S -> <Программа>

<Программа> -> <Опер> <Программа>

<Программа> -> <Опер>

<Опер> -> ;

<Опер> -> <Опер> <Об>

<Об> -> <Перем> : < Тип > ;

<Перем> -> <Перем> , V

<Перем> -> <Перем> V

<Тип> -> int

<Тип> -> vector

<Опер> -> set <Перем> = <E> ;

<Опер> -> while <Тест> do <Программа> end ;

<Тест> -> <E> R <E>

<Опер> -> for each V in <E> do <Программа> end ;

<Опер> -> if <Тест> then <Программа> <Иначе> end ;

<Иначе> -> else <Программа>

<Иначе> -> ɛ

<Опер> -> cin >> <Ввод> ;

<Ввод> -> V >> <Ввод>| V

<Ввод> -> V

<Опер> -> cout <Вывод> ;

<Вывод> -> << <E>

<Вывод> -> << <E> <Вывод>

<Опер> -> L:

<Опер> -> jump L ;

<Опер> -> switch ( <E> ) <Перечисление> <Другой\_Случай> end ;

<Перечисление> -> <Перечисление> <Переч\_when>

<Перечисление> -> <Переч\_when>

<Переч\_when> -> when <Пер\_Конст> : <Программа>

<Пер\_Конст> -> C

<Пер\_Конст> -> C , <Пер\_Конст>

<Другой\_Случай> -> otherwise <Программа>

<Опер> -> throw;

<Опер> -> rem

<E> -> <E> + <T>

<E> -> <T>

<E> -> <E> - <T>

<T> -> <T> \* <F>

<T> -> <T> / <F>

<T> -> <T> % <F>

<T> -> <F>

<T> -> <T> & <F>

<F> -> V

<F> -> C

<F> -> ( <E> )

<F> -> concat ( <E> , <E> )

<F> -> shiftl ( <E> , <E> )

<F> -> shiftr ( <E> , <E> )

<F> -> sub\_vec ( <E> , <E> , <E> )

<F> -> add ( <E> , <E> )

<F> -> delete ( <E> , <E> )

<S> -> <Prog>

<Prog> -> <Oper> <Prog>

<Prog> -> <Oper>

<Oper> -> ;

<Oper> -> <Oper> <Ob>

<Oper> -> set <Perem> = <E> ;

<Oper> -> while <Test> do <Prog> end ;

<Oper> -> for each V in <E> do <Prog> end ;

<Oper> -> if <Test> then <Prog> <Else> end ;

<Oper> -> L:

<Oper> -> jump L ;

<Oper> -> switch ( <E> ) <Translation> <Any\_hap> end ;

<Oper> -> thro w ;

<Oper> -> rem

<Ob> -> <Perem> : < Type > ;

<Perem> -> <Perem> , V

<Perem> -> <Perem> V

<Type> -> int

<Type> -> vector

<Test> -> <E> <R> <E>

<Else> -> else <Prog>

<Else> -> e

<Oper> -> cin >> <Cin> ;

<Cin> -> V >> <Cin>

<Cin> -> V

<Oper> -> cout <Cout> ;

<Cout> -> << <E>

<Cout> -> << <E> <Cout>

<Translation> -> <Translation> <Transfer\_when>

<Translation> -> <Transfer\_when>

<Transfer\_when> -> when <Transfer\_Const> : <Prog>

<Transfer\_Const> -> C

<Transfer\_Const> -> C , <Transfer\_Const>

<Any\_hap> -> otherwise <Prog>

<E> -> <T> <E'>

<E> -> <T>

<E'> -> + <T> <E'>

<E'> -> - <T> <E'>

<E'> -> + <T>

<E'> -> - <T>

<T> -> <F> <T'>

<T> -> <F>

<T'> -> \* <F> <T'>

<T'> -> / <F> <T'>

<T'> -> \* <F>

<T'> -> / <F>

<T'> -> % <F> <T'>

<T'> -> & <F> <T'>

<T'> -> % <F>

<T'> -> & <F>

<F> -> <VV>

<F> -> <Count>

<F> -> ( <E> )

<F> -> concat ( <E> <Comma> <E> )

<F> -> shiftl ( <E> <Comma> <E> )

<F> -> shiftr ( <E> <Comma> <E> )

<F> -> sub\_vec ( <E> <Comma> <E> <Comma> <E> )

<F> -> add ( <E> <Comma> <E> )

<F> -> delete ( <E> <Comma> <E> )

<S> -> <Prog>

<Prog> -> <Oper>

<Prog> -> <Oper> <Prog>

<Oper> -> <SimpleOper>

<Oper> -> <SimpleOper> <ObSequence>

<ObSequence> -> <Ob>

<ObSequence> -> <Ob> <ObSequence>

<SimpleOper> -> ;

<SimpleOper> -> set <Perem> = <E> ;

<SimpleOper> -> while <Test> do <Prog> end ;

<SimpleOper> -> for each V in <E> do <Prog> end ;

<SimpleOper> -> if <Test> then <Prog> end ;

<SimpleOper> -> if <Test> then <Prog> else <Prog> end ;

<SimpleOper> -> L : ;

<SimpleOper> -> jump L ;

<SimpleOper> -> switch ( <E> ) <CaseBlock> end ;

<SimpleOper> -> throw ;

<SimpleOper> -> rem ;

<SimpleOper> -> cin >> <InputList> ;

<SimpleOper> -> cout <OutputList> ;

<CaseBlock> -> <CaseList>

<CaseBlock> -> <CaseList> otherwise <Prog>

<CaseList> -> <Case>

<CaseList> -> <Case> <CaseList>

<Case> -> when <ConstList> : <Prog>

<ConstList> -> C

<ConstList> -> C , <ConstList>

<InputList> -> V

<InputList> -> V >> <InputList>

<OutputList> -> << <E>

<OutputList> -> << <E> <OutputList>

<Ob> -> <Perem> : <Type> ;

<Perem> -> V

<Perem> -> V , <Perem>

<Type> -> int

<Type> -> vector

<Test> -> <E> <RelOp> <E>

<RelOp> -> <=

<RelOp> -> <

<RelOp> -> >

<RelOp> -> ==

<RelOp> -> !=

<RelOp> -> >=

<E> -> <T>

<E> -> <T> + <E>

<E> -> <T> - <E>

<T> -> <F>

<T> -> <F> \* <T>

<T> -> <F> / <T>

<T> -> <F> % <T>

<T> -> <F> & <T>

<F> -> V

<F> -> C

<F> -> ( <E> )

<F> -> concat ( <ArgList2> )

<F> -> shiftl ( <ArgList2> )

<F> -> shiftr ( <ArgList2> )

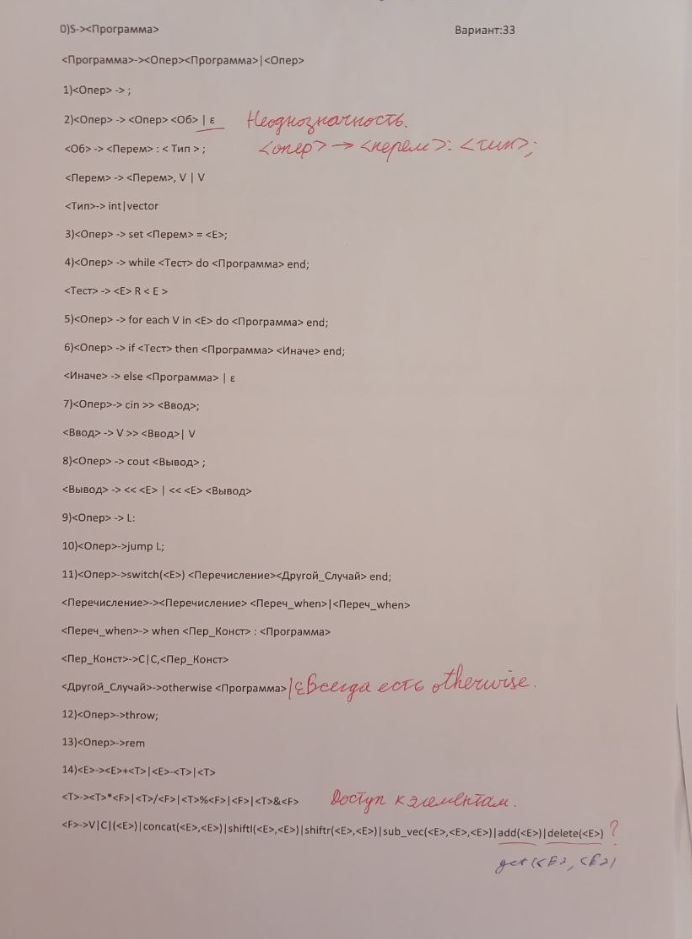
<F> -> sub\_vec ( <ArgList3> )

<F> -> add ( <ArgList2> )

<F> -> delete ( <ArgList2> )

<ArgList2> -> <E> , <E>

<ArgList3> -> <E> , <E> , <E>



<S> -> <объявления> <программа>

<S> -> <программа>

<программа> -> <оператор> <программа>

<программа>-> <оператор>

<оператор>->declare V

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | S | T | R | a | b | c | $ |
| S |  |  |  | = | = |  |  |
| T | = | < | < | < | < | < |  |
| R | = | < | < | < | < | < |  |
| a |  |  |  | > | > | > | > |
| b |  |  |  | > | > | > | > |
| c |  |  |  | > | > |  | > |
| $ |  | < | < | < | < | < |  |

<S> -> <T> <S> a

<S> -> <R> <S> b

<R> -> b

<T> -> a

<S> -> c

<E> -> <T><E’>

<E> -> <T>

<E’> -> +<T><E’>

<T> -> <F><T’>

<T> -> <F>

<T’> -> \*<F><T’>

<T’> -> \*<F>

<F> -> (<E>)

<F> -> u

void printPrecedenceTable() {

set<string> allSymbols;

allSymbols.insert(grammar.getNonTerminals().begin(), grammar.getNonTerminals().end());

allSymbols.insert(grammar.getTerminals().begin(), grammar.getTerminals().end());

allSymbols.insert("$");

const int colWidth = 12;

*// Открываем файл для записи*

ofstream outFile("precedence\_table.txt");

if (!outFile.is\_open()) {

cerr << "Ошибка при открытии файла для записи таблицы предшествования" << endl;

return;

}

outFile << "\nТаблица предшествования:\n";

outFile << string(colWidth, ' ');

for (const string& colSymbol : allSymbols) {

outFile << setw(colWidth) << colSymbol;

}

outFile << endl;

for (const string& rowSymbol : allSymbols) {

outFile << setw(colWidth) << rowSymbol;

for (const string& colSymbol : allSymbols) {

pair<string, string> key = { rowSymbol, colSymbol };

char relation = ' ';

if (precedenceTable.count(key)) {

relation = precedenceTable[key];

}

outFile << setw(colWidth) << relation;

}

outFile << endl;

}

outFile.close();

cout << "Таблица предшествования записана в файл precedence\_table.txt" << endl;

}

void printPrecedenceTable() {

vector<string> allSymbols;

vector<string> allSymbols2 = grammar.GET\_TR();

allSymbols = (grammar.GET\_NON());

for (int i = 0; i < allSymbols2.size(); i++)

{

allSymbols.push\_back(allSymbols2[i]);

}

allSymbols.push\_back("$");

const int colWidth = 12;

cout << "\nТаблица:\n";

cout << string(colWidth, ' ');

for (const string& colSymbol : allSymbols) {

cout << setw(colWidth) << colSymbol;

}

cout << endl;

for (const string& rowSymbol : allSymbols) {

cout << setw(colWidth) << rowSymbol;

for (const string& colSymbol : allSymbols) {

pair<string, string> key = { rowSymbol, colSymbol };

char relation = ' ';

if (precedenceTable.count(key)) {

relation = precedenceTable[key];

}

cout << setw(colWidth) << relation;

}

cout << endl;

}

}