МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

Кафедра компьютерных технологий и программной инженерии

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Ст.преподаватель		М.Д.Поляк
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

«НИСХОДЯЩАЯ ТРАНСЛЯЦИЯ НА ОСНОВЕ ВЫЧИСЛЕНИЯ НАСЛЕДУЕМЫХ И СИНТЕЗИРУЕМЫХ АТРИБУТОВ»

по курсу: МЕТОДЫ ТРАНСЛЯЦИИ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4831		К.А. Корнющенков
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2020

1. Задание на лабораторную работу:

Выполните программную реализацию решения задачи, выбранной по номеру варианта, в виде синтаксически управляемой нисходящей трансляции с использованием наследуемых и синтезируемых атрибутов.

2. Задание по варианту (вариант №4):

<S>::=<E>
<E>::=<E>+<T>
<S>::=<T>
<T>::=<T>*<F>
<T>::=<F>
<F>::=x
<F>::=6
<F>::=5
<F>::=(E)

3. Ход работы:

Таблица разбора:

	x	5	6	+	*	(	)	#
S	1	1	1	0	0	1	0	0
E	2	2	2	0	0	2	0	0
Z	0	0	0	3	0	0	10	10
T	4	4	4	0	0	4	0	0
H	0	0	0	11	5	0	11	11
F	6	7	8	0	0	9	0	0
x	99	0	0	0	0	0	0	0
5	0	99	0	0	0	0	0	0
6	0	0	99	0	0	0	0	0
+	0	0	0	99	0	0	0	0
*	0	0	0	0	99	0	0	0
(	0	0	0	0	0	99	0	0
)	0	0	0	0	0	0	99	0
#	0	0	0	0	0	0	0	100

Где цифры 1-11 – правила грамматики

99 – выброс

100 – допуск

0 – ошибка

Листинг программы

```
#include <iostream>

#include <string>

#include <vector>

using namespace std;

vector<int> symbolVector; //стек символов

vector<char> operationVector; //стек операций

//MARK: Check

bool checkData(string str){

    string gramm[] = { "", "E","TZ","+TZ","FH","*FH","x","5","6","(E)","","" }; // грамматика

    string tableLeft = "SEZTHFx56+*()#"; // строки таблицы (терминалы + нетерминалы + символ конца строки)

    string tableTop = "x56+*()#"; // столбцы таблицы (терминалы + символ конца строки)

    int table[14][8] = {

        //x  5  6  +  *  (  )  #

        { 1, 1, 1, 0, 0, 1, 0, 0 }, //S

        { 2, 2, 2, 0, 0, 2, 0, 0 }, //E

        { 0, 0, 0, 3, 0, 0, 10, 10 }, //Z
```

```
    { 4, 4, 4, 0, 0, 4, 0, 0 }, // T
    { 0, 0, 0, 11, 5, 0, 11, 11}, // H
    { 6, 7, 8, 0, 0, 9, 0, 0 }, // F
    { 99, 0, 0, 0, 0, 0, 0, 0 }, // x
    { 0, 99, 0, 0, 0, 0, 0, 0 }, // 5
    { 0, 0, 99, 0, 0, 0, 0, 0 }, // 6
    { 0, 0, 0, 99, 0, 0, 0, 0 }, // +
    { 0, 0, 0, 0, 99, 0, 0, 0 }, // *
    { 0, 0, 0, 0, 0, 99, 0, 0 }, // (
    { 0, 0, 0, 0, 0, 0, 99, 0 }, // )
    { 0, 0, 0, 0, 0, 0, 0, 100 }  // $
};
string stack = "#S";
str += '#'; //добавление аксиомы и символа конца строки
for (int i = 1; !str.empty(); i++){
    char symb = str.front(); // обращение к 1 элементу строки
    char stack_state = stack.back(); // обращение к последнему элементу строки

    unsigned long int indeTopTable = tableTop.find(symb);
    unsigned long int indexLeftTable = tableLeft.find(stack_state);
    if (indeTopTable == string::npos) return false; //проверка на несуществующую позицию
```

```
int command = table[indexLeftTable][indeTopTable];
switch (command){
    case 0:
        return false; // ошибка
    case 100:
        return true; // верное
    case 99:{
        if (symb == '5' || symb == '6')
            symbolVector.push_back(symb - '0');
        if (symb == 'x') //вместо x добавляем x
            symbolVector.push_back(-1);
        if (symb == '*' || symb == '+' || symb == '(' || symb == ')')
            operationVector.push_back(symb);
        str.erase(0, 1); // удаляем 0 элемент строки
        stack.pop_back();
        continue;
    }
}
string state = gramm[command];
stack.pop_back();
```

```
        stack += string(state.rbegin(), state.rend());
    }
    return false;
}

//MARK: Преобразования подданных
pair<vector<int>,vector<char>> conversionData(vector<int> symbol,vector<char> operation){
    unsigned long int size = operation.size();
    for (auto index=0;index<size;index++){
        if (symbol[index] != -1){ //!= x
            if (operation[index] == '+'){
                symbol.erase(symbol.begin() + index);
                operation.erase(operation.begin() + index);
                size = operation.size();
                index = -1;
            }else if (operation[index] == '*' && symbol[index+1] != -1){
                symbol[index] = symbol[index] * symbol[index + 1];
                symbol.erase(symbol.begin() + index + 1);
                operation.erase(operation.begin() + index);
                index = -1;
                size = operation.size();
```

```
    }
    if (index == size - 1){
        break;
    }
}else{ //==x
    if (symbol[index] == -1 && operation[index] == '^'){
        index++;
        continue;
    }else{
        int countX = 1;
        for(auto j=index;j<size;j++){
            if (symbol[j] == -1 && operation[j] == '*' && symbol[j+1] == -1){
                countX++;
                if (j == size - 1){
                    int count2 = countX - 1;
                    symbol.erase(symbol.begin() + index + count2);
                    while (count2>0) {
                        symbol.erase(symbol.begin() + index + count2 - 1);
                        operation.erase(operation.begin() + index + count2 - 1);
                        count2--;
                    }
```

```
            symbol.insert(symbol.begin() + index, countX);
            operation.insert(operation.begin() + index, '*');
            symbol.insert(symbol.begin() + index + 1, -1);
            operation.insert(operation.begin() + index + 1, '^');
            symbol.insert(symbol.begin() + index + 2, countX - 1);
            size = operation.size();
            break;
        }
    }else{
        int countSup = countX - 1;
        symbol.erase(symbol.begin() + index + countSup);
        while (countSup!=0) {
            symbol.erase(symbol.begin() + index + countSup - 1);
            operation.erase(operation.begin() + index + countSup - 1);
            countSup--;
        }
        symbol.insert(symbol.begin() + index, countX);
        operation.insert(operation.begin() + index, '*');
        symbol.insert(symbol.begin() + index + 1, -1);
        operation.insert(operation.begin() + index + 1, '^');
        symbol.insert(symbol.begin() + index + 2, countX - 1);
```

```
                size = operation.size();
                index = -1;
                break;
            }
        }
    }
}
return pair<vector<int>,vector<char>>(symbol,operation);
}

//MARK: Преобразовние в стоку
string conversionDataToString(vector<char> operationVector, vector<int> stackDigit){
    string returnDataLine = "";
    string xLine = "x";
    for (int i=0;i<operationVector.size();i++){
        if (symbolVector[i] == -1)
            returnDataLine += xLine;
        else
            returnDataLine += to_string(symbolVector[i]);
        returnDataLine += operationVector[i]; //в теории тут можно проверку добавить на digite
i+1 != числу
```

```
    }

    if (symbolVector[symbolVector.size()-1] == -1)
        returnDataLine += "1";
    else
        returnDataLine += to_string(symbolVector[symbolVector.size()-1]);

    return returnDataLine;
}

//MARK: Разделение даты на подчасти
string separationData(){
    if (symbolVector[symbolVector.size()-1] == 5 || symbolVector[symbolVector.size()-1] == 6){
        symbolVector.pop_back();
        operationVector.pop_back();
    }

    unsigned long int s = operationVector.size();
    bool check = false;
    //стеки для обработки строк между ()
    vector<int> stackDigit; //стек символов
```

```
vector<char> stackOperation; //стек операций
int index = 0;
//избавляемся от скобок
for (int i=0;i<s;i++){
    if (operationVector[i] == '('){
        index = i;
        check = true;
        stackDigit.clear();
        stackOperation.clear();
        stackDigit.push_back(symbolVector[i]);
    }else if (operationVector[i] == ')'){
        int data = 1;
        bool checkData = false;
        if (index != 0 && operationVector[index-1] == '*'){
            data = symbolVector[index - 1];
            index--;
            checkData = true;
        }
        //получаем пару значение
        auto p = conversionData(stackDigit,stackOperation);
        //удаляем все значения из скобок в исходном векторе
```

```
    symbolVector.erase(symbolVector.begin()+index,symbolVector.begin()+i);
    operationVector.erase(operationVector.begin()+index,operationVector.begin()+i+1);
    //добавляем преобразованные данные
    for (int i=0; i<p.first.size(); i++){
        if (checkData){
            symbolVector.insert(symbolVector.begin()+index+i, p.first[i]*data);
            checkData = false;
        }else{
            symbolVector.insert(symbolVector.begin()+index+i, p.first[i]);
            if (p.second[i] == '+')
                checkData = true;
        }
    }
    for (int i=0; i<p.second.size(); i++){
        operationVector.insert(operationVector.begin()+index+i, p.second[i]);
    }
    i = -1;
    s = operationVector.size();
    check = false;
}else if (check){
    stackDigit.push_back(symbolVector[i]);
```

```
            stackOperation.push_back(operationVector[i]);
        }
    }

    //обработка финальной строки
    auto p = conversionData(symbolVector,operationVector);
    symbolVector = p.first;
    operationVector = p.second;

    //2 раз для ситуации типо 6*5*x*x, иначе он конвертирует в 30*2*x^1, а так в 60*x^1
    p = conversionData(symbolVector,operationVector);
    symbolVector = p.first;
    operationVector = p.second;

    return conversionDataToString(operationVector, symbolVector);
}

//MARK:Main
int main(){
    while (true){
        string str;
```

```
        cout << "Введите выражение: \t";
        cin >> str;

        switch (checkData(str)){
            case true:{
                cout << "Выражение соответствует грамматике.\n";
                cout << separationData() << endl;
                break;
            }
            case false:{
                cout << "Некорректный ввод!\n";
                break;
            }
        }

        symbolVector.clear();
        operationVector.clear();
        cout << "\n";
    }
}
```

4. Пример работы программы

```
Введите выражение:  5*6*x*x+5*(x*x*x+5*x*x*x*x)
Выражение соответствует грамматике.
60*x^1+15*x^2+100*x^3
```

5. Вывод

В ходе выполнения лабораторной работы была написана программа для реализация задачи в виде нисходящей трансляции на основе вычисления наследуемых и синтезируемых атрибутов.