|  |  |
| --- | --- |
| Gerb-BMSTU_01 | ***«*Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_\_КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по домашнему заданию № 2**

# Дисциплина: Машинно-зависимые языки и основы компиляции

# Название домашнего задания: лексические и синтаксические анализаторы

Студент гр. ИУ6-42Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. Д. Шатский**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ C.C.Данилюк**

(Подпись, дата) (И.О. Фамилия)

Москва, 2023

**Вариант 2.23**

**Цель работы.** закрепление знаний теоретических основ и основных методов приемов разработки лексических и синтаксических анализаторов регулярных и контекстно-свободных формальных языков.

**Задание.** Разработать грамматику и распознаватель языка программирования C++ описаний структурных типов данных, включающего описание массивов, строк и структур. Предусмотреть следующие типы элементов: float, int, char, unsigned char. Например:

**int aer[8];**

**char record[80];**

**struct student { char name[22]; char family[22]; int old; } st1,ft;**

**Описание грамматики в форме Бэкуса-Наура.**

**<**vars> ::= <varstruct> ; <prog> | <varstruct> ;

<varstruct> ::= <var>|<struct>

<var> ::= <type> <identifier>[<number>]

<struct> ::= struct <identifier> { <vars> } <structidentifier>

<type> ::= float | int | char | unsigned char

<structidentifier> ::= <identifier> | <identifier>, <structidentifier>

<identifier> ::= <numberletter> | <numberletter> <identifier>

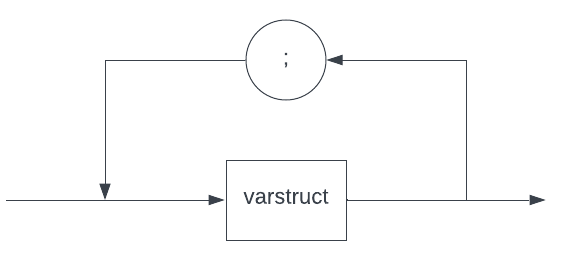
<numberletter> ::= <number>|<letter>

<number> ::= 0|...|9

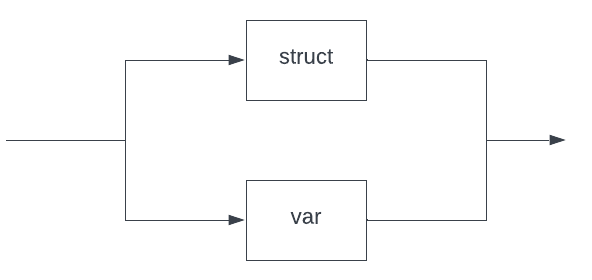
<letter> ::= a|...|z|A|...|Z

**Данная грамматика принадлежит к 2 типу по Хомскому.**

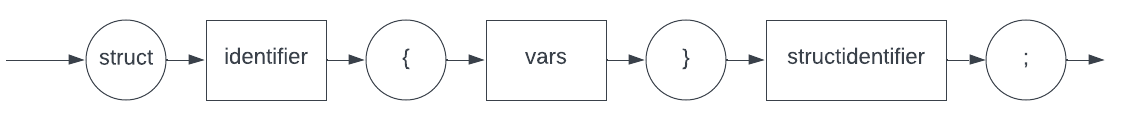
**Синтаксический анализ.**

****

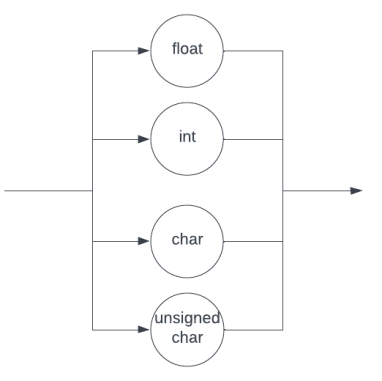
*Рисунок 1 – Синтаксическая диаграмма конструкции <vars>*

****

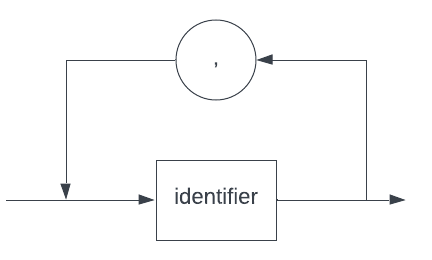
*Рисунок 2 – Синтаксическая диаграмма конструкции <varstruct>*

**

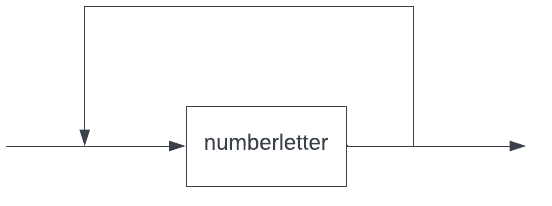
*Рисунок 3 – Синтаксическая диаграмма конструкции <struct>*

**

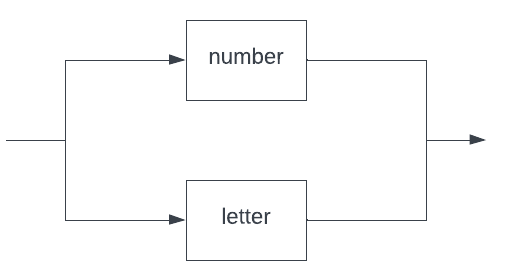
*Рисунок 4 – Синтаксическая диаграмма конструкции <type>*

**

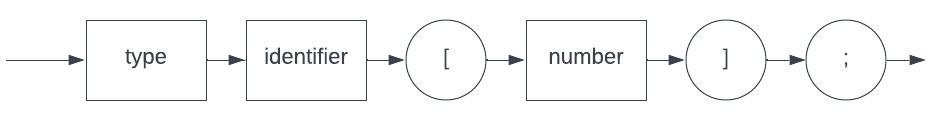
*Рисунок 5 – Синтаксическая диаграмма конструкции <structidentifier>*

**

*Рисунок 6 – Синтаксическая диаграмма конструкции <identifier>*

**

*Рисунок 7 – Синтаксическая диаграмма конструкции <numberletter>*

**

*Рисунок 8 – Синтаксическая диаграмма конструкции <var>*

**Программа.**

#include <iostream>

#include <cstring>

#include <algorithm>

#include <initializer\_list>

using namespace std;

void removeExtraSpaces(char\* inputs, char\* output)

{

int c = 0, d = 0;

while (inputs[c] != '\0') {

if (!(inputs[c] == ' ' && inputs[c + 1] == ' ')) {

output[d] = inputs[c];

d++;

}

c++;

}

output[d] = '\0';

}

void skipSpace(char\* string,int\* start)

{

if (string[\*start] == ' ')

{

\*start+=1;

}

}

void findWord(char\* string, char\* output)

{

char\* space = strchr(string, ' ');

char\* sqBracket = strchr(string, '[');

char\* rBracket = strchr(string, '{');

char\* strEnd = strchr(string, '\0');

char\* comma = strchr(string, ',');

char\* semicolon = strchr(string, ';');

size\_t fSpace = space - string;

size\_t fsqBracket = sqBracket - string;

size\_t frBracket = rBracket - string;

size\_t fstrEnd = strEnd - string;

size\_t fcomma = comma - string;

size\_t fsemicolon = semicolon - string;;

size\_t closest = min({fSpace,fsqBracket,frBracket,fstrEnd,fcomma,fsemicolon});

strncpy(output, string, closest);

}

void findNumber(char\* string, char\* output)

{

char\* sqBracket = strchr(string, ']');

size\_t fsqBracket = sqBracket - string;

size\_t closest = fsqBracket;

strncpy(output, string, closest);

}

int checkNumber(char\* str1) {

int i, x=0, p;

p=strlen(str1);

for (i = 0; i < p ; i++)

if ((str1[i] >= '0' && str1[i] <= '9')) {

continue;

} else return 0;

return 1;

}

int checkWord(char\* str1) {

int i, x=0, p;

p=strlen(str1);

for (i = 0; i < p ; i++)

if ((str1[i] >= 'a' && str1[i] <= 'z') || (str1[i] >= 'A' && str1[i] <= 'Z') || (str1[i] == ' ') || (str1[i] >= '0' && str1[i] <= '9')) {

continue;

} else return 0;

return 1;

}

void type(char\* string,bool\* correct,int\* start)

{

char\* stringpos = string+\*start;

bool foundType =false;

if (strncmp(stringpos,"float",5) == 0) {

cout << "Detected type float\n";

foundType =true;

\*start+=5;

}

if (strncmp(stringpos,"int",3) == 0) {

cout << "Detected type int\n";

foundType =true;

\*start+=3;

}

if (strncmp(stringpos,"char",4) == 0) {

cout << "Detected type char\n";

foundType =true;

\*start+=4;

}

if (strncmp(stringpos,"struct",6) == 0) {

cout << "Detected type struct\n";

foundType =true;

\*start+=6;

}

if (strncmp(stringpos,"unsigned",8) == 0) {

stringpos +=9;

if (strncmp(stringpos,"char",4) == 0) {

cout << "Detected type unsigned char\n";

foundType =true;

\*start+=13;

}

}

if (foundType) return;

char output[20] = "";

findWord(stringpos,output);

cout << "Incorrect variable type:\"" << output << "\"\n";

\*correct = false;

}

void ident(char\* string,bool\* correct,int\* start)

{

char\* stringpos = string+\*start;

char output[20] = "";

findWord(stringpos,output);

if (!checkWord(output)) {

cout << "Incorrect variable name:\"" << output << "\"\n";

\*correct = false;

}

else

{

cout << "Variable name:\"" << output << "\"\n";

\*start+=strlen(output);

}

}

void checksymbol(char\* string,bool\* correct,int\* start,const char\* symbol,const char\* correctin,const char\* incorrectin)

{

if (\*correct) {

char\* stringpos = string+\*start;

if (strncmp(stringpos,symbol,1) == 0) {

cout << "Detected symbol \""<< symbol <<"\"\n";

\*start+=1;

}

else

{

cout << "Incorrect symbol \"" << stringpos[0] << "\", expected \""<<symbol<<"\"\n";

\*correct = false;

}

}

}

void findAndCheckNumber(char\* string,bool\* correct,int\* start,const char\* correctin,const char\* incorrectin)

{

if (\*correct) {

char\* stringpos = string+\*start;

char output[20] = "";

findNumber(stringpos,output);

if (checkNumber(output)) {

cout << "Detected count: " << output << "\n";

\*start+=strlen(output);

}

else

{

cout << "Incorrect number: " << output << "\n";

\*correct = false;

}

}

}

void var(char\* string,bool\* correct,int\* start)

{

skipSpace(string,start);

type(string,correct,start);

skipSpace(string,start);

ident(string,correct,start);

skipSpace(string,start);

char\* stringpos = string+\*start;

//cout << stringpos[0];

if (strncmp(stringpos,"[",1) == 0)

{

checksymbol(string,correct,start,"[","","");

skipSpace(string,start);

findAndCheckNumber(string,correct,start,"","");

skipSpace(string,start);

checksymbol(string,correct,start,"]","","");

}

skipSpace(string,start);

checksymbol(string,correct,start,";","","");

}

void vars(char\* string,bool\* correct,int\* start);

void structtype(char\* string,bool\* correct,int\* start)

{

skipSpace(string,start);

type(string,correct,start);

skipSpace(string,start);

ident(string,correct,start);

skipSpace(string,start);

checksymbol(string,correct,start,"{","","");

vars(string,correct,start);

checksymbol(string,correct,start,"}","","");

skipSpace(string,start);

ident(string,correct,start);

skipSpace(string,start);

char\* stringpos = string+\*start;

while (!(strncmp(stringpos,";",1) == 0) && !(strncmp(stringpos,"\0",1) == 0) && \*correct)

{

checksymbol(string,correct,start,",","","");

skipSpace(string,start);

ident(string,correct,start);

skipSpace(string,start);

stringpos = string+\*start;

}

checksymbol(string,correct,start,";","","");

}

void vars(char\* string,bool\* correct,int\* start)

{

skipSpace(string,start);

char\* stringpos = string+\*start;

while (!(strncmp(stringpos,"\0",1) == 0) && \*correct && !(strncmp(stringpos,"}",1) == 0))

{

char output[20] = "";

findWord(stringpos,output);

if (strncmp(output,"struct",6) == 0)

{

structtype(string,correct,start);

}

else

{

var(string,correct,start);

}

skipSpace(string,start);

stringpos = string+\*start;

}

}

int main() {

//char str[] = " unsigned char aer[8]; char record[80]; struct student { char name[22];char family[22];int old; } st1,ft";

char str[1000] = "";

cout << "Enter string:" << "\n";

cin.getline(str,sizeof(str));

char strwspaces[strlen(str)];

removeExtraSpaces(str,strwspaces);

bool correct = true;

int start = 0;

vars(strwspaces,&correct,&start);

if (correct) {cout << "Yes" << "\n";}

else {cout << "No" << "\n";}

return 0;

}

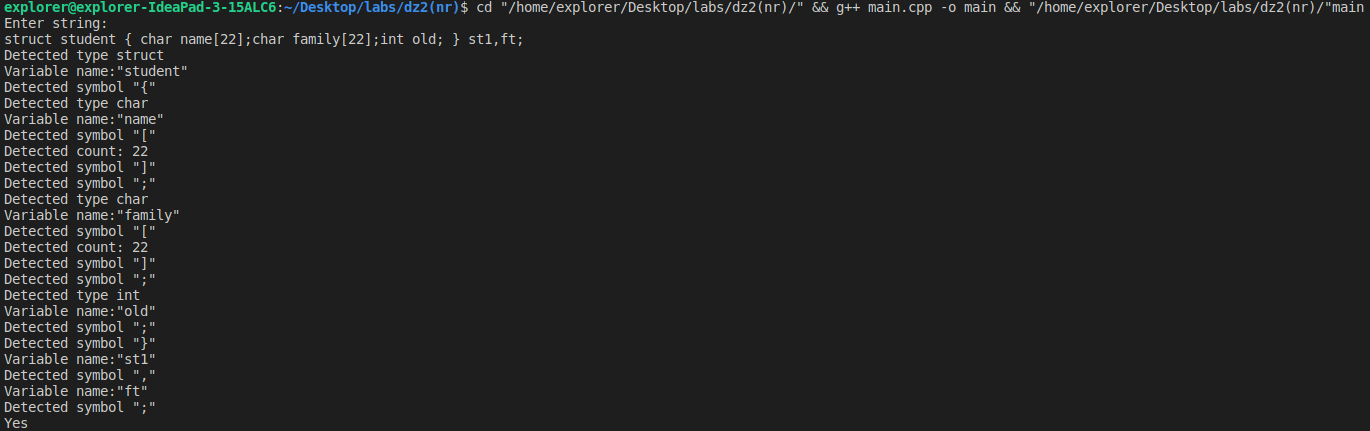
**Тесты.**

Результаты тестирования можно увидеть в таблице 1.

Таблица 1 – результаты тестирования

|  |  |  |
| --- | --- | --- |
| Исходные данные | Ожидаемый результат | Полученный результат |
| unsigned char aer[8]; char record[80]; struct student { char name[22];char family[22];int old; struct student2 { char name[22];char family[22];int old; } } st1,ft; | Yes | Yes |
| signed char aer[8]; | No | No |
| char name[2/]; | No | No |
| struct student { char name[22];char family[22];int old; struct student2 { char name[22];char family[22];int old; } stud; } sg,tg; | Yes | Yes |
| struct student2 { char name[22];char family[22];int old; ; | No | No |

Результат запуска программы на выполнение можно увидеть на рисунке 2.



*Рисунок 2 – Результат выполнения программы*

**Контрольные вопросы.**

**1. Дайте определение формального языка и формальной грамматики.**

Формальным языком называют множество цепочек символов, составленных из символов некоторого конечного алфавита.

Формальной грамматикой называют систему, которая определяет язык с помощью правил продукции и определяется как четверка: G = (VT, VN, P,S).

VT – алфавит языка

VN – вспомогательный алфавит, символы которого обозначают допустимые понятия языка

P – множество порождающих правил

S – аксиома грамматики.

**2. Как определяется тип грамматики по Хомскому?**

Классификация построена по правилам иерархии, т. е. грамматики третьего типа являются частным случаем грамматик второго типа и т. д. Тип грамматик определяется по ее самым сложным правилам.

**3. Поясните физический смысл и обозначения формы Бэкуса–Наура.**

Форма Бэкуса – Наура дает возможность более компактно написать

Грамматику, что значительно облегчает её понимание и чтение. Она связывает терминальные и нетерминальные символы и также применяет две операции: «можно заменить на» или «или».

**4. Что такое лексический анализ? Какие методы выполнения лексического анализа вы знаете?**

Лексический анализ – начальный этап процесса компиляции текста,

который написан на формальном языке программирования. При выполнении

лексического анализа текст разделяют на предложения - операторы языка.

После этого операторы – на слова. Их называют лексемами.

**5. Что такое синтаксический анализ? Какие методы синтаксического анализа вы знаете? К каким грамматикам применяются перечисленные вами методы?**

Синтаксический анализ – второй этап работы компилятора. Это

процесс распознавания в строке токенов конструкций языка. Затем определяется

способ построения анализатора в зависимости от типа грамматики. Для грамматик типа 3 используют конечные автоматы. Для грамматик типа 2 используют автоматы с магазинной памятью.

**6. Что является результатом лексического анализа?**

Результатом лексического анализа является таблица токенов.

**7. Что является результатом синтаксического анализа?**

Результатом синтаксического анализа являются распознанные конструкции языка.

**8. В чем заключается метод рекурсивного спуска?**

Метод рекурсивного спуска заключается в построение синтаксической

диаграммы и написание программы, начинающую вызов с функции,

реализующей аксиому языка.

**9. Что такое таблица предшествования и для чего она строится?**

Таблица предшествования отображает отношения предшествования между терминальными символами строки.

Таблица предшествования строится для определения основы, которая должна быть свернута на следующем этапе грамматического разбора.

**10. Как с использованием таблицы предшествования осуществляют синтаксический анализ?**

Использование таблиц предшествования помогает однозначно определить начало основы, конец основы и принадлежность одной основе. Также для определения, сформированы ли соответствующие комбинации. Далее выполняется операция свертки, когда соответствующая часть основы получена полностью. В конце получаем разбор предложения на отдельные операции.

**Вывод.** Былиизучены теоретические основы и основные методы приемов разработки лексических и синтаксических анализаторов регулярных и контекстно-свободных формальных языков.