IML(Ivan markup language)

Проектът ми е изграден на основата на абстрактният клас **Tag**. Той се задава чрез име на тага от тип **string** атрибут на тага пак от тип **string** и вектор от елементите които се съдържат в тага които са от тип double – **vector<double>** .При „парсването“ евентуално вложените елементи първо се оценяват и след това се добавят към вектора със елементите.Оценката на таг става чрез виртуалната функция **apply** която е от тип **vector<double>.**За различните тагове са реализирани различни класове които наследяват базовият **Tag**. Всички наследници съдържат логика по използването на стрингът атрибут.

Също така съм реализирал няколко помощни класа които внасят самата функционалност на „парсване“ на езика.Това е класът **Helper** съдържащ функции които тестват дадена част от входният текст – дали е атрибут, таг или число, дали е идентационен символ, както и функционалността по прибавянето на един вектор към края на друг (подробно описание на фунцкиите ще дам след малко).Следва класът **Formatter** който служи за подготовка и обработка на входните данни, така че те да бъдат по-лесно интерпретирани – това включва разделянето на входният текст на „думи“ както и самото форматиране на тези думи както и прави търсене на текста за намиране на съответният затварящ таг когато се стигне до отварящ. Също така във него е включена функционалността за четене и писане от файл. Класът **Parser** осъществява гавната логика по превръщането на форматираните думи във контейнер от числа със плаваща запетая. Всичките публични методи на тези помощни класове са статични.

Workflow-а на програмата е следният: първо се прочита файлът, добавят се празни символи между таговете и числата и текстът се разделя на думи. След това се извиква рекурсивната функция **_parse** която е защитена. Дефиницията и е следната:

private static vector<double> Parser::_parse(
vector<string> words,
int start_pos,
int end_pos
) – по зададен контейнер от думи и начална и крайна позиция изчислява стойността на всички елементи между двата индекса. Елементите могат да бъдат числа както и други тагове. За целта обхождам даденият интервал като ако срещна число го вкарвам във контейнер от числа, а ако срещна таг му намирам затарящият (ако има) и след това извиквам функцията

private static Tag* Parser::factory(
vector<string> words,
int start_pos,
int end_pos
) – тя връща елемент от тип **Tag*** който съдържа всичките оценени елементи между двата му подадени индекса. След това извиквам **apply** която оценява елемента и прибавям оценката му към контейнера с натрупаните досега числа. Принципът на който работи функцията **factory** е следният: взимам името и атрибута на тага(ако има) след това извиквам функцията parse за останалите елементи между тези два индекса и след това

конструирам обект от съответният клас. Всички класове се конструират като базовият, като единствената им разлика е в начина на работа на **apply** и оценката на атрибута им.

Tag::Tag(
string tag_name,
string attribute,
vector<double> elements
).

След това записвам полученият резултат във файл посредстом функцията **write_file** на **Parser**.

Описание на функциите:

Helper::

bool is_tag(string item) – проверка дали думата е таг

bool is_number(string item) – проверка дали е число

bool is_attribute(string item) – проверка дали е атрибут(тоест дали е оградено в кавички число. Проверката дали е валиден атрибут и дали вече има атрибут се прави суответно при създаването на елемента от factory както и преди използване на съответният атрибут).

bool not_whitespace(char symbol) – проверява дали символът не е ‘ ’, ‘\t’, ‘\n’

void append(vector<double> target, vector<double> source) – осъществява логиката по добавянето на един контейнер към друг.

Formatter::

vector<string> split_whitespace(string text) – разделям текстът на думи посредством празните символи – затова е важно преди това да съм добавил празни символи между таговете и числата и между два тага.

string remove_parenthesis(string item) – премахва начупените скоби от думите

int scanForClosingTag(vector<string> container, string tag, int start_pos, int end_pos) – сканира контейнера на думите и намира (ако има) позицията в контейнера на съответният затварящ таг. Указва му се от коя до коя позиция да търси. При не наличие на даденият таг връща позиция -1 която е невалидна позиция.

string read_file(string source) – прочита файлът чиито път е променливата **source** и го връща във формата на **string**.

string format_input(string raw) – добавя празни символи между кавичките и начупените скоби, между начупени скоби и числа и между съседни начупени скоби.

void write_file(vector<double> container, string destination) – пише съдържанието на контейнера разделено чрез празно място във файл чийто път е обозначен чрез променливата **destination**.

Parser::

private static vector<double> _parse(
vector<string> container,
int start_pos,
int end_pos
) – по зададен контейнер и две позиции изчислява всички думи и ги вкарва във контейнер. Това включва и изчислението на различните тагове.

double parse_number(string item) – конструира число от **string**

Tag* factory(vector<string> container, int start_pos, int end_pos)