Грамматика выражений

```
E → E7
E7 → E7 || E6 | E6
E6 → E6 && E5 | E5
E5 → E4 == E4 | E4 != E4 | E4 > E4 | E4 < E4 | E4 <= E4 | E4
E4 → E4 + E3 | E4 – E3 | E3
E3 → E3 * E2 | E2
E2 → !E1 | E1
E1 → ++ id | id ++ | ( E ) | num | id | id(ArgList)
```

```
E → E7
E7 → E6 E7’
E7’ → || E6 E7’ | ε
E6 → E5 E6’
E6’ → && E5 E6’ | ε
E5 → E4 E5’
E5’ → == E4 | != E4 | > E4 | < E4 | <= E4 | ε
E4 → E3 E4’
E4’ → + E3 E4’ | - E3 E4’ | ε
E3 → E2 E3’
E3’ → * E2 E3’ | ε
E2 → ! E1 | E1
E1 → ++ id | ( E ) | num | id E1’
E1’ → ++ | ( ArgList ) | ε
ArgList → E ArgList’ | ε
ArgList’ → , E ArgList’ | ε
```

Грамматика остальных конструкций языка MiniC

```
DeclareStmt → Type id DeclareStmt’
DeclareStmt’ → ( ParamList ) { StmtList }
DeclareStmt’ → = num DeclVarList’ ; | DeclVarList’ ;
Type → char | int
DeclVarList’ → , id InitVar DeclVarList’ | ε
InitVar → = num | = chr | ε
ParamList → Type id ParamList’ | ε
ParamList’ → , Type id ParamList’ | ε
StmtList → Stmt StmtList | ε
Stmt → DeclareStmt | AssignOrCallOp | WhileOp | ForOp | IfOp | SwitchOp | IOp | OOp | ;
| { StmtList } | return E ;
AssignOrCallOp → AssignOrCall ;
AssignOrCall → id AssignOrCall’
AssignOrCall’ = E | ( ArgList )
WhileOp → while (E) Stmt
ForOp → for (ForInit; ForExp; ForLoop) Stmt
ForInit → AssignOrCall | ε
ForExp → E | ε
ForLoop → AssignOrCall | ++ id | ε
IfOp → if ( E ) Stmt ElsePart
ElsePart → else Stmt | ε
SwitchOp → switch ( E ) { Cases }
Cases → ACase Cases’
Cases’ → ACase Cases’ | ε
ACase → case num : StmtList | default : StmtList
IOp → in id ;
OOp → out OOp’ ;
OOp’ → E | str
```

Транслирующие грамматики

Каждый нетерминал грамматики помимо прочих содержит наследуемый атрибут контекста (**C или C'**), который копируется из родительского во все дочерние. Так как данный атрибут есть у всех нетерминальных символов, то для краткости записи в описании грамматики он опущен кроме тех случаев, когда он явно требуется. Атрибут содержит ID функции, внутри которой производится синтаксический разбор, или -1 – для глобального контекста.

В грамматике все синтезируемые атрибуты выделены красным цветом.

Транслирующая грамматика для выражений языка MiniC

№	Правило	Семантическое определение
1	$E_p \rightarrow E7_q$	p = q
2	$E7_p \rightarrow E6_q\ E7'_{rs}$	r = q; p = s
3	$E7'_{pq} \rightarrow \|\|\ E6_r\ \{OR\}_{prs}\ E7'_{st}$	s = alloc(C); q = t
4	$E7'_{pq} \rightarrow \varepsilon$	q = p
5	$E6_p \rightarrow E5_q\ E6'_{rs}$	r = q; p = s
6	$E6'_{pq} \rightarrow \&\&\ E5_r\ \{AND\}_{prs}\ E6'_{st}$	s = alloc(C); q = t
7	$E6'_{pq} \rightarrow \varepsilon$	q = p
8	$E5_p \rightarrow E4_q\ E5'_{rs}$	r = q; p = s
9	$E5'_{pq} \rightarrow ==\ E4_r\ \{MOV\}_{1,,s}\ \{EQ\}_{prl}\ \{MOV\}_{0,,s}\ \{LBL\}_{,,l}$	s = alloc(C); l = newLabel(); q = s
10	$E5'_{pq} \rightarrow !=\ E4_r\ \{MOV\}_{1,,s}\ \{NE\}_{prl}\ \{MOV\}_{0,,s}\ \{LBL\}_{,,l}$	s = alloc(C); l = newLabel(); q = s
11	$E5'_{pq} \rightarrow >\ E4_r\ \{MOV\}_{1,,s}\ \{GT\}_{prl}\ \{MOV\}_{0,,s}\ \{LBL\}_{,,l}$	s = alloc(C); l = newLabel(); q = s
12	$E5'_{pq} \rightarrow <\ E4_r\ \{MOV\}_{1,,s}\ \{LT\}_{prl}\ \{MOV\}_{0,,s}\ \{LBL\}_{,,l}$	s = alloc(C); l = newLabel(); q = s
13	$E5'_{pq} \rightarrow <=\ E4_r\ \{MOV\}_{1,,s}\ \{LE\}_{prl}\ \{MOV\}_{0,,s}\ \{LBL\}_{,,l}$	s = alloc(C); l = newLabel(); q = s
14	$E5'_{pq} \rightarrow \varepsilon$	q = p
15	$E4_p \rightarrow E3_q\ E4'_{rs}$	r = q; p = s
16	$E4'_{pq} \rightarrow +\ E3_r\ \{ADD\}_{prs}\ E4'_{st}$	s = alloc(C); q = t
17	$E4'_{pq} \rightarrow -\ E3_r\ \{SUB\}_{prs}\ E4'_{st}$	s = alloc(C); q = t
18	$E4'_{pq} \rightarrow \varepsilon$	q = p
19	$E3_p \rightarrow E2_q\ E3'_{rs}$	r = q; p = s
20	$E3'_{pq} \rightarrow *\ E2_r\ \{MUL\}_{prs}\ E3'_{st}$	s = alloc(C); q = t
21	$E3'_{pq} \rightarrow \varepsilon$	q = p
22	$E2_p \rightarrow !\ E1_q\ \{NOT\}_{q,,r}$	r = alloc(C); p = r
23	$E2_p \rightarrow E1_q$	p = q
24	$E1_p \rightarrow (\ E_q\ )$	p = q
25	$E1_p \rightarrow num_{val}$	p = val
26	$E1_p \rightarrow chr_{val}$	p = val
27	$E1_p \rightarrow ++\ id_{name}\ \{ADD\}_{q1q}$	q = checkVar(C, name); p = q
28	$E1_p \rightarrow id_{name}\ E1'_{rs}$	r = name; p = s
29	$E1'_{pq} \rightarrow ++\ \{MOV\}_{s,,r}\ \{ADD\}_{s1s}$	s = checkVar(C,p); r=alloc(C); q = r
30	$E1'_{pq} \rightarrow (\ ArgList_n\ )\ \{CALL\}_{s,r}$	s = checkFunc(p,n); r=alloc(C); q = r
31	$E1'_{pq} \rightarrow \varepsilon$	q = checkVar(C, p)
32	$ArgList_n \rightarrow E_p\ \{PARAM\}_{,,p}\ ArgList'_m$	n = m + 1
33	$ArgList_n \rightarrow \varepsilon$	n = 0
34	$ArgList'_n \rightarrow ,\ E_p\ \{PARAM\}_{,,p}\ ArgList'_m$	n = m + 1
35	$ArgList'_n \rightarrow \varepsilon$	n = 0

Транслирующая грамматика для основных конструкций языка MiniC

№	Правило	Семантическое определение
1	DeclareStmt → $Type_p$ id_{name} $DeclareStmt'_{qr}$	q = p; r = name
2	$DeclareStmt'_{pqC}$ → ($ParamList_{nC'}$) { $StmtList_{C'}$ } **{RET}**$_{,,'0'}$	**if** C > -1: error (function definition inside function) **else:** C' = addFunc(q,p) получив n из ParamList, записываем его в ТС для строки C'
3	$DeclareStmt'_{pq}$ → = num_{val} $DeclVarList'_r$;	addVar(q,C,p,val); r = p
4	$DeclareStmt'_{pq}$ → $DeclVarList'_r$;	addVar(q,C,p); r = p
5	$Type_p$ → char	p = char
6	$Type_p$ → int	p = int
7	$DeclVarList'_p$ → , id_{name} $InitVar_{rs}$ $DeclVarList'_t$	r = p; s = name; t = p
8	$DeclVarList'_p$ → **ε**	
9	$InitVar_{pq}$ → = num_{val} ИЛИ = chr_{val}	addVar(q,C,p,val)
10	$InitVar_{pq}$ → **ε**	addVar(q,C,p)
11	$ParamList_n$ → $Type_q$ id_{name} $ParamList'_s$	addVar(name,C,q); n = s + 1
12	$ParamList_n$ → **ε**	n = 0
13	$ParamList'_n$ → , $Type_q$ id_{name} $ParamList'_s$	addVar(name,C,q); n = s + 1
14	$ParamList'_n$ → **ε**	n = 0
15	StmtList → Stmt StmtList	стартовый нетерминал
16	StmtList → **ε**	
17	Stmt → DeclareStmt	
18	Stmt → AssignOrCallOp	**if** C == -1: error (operator should be inside function)
19	Stmt → WhileOp	
20	Stmt → ForOp	
21	Stmt → IfOp	
22	Stmt → SwitchOp	
23	Stmt → IOp	
24	Stmt → OOp	
25	Stmt → { StmtList }	
26	Stmt → return E_p **{RET}**$_{,,p}$;	
27	Stmt → ;	
28	AssignOrCallOp → AssignOrCall ;	
29	AssignOrCall → id_{name} $AssignOrCall'_q$	q = name
30	$AssignOrCall'_p$ → = E_q **{MOV}**$_{q,,r}$	r = checkVar(C,p)
31	$AssignOrCall'_p$ → ($ArgList_n$) **{CALL}**$_{q,r}$	q = checkFunc(p,n); r = alloc(C)
32	WhileOp → while **{LBL}**$_{,,l1}$ (E_p) **{EQ}**$_{p,0,l2}$ Stmt **{JMP}**$_{,,l1}$ **{LBL}**$_{,,l2}$	l1 = newLabel(); l2 = newLabel()
33	ForOp → for (ForInit ; **{LBL}**$_{,,l1}$ $ForExp_p$; **{EQ}**$_{p,0,l4}$ **{JMP}**$_{,,l3}$ **{LBL}**$_{,,l2}$ ForLoop **{JMP}**$_{,,l1}$) **{LBL}**$_{,,l3}$ Stmt **{JMP}**$_{,,l2}$ **{LBL}**$_{,,l4}$	l1 = newLabel(); l2 = newLabel() l3 = newLabel(); l4 = newLabel()
34	ForInit → AssignOrCall	
35	ForInit → **ε**	
36	$ForExp_p$ → E_q	p = q
37	$ForExp_p$ → **ε**	p = '1'

38	ForLoop → AssignOrCall	
39	ForLoop → ++ id_{name} **{ADD}**$_{p1p}$	p = checkVar(C,name)
40	ForLoop → ε	
41	IfOp → if (E_p) **{EQ}**$_{p,0,l1}$ Stmt **{JMP}**$_{,,l2}$ **{LBL}**$_{,,l1}$ ElsePart **{LBL}**$_{,,l2}$	l1 = newLabel(); l2 = newLabel()
42	ElsePart → else Stmt	
43	ElsePart → ε	
44	SwitchOp → switch (E_p) { $Cases_{q,end}$ } **{LBL}**$_{,,end}$	q = p; end = newLabel()
45	$Cases_{p,end}$ → $ACase_{q,end1,def1}$ $Cases'_{r,end2,def2}$	q = r = p; end1 = end2 = end; def2 = def1

46	$Cases'_{p,end,def}$ → $ACase_{q,end1,def1}$ $Cases'_{r,end2,def2}$	q = r = p; end1 = end2 = end;
		if(def >= 0 && def1 >= 0){
		  SYNTAX ERROR: two default sect.
		} else def2 = max(def, def1)

47	$Cases'_{p,end,def}$ → ε **{JMP}**$_{,,q}$	if (def >= 0) q = def; else q = end
48	$ACase_{p,end,def}$ → case num_{val} **{NE}**$_{p,val,next}$: StmtList **{JMP}**$_{,,end}$ **{LBL}**$_{,,next}$	next = newLabel(); def = -1
49	$ACase_{p,end,def}$ → default : **{JMP}**$_{,,next}$ **{LBL}**$_{,,def}$ StmtList **{JMP}**$_{,,end}$ **{LBL}**$_{,,next}$	next = newLabel(); def = newLabel()
50	IOp → in id_{name} ; **{IN}**$_{,,p}$	p = checkVar(C,name)
51	OOp → out OOp' ;	
52	OOp' → E_p **{OUT}**$_{,,p}$	
53	OOp' → str_s **{OUT}**$_{,,s}$	

Назначение функций

- метка newLabel()
  Возвращает номер следующей по порядку метки.

- код alloc(scope)
  Создает в таблице символов новую строку для переменной без имени (или с именем, которое не может быть присвоено нормальной переменной, чтобы функция checkId ее случайно не нашла) в контексте scope и возвращает ее код.

- код addVar(name, scope, type, init = 0)
  Вызывается для объявления переменной или параметра функции. Параметры:

  - name – имя идентификатора
  - scope – контекст, в котором происходит объявление: -1 для глобального или код функции для локального
  - type – тип переменной
  - init – значение по умолчанию

  Проверяет, что в таблице символов нет идентификатора с совпадающими scope и именем name. Если это так, то создается новая запись, для которой прописываются переданные type, init, scope и kind (равный var) и возвращается код созданной записи. Иначе возвращается значение, которое не может быть кодом никакой записи (идентификатор уже занят в данном контексте).

- код addFunc(name, type)
  Вызывается для объявления функции. Работает только в глобальном контексте. Если идентификатор name уже объявлен (как функция или переменная), то возвращает false. Работает аналогично addVar, но устанавливает kind = func. Возвращает код созданной записи или ошибку (значение, которое не может быть кодом никакой записи).

- код checkVar(scope, name)
  Вызывается для проверки, является ли данный идентификатор объявленной **переменной** в текущем локальном или глобальном контекстах. Параметры:

- scope – код контекста
- name – имя идентификатора

Функция работает по следующему алгоритму:

1. ищет в контексте scope идентификатор с именем name; если не находит и scope > -1, то повторяет поиск для scope = -1;
2. если найденный идентификатор не объявлен (т.е. его не удалось найти), то возвращает ошибку;
3. если объявлен, но это не переменная, то возвращает ошибку;
4. иначе (это переменная и она объявлена) возвращает код найденной записи.

- код checkFunc(name, len)
  Вызывается для проверки, является ли данный идентификатор объявленной **функцией**. Работает аналогично методу checkVar(), за исключением двух особенностей:
  - поиск производится только в глобальном контексте;
  - проверяется совпадение количества переданных аргументов len с количеством параметров, заданном при объявлении функции. Если количество аргументов не совпадает, возвращается ошибка.