1. Министерство высшего образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. Институт прикладной математики и механики
4. **Высшая школа Кибербезопасности и Защиты Информации**
5. **КУРСОВАЯ РАБОТА**
6. **Создание распознавателя расширяемого языка разметки XML**
7. по дисциплине «Формальные грамматики и теория компиляторов»
8. Выполнил
9. студент гр.3651003/70801 <*подпись*> Р.И.Дружков
10. Руководитель
11. старший преподаватель <*подпись*> П.В.Семьянов
12. «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.
13. Санкт-Петербург
14. 2020

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc50339558)

[1 Язык XML 4](#_Toc50339559)

[2 Создание распознавателя 5](#_Toc50339560)

[3 Распознаваемые ошибки 5](#_Toc50339561)

[4 Тестирование распознавателя 6](#_Toc50339562)

[Заключение 11](#_Toc50339563)

[Список использованных источников 12](#_Toc50339564)

[Приложение 1 13](#_Toc50339565)

[Приложение 2 18](#_Toc50339566)

[Приложение 3 22](#_Toc50339567)

Введение

Целью данной курсовой работы является написание распознавателя современного формального языка с помощью генератора синтаксических анализаторов YACC, генератора лексических анализаторов Flex и языка C или C++.

Первые версии YACC и Flex увидели свет более 30-ти лет назад. Несмотря на то, что обе программы поддерживаются до сих пор, они уже довольно стары. Встает вопрос о том, позволяют ли принципы, заложенные в данные утилиты, разрабатывать распознаватели современных формальных языков. Актуальность курсовой работы заключается в исследовании возможности построения распознавателя современного формального языка с помощью утилит YACC и Flex.

В качестве современного формального языка был выбран расширяемый язык разметки XML. XML - язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком, с подчёркиванием нацеленности на использование в Интернете [1].

Для достижения поставленной цели необходимо выполнить следующие задачи:

* изучить синтаксис, спецификацию и особенности расширяемого языка разметки XML;
* разработать программу для Flex, с целью создания лексического анализатора для нашей грамматики;
* разработать грамматику расширяемого языка разметки XML для генератора синтаксических анализаторов YACC;
* разработать программу на языке С, использующую результаты работы Flex и YACC для построения распознавателя XML;
* протестировать разработанный распознаватель XML;

# Язык XML

XML - расширяемый язык разметки. Рекомендован Консорциумом Всемирной паутины (W3C). Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком, с подчёркиванием нацеленности на использование в Интернете. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка [1].

Опишем ключевые особенности языка XML [2].

* XML документы могут иметь пролог, описывающий используемые версию XML и кодировку;
* XML документы имеют древовидную структуру, дерево XML начинается с одного корневого элемента, продолжается через ветви к элементам – детям;
* каждый элемент может быть родительским;
* каждый элемент может иметь текстовый контент;
* элемент описывается с помощью открывающего и закрывающего тэга элемента;
* открывающий тег элемента начинается с символа «<», далее идет имя элемента, его атрибуты и символ «>»;
* закрывающий тег элемента начинается с символов «</», далее идет имя элемента и символ «>»;
* атрибуты начинаются с имени атрибута, знака «=» и значения атрибута в одинарных или двойных кавычках;
* число атрибутов элемента неограниченно;
* в название элемента можно добавить пространство имен, с помощью которого могут быть разрешены конфликты имен;

# Создание распознавателя

В ходе выполнения работы были написаны программа для Flex (Приложение 1), грамматика расширяемого языка разметки XML для генератора синтаксических анализаторов YACC (Приложение 2), программа на языке С++ (Приложение 3), использующую результаты работы Flex и YACC для построения распознавателя XML.

При создании программы для Flex была изучена возможность применения начальных состояний. Применение данных состояний значительно упростило написание программы для Flex.

При написании грамматики для YACC делался упор на создание грамматики без конфликтов. В результате количество конфликтов в грамматике было сведено к четырем shift/reduce конфликтом. Конфликты возникли в ходе добавления правил для обработки ошибок.

Программа на C++ не только осуществляет вызов функции yyparse и предоставляет реализацию функции yyerror, но и содержит код для проверки некоторых ошибочных состояний.

Я старался сделать так, чтобы распознаватель не завершал работу после первой ошибки, а дальше анализировал переданный файл.

# Распознаваемые ошибки

Ошибки, распознаваемые в программе для Flex:

* незакрытый комментарий;
* незакрытая секция CDATA;
* некорректный символ в данном контексте;
* последовательность символов «--» внутри комментария;
* незакрытые одинарные кавычки;
* незакрытые двойные кавычки;
* пустой файл;
* неверное расположение комментария;

Ошибки, распознаваемые в грамматике для YACC:

* более одного корневого элемента;
* отсутствие корневого элемента;
* XML пролог без атрибутов;
* нарушение структуры элемента;
* нарушение структуры открывающего тега элемента;
* нарушение структуры закрывающего тега элемента;
* нарушение структуры атрибута;
* неизвестная синтаксическая ошибка;

Ошибки, распознаваемые в программе на С++:

* незакрытый открывающий тег элемента;
* незакрытый закрывающий тег элемента;
* некорректное гнездование тегов элемента;
* некорректное имени элемента или атрибута;
* некорректный XML пролог;

# Тестирование распознавателя

Протестируем распознавание некоторых ошибок:

Код XML документа для теста распознавания незакрытого комментария:

1. <?xml version="1.0" encoding="UTF-8"?>
2. <note>
3. <to>Tove</to>
4. <!-- not closed comment
5. <body>Don't forget me **this** weekend!</body>
6. </note>

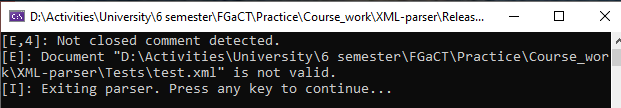


Рисунок 1 - сообщения распознавателя.

Код XML документа для теста распознавания незакрытой секции CDATA:

1. <?xml version="1.0" encoding="UTF-8"?>
2. <note>
3. <to>Tove</to>
4. <![CDATA[
5. <body>Don't forget me **this** weekend!</body>
6. </note>

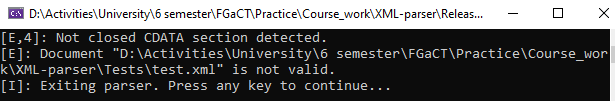


Рисунок 2 - сообщения распознавателя.

Код XML документа для теста распознавания последовательность символов «--» внутри комментария:

1. <?xml version="1.0" encoding="UTF-8"?>
2. <note>
3. <to>Tove</to>
4. <!-- "--" -->
5. <body>Don't forget me **this** weekend!</body>
6. </note>

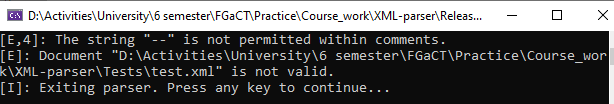


Рисунок 3 - сообщения распознавателя.

Код XML документа для теста распознавания более одного корневого элемента:

1. <?xml version="1.0" encoding="UTF-8"?>
2. <root\_1>
3. <to>Tove</to>
4. <!-- "comment" -->
5. <body>Don't forget me **this** weekend!</body>
6. </root\_1>
7. <root\_2>
8. </root\_2>

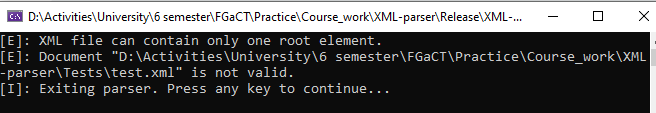


Рисунок 4 - сообщения распознавателя.

Код XML документа для теста распознавания незакрытого открывающего тега элемента:

1. <?xml version="1.0" encoding="UTF-8"?>
2. <root\_1>
3. <NOT\_CLOSED\_OPEN\_TAG>Tove
4. <!-- "comment" -->
5. <body>Don't forget me **this** weekend!</body>
6. </root\_1>

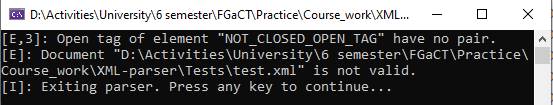


Рисунок 5 - сообщения распознавателя.

Код XML документа для теста распознавания незакрытого закрывающего тега элемента:

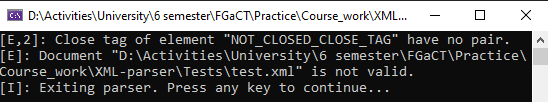


Рисунок 6 - сообщения распознавателя.

Код XML документа для теста распознавания некорректного гнездования элементов;

1. <root>Text
2. <improperly\_nested\_1>Text
3. <improperly\_nested\_2>Text
4. </improperly\_nested\_1>Text
5. </improperly\_nested\_2>Text
6. </root>

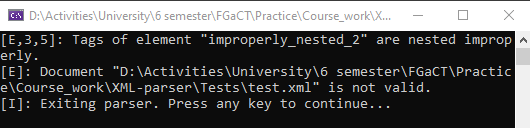


Рисунок 7 - сообщения распознавателя.

Здесь, можно считать элементы improperly\_nested\_1 и improperly\_nested\_2 некорректно расположенными только по отдельности. Если мы уберем элемент improperly\_nested\_2, то элемент improperly\_nested\_1 окажется корректно расположенным, аналогично для improperly\_nested\_2. Следовательно ошибка должна быть связана с одним из элементов и только одним, без разницы каким.

Проведем тест, в котором будет несколько ошибок (некорректное гнездования элементов, незакрытый открывающий тег элемента, незакрытый закрывающий тег элемента, неверное расположение комментария, более одного корневого элемента, незакрытый комментарий, некорректные значения атрибутов XML пролога, незакрытые двойные кавычки).

Код XML документа для теста:

1. <?xml version="trash"?>
2. <root\_1>Text
3. </not\_closed\_close\_tag\_1>Text
4. <element\_1>Text
5. <improperly\_nested\_1>Text
6. <improperly\_nested\_2>Text
7. </improperly\_nested\_1>Text
8. </not\_closed\_close\_tag\_2>
9. </improperly\_nested\_2>Text
10. </element\_1>Text
11. <element\_2>Text
12. <improperly\_nested\_3 attribute="value" >Text
13. <improperly\_nested\_4>Text
14. <not\_closed\_open\_tag\_1>Text
15. </improperly\_nested\_3>Text
16. </improperly\_nested\_4>Text
17. </element\_2>Text
18. <not\_closed\_open\_tag\_2>Text
19. </root\_1>
20. <root\_2>Text
21. <element\_3>Text
22. </element\_3>Text
23. </root\_2>
24. <!-- -- -->

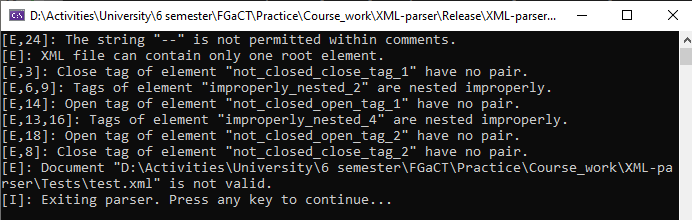


Рисунок 8 - сообщения распознавателя.

Заключение

В ходе выполнения курсовой работы был написан распознаватель расширяемого языка разметки XML. Были выполненные поставленные во введении задачи. К сожалению, ввиду недостатка опыта, не удалось создать грамматику без конфликтов. Добавление обработки ошибок привело к возникновению 4-х shift/reduce ошибок.

Я не могу однозначно утверждать, что с помощью утилит YACC и Flex возможно разрабатывать распознаватели современных формальных языков, но, по моим ощущения, с помощью данных утилит возможно создавать грамматики для простых современных языков.

Список использованных источников

1. XML [электронный ресурс], режим доступа: <https://ru.wikipedia.org/wiki/XML>, свободный.
2. XML Tutorial [электронный ресурс], режим доступа: <https://www.w3schools.com/xml/default.asp>, свободный.

Приложение 1

%option noyywrap

%option stack

%{

#include <stdbool.h>

#include <stdlib.h>

#include "C\_str.h"

#include "C\_Attribute.h"

#include "C\_Attributes.h"

#include "TagType.h"

#include "y.tab.h"

int g\_line = 1;

int commentStartLine;

int cdataStartLine;

int quotedStartLine;

bool rootIsScanned = false;

char Input();

void Unput(char c);

#ifndef unix

#define fileno \_fileno

#endif

%}

%x PROLOG

%x BODY

%x DOUBLE\_QUOTED

%x SINGLE\_QUOTED

%x OPEN\_TAG

%x CLOSE\_TAG

%x EXPECTING\_ROOT

%x EXPECTING\_NAME

%x COMMENT

%x CDATA

%x DOCTYPE

digit [0-9]

float {digit}+\.{digit}+

whiteSpace [ \t\r\n]

name [a-zA-Z\_]+[:a-zA-Z\_0-9\.\-]\*

escapedCharacter (&amp;|&lt;|&gt;|&quot;|&apos;)

singleQuotedText ({escapedCharacter}|[^<&'])\*

doubleQuotedText ({escapedCharacter}|[^<&"])\*

text (&amp;|&lt;|&gt;|&quot;|&apos;|[^<& \t\r\n])\*

commentOpenTag \<!\-\-

cdataOpenTag \<\!\[CDATA\[

%%

<INITIAL>[^\<]+ {

bool isTrash = false;

int trashStartLine;

for(int i=0; yytext[i] != 0; i++)

{

if (yytext[i] == '\n')

g\_line++;

else if (( isTrash == false ) &&

( yytext[i] != ' ' ) &&

( yytext[i] != '\t' ) &&

( yytext[i] != '\r' ))

{

isTrash = true;

trashStartLine = g\_line;

}

}

if(isTrash)

PrintError("[E,%d]: Trash data in the beggining of the file.\n", trashStartLine);

}

<INITIAL>\< {

char c = Input();

if( c == EOF)

{

PrintError("[E]: unexpected end of file\n");

Exit(-1);

}

if(c == '?')

{

rootIsScanned = true;

yy\_push\_state( PROLOG );

yy\_push\_state( EXPECTING\_NAME );

return K\_PROLOG\_OPEN\_BR;

}

else

{

Unput( c );

Unput( '<' );

yy\_push\_state( EXPECTING\_ROOT );

}

}

<INITIAL><<EOF>> {

PrintError("[E]: File is empty or doesn't contain any XML data.\n");

Exit(-1);

}

<EXPECTING\_ROOT>\< {

yy\_push\_state( BODY );

yy\_push\_state( OPEN\_TAG );

yy\_push\_state( EXPECTING\_NAME );

return K\_ELEM\_OPEN\_TAG\_OPEN\_BRACKET;

}

<EXPECTING\_ROOT><<EOF>> {

if(!rootIsScanned)

{

PrintError("[E]: File is empty.\n");

Exit(-1);

}

}

<EXPECTING\_ROOT>\<\!DOCTYPE {

yy\_push\_state( DOCTYPE );

return K\_DOCTYPE\_START;

}

<PROLOG>\?\> {

yy\_push\_state( EXPECTING\_ROOT );

return K\_PROLOG\_CLOSE\_BR;

}

<DOUBLE\_QUOTED>\" {

yy\_pop\_state();

quotedStartLine = g\_line;

return \*yytext;

}

<DOUBLE\_QUOTED><<EOF>> {

PrintError("[E,%d]: Not closed double quoted section detected.\n", quotedStartLine);

Exit(-1);

}

<DOUBLE\_QUOTED>{doubleQuotedText} {

str\_Init( & yylval.type\_C\_str, yytext);

yylval.type\_C\_str.line = g\_line;

return ATTRIBUTE\_VALUE;

}

<SINGLE\_QUOTED>\' {

yy\_pop\_state();

return \*yytext;

}

<SINGLE\_QUOTED>{singleQuotedText} {

str\_Init( & yylval.type\_C\_str, yytext);

yylval.type\_C\_str.line = g\_line;

return ATTRIBUTE\_VALUE;

}

<SINGLE\_QUOTED><<EOF>> {

PrintError("[E,%d]: Not closed single quoted section detected.\n", quotedStartLine);

Exit(-1);

}

<BODY>{cdataOpenTag} {

cdataStartLine = g\_line;

yy\_push\_state( CDATA );

}

<BODY,OPEN\_TAG,CLOSE\_TAG>\< {

yy\_push\_state( BODY );

yy\_push\_state( OPEN\_TAG );

yy\_push\_state( EXPECTING\_NAME );

return K\_ELEM\_OPEN\_TAG\_OPEN\_BRACKET;

}

<BODY,EXPECTING\_ROOT,OPEN\_TAG,CLOSE\_TAG>\<\/ {

yy\_push\_state( CLOSE\_TAG );

yy\_push\_state( EXPECTING\_NAME );

return K\_ELEM\_CLOSE\_TAG\_OPEN\_BRACKET;

}

<BODY>{text} {;}

<BODY>& {

PrintError("[E,%d]: \"&\" is not escaped or is part of not existing escape sequence.\n", g\_line);

}

<CDATA>[^\]] {;}

<CDATA>\]\] {;}

<CDATA>\] {;}

<CDATA><<EOF>> {

PrintError("[E,%d]: Not closed CDATA section detected.\n", cdataStartLine);

Exit(-1);

}

<CDATA>\]\]\> {

yy\_pop\_state();

}

<OPEN\_TAG>\/\> {

yy\_pop\_state();

return K\_ELEM\_SELF\_CLOSING\_TAG;

}

<COMMENT>\-\- {

PrintError("[E,%d]: The string \"--\" is not permitted within comments.\n", g\_line);

Unput('-');

}

<COMMENT><<EOF>> {

PrintError("[E,%d]: Not closed comment detected.\n", commentStartLine);

Exit(-1);

}

<COMMENT>\-\-\> {

yy\_pop\_state();

}

<COMMENT>[^\-] {;}

<COMMENT>\- {;}

<EXPECTING\_NAME>{name} {

str\_Init( & yylval.type\_C\_str, yytext);

yylval.type\_C\_str.line = g\_line;

yy\_pop\_state();

return NAME;

}

<EXPECTING\_NAME>. {

PrintError("[E,%d]: Invalid symbol \"%c\" for start of the name.\n", g\_line, \*yytext);

}

<EXPECTING\_ROOT,BODY>{commentOpenTag} {

commentStartLine = g\_line;

yy\_push\_state( COMMENT );

}

<OPEN\_TAG,CLOSE\_TAG,DOCTYPE>\> {

yy\_pop\_state();

return K\_ELEM\_TAG\_CLOSE\_BRACKET;

}

<OPEN\_TAG,CLOSE\_TAG,PROLOG>: { return \*yytext; }

<PROLOG,OPEN\_TAG,CLOSE\_TAG,DOCTYPE>{name} {

str\_Init( & yylval.type\_C\_str, yytext);

yylval.type\_C\_str.line = g\_line;

return NAME;

}

<PROLOG,OPEN\_TAG,CLOSE\_TAG>[=] { return \*yytext; }

<PROLOG,OPEN\_TAG,CLOSE\_TAG>\" {

yy\_push\_state( DOUBLE\_QUOTED );

return \*yytext;

}

<PROLOG,OPEN\_TAG,CLOSE\_TAG>\' {

yy\_push\_state( SINGLE\_QUOTED );

return \*yytext;

}

<CDATA,EXPECTING\_NAME,EXPECTING\_ROOT,COMMENT,PROLOG,OPEN\_TAG,CLOSE\_TAG,BODY,DOCTYPE>{whiteSpace}+ {

g\_line += CountChar(yytext, '\n');

}

<PROLOG,DOUBLE\_QUOTED,SINGLE\_QUOTED,OPEN\_TAG,CLOSE\_TAG,EXPECTING\_NAME,DOCTYPE>{commentOpenTag} {

commentStartLine = g\_line;

PrintError("[E,%d]: Comments are allowed only inside of element body or just after prologue.\n", g\_line);

yy\_push\_state( COMMENT );

}

<INITIAL,EXPECTING\_ROOT,PROLOG,DOUBLE\_QUOTED,SINGLE\_QUOTED,OPEN\_TAG,CLOSE\_TAG,EXPECTING\_NAME,DOCTYPE>{cdataOpenTag} {

cdataStartLine = g\_line;

PrintError("[E,%d]: CDATA sections are allowed only inside of element body.\n", g\_line);

yy\_push\_state( CDATA );

}

<PROLOG,DOCTYPE,EXPECTING\_NAME,BODY,DOUBLE\_QUOTED,SINGLE\_QUOTED,OPEN\_TAG,CLOSE\_TAG,EXPECTING\_ROOT,COMMENT,CDATA>. {

PrintError("[E,%d]: Wrong place for symbol \"%c\".\n", g\_line, \*yytext);

}

%%

char Input()

{

char c = input();

if(c == '\n')

{

g\_line++;

}

return c;

}

void Unput(char c)

{

if(c == '\n')

{

g\_line--;

}

unput( c );

}

Приложение 2

%{

#include <stdbool.h>

#include "C\_str.h"

#include "C\_Attribute.h"

#include "C\_Attributes.h"

#include "TagType.h"

#ifdef \_DEBUG

#define YYDEBUG 1

yydebug = 1;

#endif

extern int g\_unknownSyntaxErrorQnt;

extern bool g\_checkNesting;

extern int g\_line;

%}

%union {

char \* type\_cstr;

float type\_float;

struct C\_str type\_C\_str;

struct C\_Attribute type\_C\_Attribute;

struct C\_Attributes type\_C\_Attributes;

}

%token <type\_C\_str> ATTRIBUTE\_VALUE

%token <type\_C\_str> NAME // Element or attribute name.

// "K" means Keyword.

%token <type\_cstr> K\_PROLOG\_OPEN\_BR // "<?"

%token <type\_cstr> K\_PROLOG\_CLOSE\_BR // "?>"

%token <type\_cstr> K\_DOCTYPE\_START // "<!DOCTYPE"

%token <type\_cstr> K\_ELEM\_OPEN\_TAG\_OPEN\_BRACKET // "<"

%token <type\_cstr> K\_ELEM\_CLOSE\_TAG\_OPEN\_BRACKET // "</"

%token <type\_cstr> K\_ELEM\_TAG\_CLOSE\_BRACKET // ">"

%token <type\_cstr> K\_ELEM\_SELF\_CLOSING\_TAG // "/>"

%type <type\_C\_Attribute> attribute

%type <type\_C\_Attribute> attributes

%%

xml\_file: doctype element

| xml\_prolog doctype element

| doctype element tags

| xml\_prolog doctype element tags

| doctype tags

| doctype element elements

{

PrintError("[E]: XML file can contain only one root element.\n");

}

| xml\_prolog doctype element elements

{

PrintError("[E]: XML file can contain only one root element.\n");

}

| doctype element elements tags

{

PrintError("[E]: XML file can contain only one root element.\n");

}

| xml\_prolog doctype element elements tags

{

PrintError("[E]: XML file can contain only one root element.\n");

}

| xml\_prolog doctype

{

PrintError("[E]: XML file must contain root element.\n");

}

| element

| xml\_prolog element

| element tags

| xml\_prolog element tags

| xml\_prolog close\_tag tags

| xml\_prolog doctype close\_tag tags

| doctype close\_tag tags

| close\_tag tags

| element elements

{

PrintError("[E]: XML file can contain only one root element.\n");

}

| xml\_prolog element elements

{

PrintError("[E]: XML file can contain only one root element.\n");

}

| element elements tags

{

PrintError("[E]: XML file can contain only one root element.\n");

}

| xml\_prolog element elements tags

{

PrintError("[E]: XML file can contain only one root element.\n");

}

| xml\_prolog

{

PrintError("[E]: XML file must contain root element.\n");

}

;

xml\_prolog: K\_PROLOG\_OPEN\_BR NAME attributes K\_PROLOG\_CLOSE\_BR

{

ValidateProlog($2, & $3);

free\_C\_str(& $2);

}

| K\_PROLOG\_OPEN\_BR NAME K\_PROLOG\_CLOSE\_BR

{

PrintError("[E]: XML prolog must contain attributes.\n");

}

;

doctype: K\_DOCTYPE\_START NAME K\_ELEM\_TAG\_CLOSE\_BRACKET

element: self\_closing\_tag

| open\_tag close\_tag

| open\_tag element\_body close\_tag

| open\_tag error

{

g\_unknownSyntaxErrorQnt--;

yyerrok;

}

| open\_tag element\_body error

{

g\_unknownSyntaxErrorQnt--;

yyerrok;

}

;

elements: element

| elements element

;

element\_body: element

| element\_body element

;

self\_closing\_tag: K\_ELEM\_OPEN\_TAG\_OPEN\_BRACKET NAME K\_ELEM\_SELF\_CLOSING\_TAG

{

ValidateName($2);

free\_C\_str(& $2);

}

| K\_ELEM\_OPEN\_TAG\_OPEN\_BRACKET NAME attributes K\_ELEM\_SELF\_CLOSING\_TAG

{

ValidateName($2);

Validate\_C\_Attributes( & $3);

free\_C\_str(& $2);

}

;

open\_tag: K\_ELEM\_OPEN\_TAG\_OPEN\_BRACKET NAME K\_ELEM\_TAG\_CLOSE\_BRACKET

{

ValidateName($2);

AddToTagList($2, Open);

free\_C\_str(& $2);

}

| K\_ELEM\_OPEN\_TAG\_OPEN\_BRACKET NAME attributes K\_ELEM\_TAG\_CLOSE\_BRACKET

{

ValidateName($2);

Validate\_C\_Attributes( & $3);

AddToTagList($2, Open);

free\_C\_str(& $2);

}

| K\_ELEM\_OPEN\_TAG\_OPEN\_BRACKET NAME attributes error

{

ValidateName($2);

Validate\_C\_Attributes( & $3);

PrintError("[E,%d]: (Critical error) Open tag of element \"",$2.line);

PrintStr($2);

PrintError("\" structure error.\n");

g\_checkNesting = false;

g\_unknownSyntaxErrorQnt--;

free\_C\_str(& $2);

yyerrok;

Exit(-1);

}

| K\_ELEM\_OPEN\_TAG\_OPEN\_BRACKET NAME error K\_ELEM\_TAG\_CLOSE\_BRACKET

{

ValidateName($2);

PrintError("[E,%d]: (Critical error) Open tag of element \"",$2.line);

PrintStr($2);

PrintError("\" structure error.\n");

g\_checkNesting = false;

g\_unknownSyntaxErrorQnt--;

free\_C\_str(& $2);

yyerrok;

Exit(-1);

}

;

close\_tag: K\_ELEM\_CLOSE\_TAG\_OPEN\_BRACKET NAME K\_ELEM\_TAG\_CLOSE\_BRACKET

{

ValidateName($2);

AddToTagList($2, Close);

free\_C\_str(& $2);

}

| K\_ELEM\_CLOSE\_TAG\_OPEN\_BRACKET NAME error

{

ValidateName($2);

PrintError("[E,%d]: Close tag of element \"",$2.line);

PrintStr($2);

PrintError("\" doesn't have \">\".\n");

g\_unknownSyntaxErrorQnt--;

g\_checkNesting = false;

free\_C\_str(& $2);

yyerrok;

}

tags: close\_tag

| open\_tag

| tags open\_tag

| tags close\_tag

;

attributes: attribute

{

Init\_C\_Atributes(& $$);

Add\_C\_Attribute(& $1, & $$);

free\_C\_str(& $1.name);

free\_C\_str(& $1.value);

}

| attributes attribute

{

Add\_C\_Attribute(& $2, & $1);

free\_C\_str(& $2.name);

free\_C\_str(& $2.value);

}

;

attribute: NAME '=' '\'' ATTRIBUTE\_VALUE '\''

{

$$.name = $1;

$$.value = $4;

}

| NAME '=' '"' ATTRIBUTE\_VALUE '"'

{

$$.name = $1;

$$.value = $4;

}

| NAME '=' error

{

PrintError("[E,%d]: Attribute value must be quoted with \" or \' quotes.\n", $1.line);

$$.name = $1;

str\_Init( & $$.value, "error");

g\_unknownSyntaxErrorQnt--;

yyerrok;

}

;

%%

Приложение 3

XML-parser.cpp:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <string>

#include <list>

#include "TagType.h"

#include "Utility.hpp"

using namespace std;

bool g\_errorOccured = false;

extern "C" int g\_unknownSyntaxErrorQnt = 0;

extern "C" bool g\_checkNesting = true;

list<Tag> tags;

string fileToParse = "";

extern "C"

{

extern int g\_line;

extern FILE\* yyin;

extern int yyparse();

void Exit(int exitCode);

void yyerror(const char\* s)

{

// printf("[E,%d]: %s.\n", g\_line, s);

g\_errorOccured = true;

g\_unknownSyntaxErrorQnt++;

}

}

int main(int argc, char \*\* argv)

{

if (argc != 2)

{

printf("[E]: File path not specified.\n");

Exit(-1);

}

fileToParse.assign(argv[1]);

yyin = fopen(fileToParse.c\_str(), "r");

if (yyin == NULL)

{

printf("[E]: Can't open file: %s\n", fileToParse.c\_str());

Exit(-1);

}

yyparse();

fclose(yyin);

if (g\_checkNesting)

{

ValidateTagsNesting(tags);

}

Exit(1);

}

Utility.cpp:

#ifndef unix

#define \_CRT\_SECURE\_NO\_WARNINGS

#endif

#include <string>

#include <utility>

#ifndef unix

#include <conio.h>

#endif

#include <stdio.h>

#include <stdlib.h>

#include <stdarg.h>

#include <string.h>

#include "C\_str.h"

#include "C\_Attributes.h"

#include "TagType.h"

#include "Utility.hpp"

#include "Attribute.hpp"

#ifdef unix

#define \_stricmp strcasecmp

#endif

using namespace std;

extern string fileToParse;

extern "C" extern int g\_unknownSyntaxErrorQnt;

extern "C" void PrintError(char const\* const \_Format, ...)

{

va\_list args;

va\_start(args, \_Format);

vfprintf(stderr, \_Format, args);

g\_errorOccured = true;

}

extern "C" int CountChar(char\* str, char c)

{

int occurencyCounter = 0;

for (int i = 0; str[i] != 0; i++)

{

if (str[i] == c)

{

occurencyCounter++;

}

}

return occurencyCounter;

}

extern "C" bool IsWhiteSpace(char\* str)

{

for (int i = 0; str[i] != 0; i++)

{

if ((str[i] != ' ') &&

(str[i] != '\t') &&

(str[i] != '\n') &&

(str[i] != '\r'))

{

return false;

}

}

return true;

}

extern "C" void Exit(int exitCode)

{

int \_exitCode = exitCode;

if (g\_errorOccured)

{

printf("[E]: Document \"%s\" is not valid.\n", fileToParse.c\_str());

if (g\_unknownSyntaxErrorQnt)

{

printf("[E]: %d unknown syntax error(s) ocured.\n", g\_unknownSyntaxErrorQnt);

}

\_exitCode = -1;

}

else

{

if (fileToParse != "" )

{

printf("[I]: Document \"%s\" is valid.\n", fileToParse.c\_str());

}

}

#ifndef unix

printf("[I]: Exiting parser. Press any key to continue...\n\n");

\_getch();

#else

printf("\n");

#endif

exit(\_exitCode);

}

extern "C" void PrintStr(C\_str s)

{

printf("%s", reinterpret\_cast<string\*>(s.p\_string)->c\_str());

}

bool Tag::isPair(const Tag & other)

{

if (this->type != other.type &&

this->name == other.name)

{

return true;

}

return false;

}

TagType Tag::Type()

{

return this->type;

}

string Tag::Name()

{

return this->name;

}

bool Tag::operator==(const Tag& other)

{

if (this->name == other.name &&

this->type == other.type)

{

return true;

}

return false;

}

int Tag::Line()

{

return this->line;

}

extern "C" void AddToTagList(C\_str name, TagType type)

{

tags.push\_back(Tag( \* reinterpret\_cast<string\*>(name.p\_string),

type, name.line));

}

bool DeleteProperlyNestedTag(list<Tag>& tags)

{

for (auto it = tags.begin(); it != tags.end(); it++)

{

auto nextIt = it;

nextIt++;

if (nextIt == tags.end())

{

break;

}

if ((\*it).Type() == Open &&

(\*it).isPair(\*nextIt) == true)

{

tags.erase(it);

tags.erase(nextIt);

return true;

}

}

return false;

}

bool DeleteProperlyNestedTags(list<Tag>& tags)

{

bool deletedSome = false;

while (1)

{

if (DeleteProperlyNestedTag(tags))

{

deletedSome = true;

}

else

{

break;

}

}

return deletedSome;

}

pair<list<Tag>::iterator, list<Tag>::iterator>

FindImproperlyNestedOpenTag(list<Tag>& tags)

{

if (tags.empty())

{

return make\_pair(tags.end(), tags.end());

}

if (tags.size() == 1)

{

if ((\*tags.begin()).Type() == Open)

{

return make\_pair(tags.begin(), tags.end());

}

else

{

return make\_pair(tags.end(), tags.end());

}

}

auto it = tags.begin();

auto nextIt = it;

nextIt++;

while (nextIt != tags.end())

{

if ( (\*it).Type() == Open &&

(\*nextIt).Type() == Close)

{

if (! (\*it).isPair(\*nextIt))

{

return make\_pair(it, nextIt);

}

}

it++;

nextIt++;

}

if ((\*it).Type() == Open)

{

return make\_pair(it, tags.end());

}

return make\_pair(tags.end(), tags.end());

}

pair<list<Tag>::iterator, list<Tag>::iterator>

FindImproperlyNestedCloseTag(list<Tag>& tags)

{

if (tags.empty())

{

return make\_pair(tags.end(), tags.end());

}

if ((\*tags.begin()).Type() == Close)

{

return make\_pair(tags.begin(), tags.end());

}

if (tags.size() == 1)

{

return make\_pair(tags.end(), tags.end());

}

auto previousIt = tags.end();

previousIt--;

auto it = previousIt;

previousIt--;

while (1)

{

if ((\*it).Type() == Close &&

(\*previousIt).Type() == Open)

{

if (!(\*it).isPair(\*previousIt))

{

return make\_pair(it, previousIt);

}

}

if (previousIt == tags.begin())

{

break;

}

it--;

previousIt--;

}

return make\_pair(tags.end(), tags.end());

}

list<Tag>::iterator FindCloseTagPair(list<Tag>::iterator closeTagIt, list<Tag>& tags)

{

if ( closeTagIt == tags.begin() ||

closeTagIt == tags.end() )

{

return tags.end();

}

auto expectantIt = closeTagIt;

bool lastIteratorChecked = false;

while (!lastIteratorChecked)

{

expectantIt--;

if (\*closeTagIt == \*expectantIt)

{

break;

}

if ((\*closeTagIt).isPair(\*expectantIt))

{

return expectantIt;

}

if (expectantIt == tags.begin())

{

lastIteratorChecked = true;

}

}

return tags.end();

}

list<Tag>::iterator FindOpenTagPair(list<Tag>::iterator openTagIt, list<Tag>& tags)

{

if (openTagIt == tags.end())

{

return tags.end();

}

auto it = openTagIt;

it++;

for(auto expectantIt = it; expectantIt != tags.end(); expectantIt++)

{

if (\*openTagIt == \*expectantIt)

{

break;

}

if ((\*openTagIt).isPair(\*expectantIt))

{

return expectantIt;

}

}

return tags.end();

}

extern "C" bool ValidateName(C\_str name)

{

return ValidateName( \* reinterpret\_cast<string \*>(name.p\_string), name.line);

}

void ValidateTagsNesting(list<Tag> & tags)

{

while (!tags.empty())

{

DeleteProperlyNestedTags(tags);

if (tags.empty())

{

break;

}

while (1)

{

auto pair = FindImproperlyNestedOpenTag(tags);

auto improperlyNestedOpenTagIt = pair.first;

if (improperlyNestedOpenTagIt == tags.end())

{

break;

}

auto closeTagIt = FindOpenTagPair(improperlyNestedOpenTagIt, tags);

if (closeTagIt == tags.end())

{

PrintError("[E,%d]: Open tag of element \"%s\" have no pair.\n", (\*improperlyNestedOpenTagIt).Line(), (\*improperlyNestedOpenTagIt).Name().c\_str());

tags.erase(improperlyNestedOpenTagIt);

}

else

{

if ( FindCloseTagPair(pair.second, tags) != tags.end())

{

PrintError("[E,%d,%d]: Tags of element \"%s\" are nested improperly.\n",

(\*improperlyNestedOpenTagIt).Line(), (\*closeTagIt).Line(), (\*closeTagIt).Name().c\_str());

tags.erase(improperlyNestedOpenTagIt);

tags.erase(closeTagIt);

}

else

{

PrintError("[E,%d]: Close tag of element \"%s\" have no pair.\n", (\*pair.second).Line(), (\*pair.second).Name().c\_str());

tags.erase(pair.second);

}

}

}

while (1)

{

auto pair = FindImproperlyNestedCloseTag(tags);

auto improperlyNestedCloseTagIt = pair.first;

if (improperlyNestedCloseTagIt == tags.end())

{

break;

}

auto openTagIt = FindCloseTagPair(improperlyNestedCloseTagIt, tags);

if (openTagIt == tags.end())

{

PrintError("[E,%d]: Close tag of element \"%s\" have no pair.\n", (\*improperlyNestedCloseTagIt).Line(), (\*improperlyNestedCloseTagIt).Name().c\_str());

tags.erase(improperlyNestedCloseTagIt);

}

else

{

if (FindOpenTagPair(pair.second, tags) != tags.end())

{

PrintError("[E,%d,%d]: Tags of element \"%s\" are nested improperly.\n",

(\*improperlyNestedCloseTagIt).Line(), (\*openTagIt).Line(), (\*openTagIt).Name().c\_str());

tags.erase(improperlyNestedCloseTagIt);

tags.erase(openTagIt);

}

else

{

PrintError("[E,%d]: Open tag of element \"%s\" have no pair.\n", (\*pair.second).Line(), (\*pair.second).Name().c\_str());

tags.erase(pair.second);

}

}

}

}

}

bool ValidateName(string name, int line)

{

if (name.size() < 3)

{

return true;

}

char \_1st\_3\_chars[4] = { 0 };

strncpy(\_1st\_3\_chars, name.c\_str(), 3);

if (\_stricmp(\_1st\_3\_chars, "xml") == 0)

{

PrintError("[E,%d]: name \"%s\" contains \"[xX][mM][lL]\" pattern in the beggining of the string.\n", line, name.c\_str());

return false;

}

return false;

}

bool ValidateAtrributeList(list<Attribute> & attributes)

{

bool errorOccured = false;

for (auto& attribute : attributes)

{

if (!ValidateName(attribute.name(), attribute.nameLine()))

{

errorOccured = true;

}

}

return errorOccured;

}

extern "C" bool ValidateProlog(C\_str \_name, C\_Attributes \* c\_attributes)

{

string& name = \*reinterpret\_cast<string\*>(\_name.p\_string);

int nameLine = \_name.line;

list<Attribute> & attributes = \*reinterpret\_cast<list<Attribute>\*>(c\_attributes->p\_list);

bool errorOccured = false;

if (attributes.size() > 3)

{

PrintError("[E,%d]: Prolog can contain only \"version\", \"encoding\" and \"standalone\" atttributes.\n", nameLine);

return false;

}

if (name != "xml")

{

PrintError("[E,%d]: Prolog must start with xml word.\n", nameLine);

errorOccured = true;

}

auto it = attributes.begin();

if ((\*it).name() != "version")

{

PrintError("[E,%d]: Prolog 1st attribute must be \"version\".\n", (\*it).nameLine());

errorOccured = true;

}

else if ((\*it).value() != "1.0")

{

PrintError("[E,%d]: Prolog \"version\" attribute can only contain \"1.0\".\n", (\*it).valueLine());

errorOccured = true;

}

it++;

if (it == attributes.end())

{

return errorOccured;

}

if ((\*it).name() != "encoding")

{

PrintError("[E,%d]: Prolog 2d attribute must be \"encoding\".\n", (\*it).nameLine());

errorOccured = true;

}

else if (\_stricmp((\*it).value().c\_str(), "ASCII") != 0 &&

\_stricmp((\*it).value().c\_str(), "Windows-1252") != 0 &&

\_stricmp((\*it).value().c\_str(), "UTF-8") != 0 &&

\_stricmp((\*it).value().c\_str(), "ISO-8859-1") != 0)

{

PrintError("[E,%d]: Prolog \"encoding\" attribute can only contain \"ASCII\", \

\"Windows-1252\", \"UTF-8\", \"ISO-8859-1\" in any case.\n", (\*it).valueLine());

errorOccured = true;

}

it++;

if (it == attributes.end())

{

return errorOccured;

}

if ((\*it).name() != "standalone")

{

PrintError("[E,%d]: Prolog 3d attribute must be \"standalone\".\n", (\*it).nameLine());

errorOccured = true;

}

else if ((\*it).value() != "yes" &&

(\*it).value() != "no")

{

PrintError("[E,%d]: Prolog \"standalone\" attribute can only contain \"yes\" or \"no\".\n", (\*it).valueLine());

errorOccured = true;

}

return errorOccured;

}

C\_str.cpp:

#include <string>

#include "C\_str.h"

using namespace std;

extern "C" void str\_Init(C\_str \* p\_str, const char \* initValue)

{

p\_str->p\_string = new string(initValue);

}

extern "C" void str\_Destroy(C\_str \* p\_str)

{

delete(p\_str->p\_string);

}

extern "C" void str\_PushBack(C\_str \* p\_str, char c)

{

string\* p\_cpp\_str = reinterpret\_cast<string \*>(p\_str->p\_string);

p\_cpp\_str->push\_back(c);

}

extern "C" void str\_Assign(struct C\_str\* p\_str, char \* value)

{

string\* p\_cpp\_str = reinterpret\_cast<string\*>(p\_str->p\_string);

p\_cpp\_str->assign(value);

}

extern "C" void free\_C\_str(struct C\_str\* p\_str)

{

delete((string\*)p\_str->p\_string);

}

#include <list>

#include "C\_Attribute.h"

#include "C\_Attributes.h"

#include "Attribute.hpp"

#include "Utility.hpp"

extern "C" void Init\_C\_Atributes(C\_Attributes \* attributes)

{

attributes->p\_list = new list<Attribute>();

}

extern "C" void Add\_C\_Attribute(C\_Attribute \* c\_attr, C\_Attributes \* attributes)

{

list<Attribute>\* p\_attributeList = reinterpret\_cast<list<Attribute>\*>(attributes->p\_list);

string\* p\_name = reinterpret\_cast<string\*>(c\_attr->name.p\_string);

string\* p\_value = reinterpret\_cast<string\*>(c\_attr->value.p\_string);

p\_attributeList->push\_back(Attribute(\*p\_name, c\_attr->name.line, \*p\_value, c\_attr->value.line));

}

extern "C" bool Validate\_C\_Attributes(C\_Attributes \* c\_attributes)

{

return ValidateAtrributeList( \* reinterpret\_cast<list<Attribute> \*>( c\_attributes->p\_list ));

}

Utility.hpp:

#include <string>

#include <list>

#include <stdio.h>

#include <stdlib.h>

#include <stdarg.h>

#include "TagType.h"

#include "C\_str.h"

#include "C\_Attribute.h"

#include "C\_Attributes.h"

#include "Attribute.hpp"

using namespace std;

extern bool g\_errorOccured;

class Tag

{

public:

Tag(string name, TagType type, int line) : name(name), type(type), line(line) {}

bool isPair(const Tag & other);

TagType Type();

string Name();

int Line();

bool operator==(const Tag& other);

private:

string name;

TagType type;

int line;

};

extern list<Tag> tags;

extern "C" void PrintError(char const\* const \_Format, ...);

extern "C" int CountChar(char\* str, char c);

extern "C" bool IsWhiteSpace(char\* str);

extern "C" void Exit(int exitCode);

extern "C" void AddToTagList(C\_str name, TagType type);

extern "C" bool ValidateName(C\_str name);

extern "C" bool ValidateProlog(C\_str name, C\_Attributes \* c\_attributes);

void ValidateTagsNesting(list<Tag>& tags);

bool ValidateAtrributeList(list<Attribute> & attributes);

bool ValidateName(string name, int line);

TagType.h:

#pragma once

enum TagType

{

Open,

Close

};

C\_str.h:

#pragma once

struct C\_str

{

void\* p\_string;

int line;

};

C\_Attributes.h

#pragma once

#include "C\_Attribute.h"

struct C\_Attributes

{

void\* p\_list;

};

C\_Attribute.h

#pragma once

#include "C\_str.h"

struct C\_Attribute

{

struct C\_str name;

struct C\_str value;

};

Attribute.hpp:

#pragma once

#include <string>

using namespace std;

class Attribute

{

public:

Attribute(string name, int nameLine, string value, int valueLine)

: \_name(name), \_nameLine(nameLine), \_value(value), \_valueLine(valueLine) {}

string name() { return this->\_name; }

string value() { return this->\_value; }

int nameLine() { return this->\_nameLine; }

int valueLine() { return this->\_valueLine; }

private:

string \_name;

string \_value;

int \_nameLine;

int \_valueLine;

};