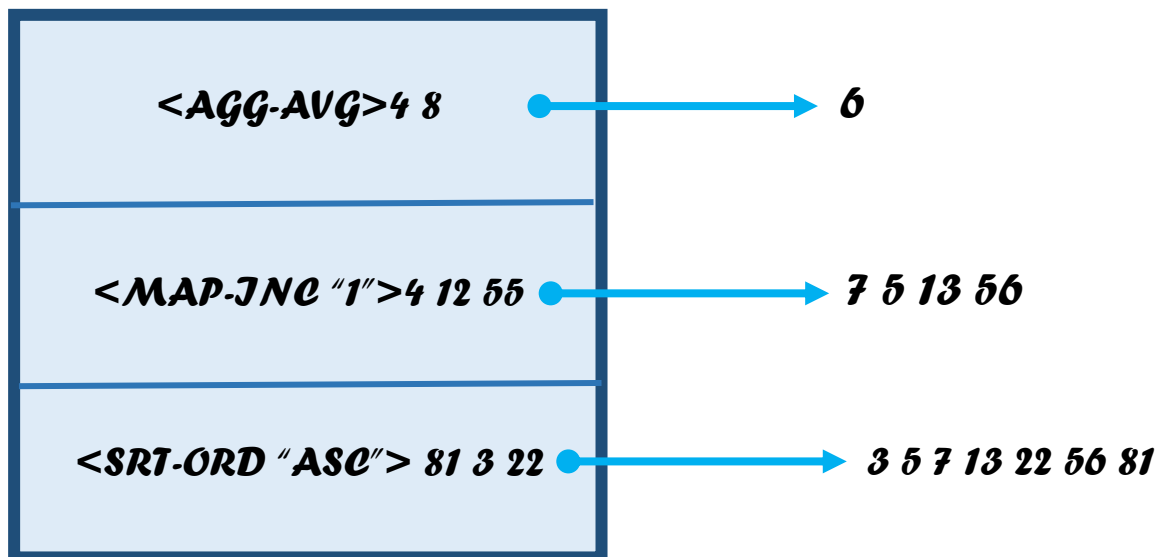


## 4. Планиране, описание и създаване на тестови сценарии

Първите стъпки от тестването на проекта включват тестване на конструкторите на класовете, тъй като ако се получи грешка там, ще доведе до грешки в програмата и на по-късен етап би било трудно да се намерят. След това бяха тествани всички агрегиращи, трансформирани и сортиращи функции, като всяка една от тях беше тествана, както с валиден, така и с невалиден вход. Също така всяка една беше тествана с различни числа - положителни, отрицателни и дробни. След това беше тествана функцията за отваряне на файл. Накрая бяха създадени няколко примера, които показват работата на програмата.

**firstExample.txt**

```
<SRT-ORD "ASC">81 3<MAP-INC "1">4 12 55<AGG-AVG>4 8</AGG-AVG>
</MAP-INC>22</SRT-ORD>
```



Основната идея тук е, че при получаване на тага <SRT-ORD "ASC">81 3, ние го добавяме в стека. При получаване на следващия таг <MAP-INC "1">4 12 55, ние отново го добавяме в стека. Получаваме и следващ таг <AGG-AVG>4 8, като и него добавяме в стека. Когато видим затварящ таг </AGG-AVG>, пресмятаме стойността на <AGG-AVG>4 8, която е 6. След това виждаме следващия затварящ таг </MAP-INC> и пресмятаме и неговата стойност като прибавяме стойността на предишния таг, тоест получаваме векторът 7 5 13 56. Виждаме числото 22, което се отнася за тага <SRT-ORD "ASC"> и пресмятаме стойността му и получаваме 3 5 7 13 22 56 81. Виждаме, че стекът вече е празен, тоест крайният резултат е 3 5 7 13 22 56 81.

Към проекта са приложени няколко примера, както и изображения, които показват как всеки един от тях работи.

**Връзка към хранилище в Github:**

<https://github.com/VeselaStoyanova/IML>