

Грамматика выражений

$E \rightarrow E7$
 $E7 \rightarrow E7 \mid \mid E6 \mid E6$
 $E6 \rightarrow E6 \ \&\& \ E5 \mid E5$
 $E5 \rightarrow E4 \ == \ E4 \mid E4 \ != \ E4 \mid E4 \ > \ E4 \mid E4 \ < \ E4 \mid E4 \ <= \ E4 \mid E4$
 $E4 \rightarrow E4 \ + \ E3 \mid E4 \ - \ E3 \mid E3$
 $E3 \rightarrow E3 \ * \ E2 \mid E2$
 $E2 \rightarrow !E1 \mid E1$
 $E1 \rightarrow ++ \ id \mid id \ ++ \mid (\ E \) \mid num \mid id \mid id(ArgList)$

$E \rightarrow E7$
 $E7 \rightarrow E6 \ E7'$
 $E7' \rightarrow \mid \mid E6 \ E7' \mid \epsilon$
 $E6 \rightarrow E5 \ E6'$
 $E6' \rightarrow \&\& \ E5 \ E6' \mid \epsilon$
 $E5 \rightarrow E4 \ E5'$
 $E5' \rightarrow == \ E4 \mid != \ E4 \mid > \ E4 \mid < \ E4 \mid <= \ E4 \mid \epsilon$
 $E4 \rightarrow E3 \ E4'$
 $E4' \rightarrow + \ E3 \ E4' \mid - \ E3 \ E4' \mid \epsilon$
 $E3 \rightarrow E2 \ E3'$
 $E3' \rightarrow * \ E2 \ E3' \mid \epsilon$
 $E2 \rightarrow ! \ E1 \mid E1$
 $E1 \rightarrow ++ \ id \mid (\ E \) \mid num \mid id \ E1'$
 $E1' \rightarrow ++ \mid (\ ArgList \) \mid \epsilon$
 $ArgList \rightarrow E \ ArgList' \mid \epsilon$
 $ArgList' \rightarrow , \ E \ ArgList' \mid \epsilon$

Грамматика остальных конструкций языка MiniC

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

Транслирующие грамматики

Каждый нетерминал грамматики помимо прочих содержит наследуемый атрибут контекста (С или С'), 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	$E7'_{pq} \rightarrow E6_r \{OR\}_{prs} E7'_{st}$	$s = \text{alloc}(C); q = t$
4	$E7'_{pq} \rightarrow \epsilon$	$q = p$
5	$E6_p \rightarrow E5_q E6'_{rs}$	$r = q; p = s$
6	$E6'_{pq} \rightarrow \&\& E5_r \{AND\}_{prs} E6'_{st}$	$s = \text{alloc}(C); q = t$
7	$E6'_{pq} \rightarrow \epsilon$	$q = p$
8	$E5_p \rightarrow E4_q E5'_{rs}$	$r = q; p = s$
9	$E5'_{pq} \rightarrow == E4_r \{MOV\}_{1,,s} \{EQ\}_{pr1} \{MOV\}_{0,,s} \{LBL\}_{,,1}$	$s = \text{alloc}(C); l = \text{newLabel}(); q = s$
10	$E5'_{pq} \rightarrow != E4_r \{MOV\}_{1,,s} \{NE\}_{pr1} \{MOV\}_{0,,s} \{LBL\}_{,,1}$	$s = \text{alloc}(C); l = \text{newLabel}(); q = s$
11	$E5'_{pq} \rightarrow > E4_r \{MOV\}_{1,,s} \{GT\}_{pr1} \{MOV\}_{0,,s} \{LBL\}_{,,1}$	$s = \text{alloc}(C); l = \text{newLabel}(); q = s$
12	$E5'_{pq} \rightarrow < E4_r \{MOV\}_{1,,s} \{LT\}_{pr1} \{MOV\}_{0,,s} \{LBL\}_{,,1}$	$s = \text{alloc}(C); l = \text{newLabel}(); q = s$
13	$E5'_{pq} \rightarrow <= E4_r \{MOV\}_{1,,s} \{LE\}_{pr1} \{MOV\}_{0,,s} \{LBL\}_{,,1}$	$s = \text{alloc}(C); l = \text{newLabel}(); q = s$
14	$E5'_{pq} \rightarrow \epsilon$	$q = p$
15	$E4_p \rightarrow E3_q E4'_{rs}$	$r = q; p = s$
16	$E4'_{pq} \rightarrow + E3_r \{ADD\}_{prs} E4'_{st}$	$s = \text{alloc}(C); q = t$
17	$E4'_{pq} \rightarrow - E3_r \{SUB\}_{prs} E4'_{st}$	$s = \text{alloc}(C); q = t$
18	$E4'_{pq} \rightarrow \epsilon$	$q = p$
19	$E3_p \rightarrow E2_q E3'_{rs}$	$r = q; p = s$
20	$E3'_{pq} \rightarrow * E2_r \{MUL\}_{prs} E3'_{st}$	$s = \text{alloc}(C); q = t$
21	$E3'_{pq} \rightarrow \epsilon$	$q = p$
22	$E2_p \rightarrow ! E1_q \{NOT\}_{q,,r}$	$r = \text{alloc}(C); p = r$
23	$E2_p \rightarrow E1_q$	$p = q$
24	$E1_p \rightarrow (E_q)$	$p = q$
25	$E1_p \rightarrow \text{num}_{val}$	$p = \text{val}$
26	$E1_p \rightarrow \text{chr}_{val}$	$p = \text{val}$
27	$E1_p \rightarrow ++ \text{id}_{name} \{ADD\}_{q1q}$	$q = \text{checkVar}(C, \text{name}); p = q$
28	$E1_p \rightarrow \text{id}_{name} E1'_{rs}$	$r = \text{name}; p = s$
29	$E1'_{pq} \rightarrow ++ \{MOV\}_{s,,r} \{ADD\}_{s1s}$	$s = \text{checkVar}(C, p); r = \text{alloc}(C); q = r$
30	$E1'_{pq} \rightarrow (\text{ArgList}_n) \{CALL\}_{s,r}$	$s = \text{checkFunc}(p, n); r = \text{alloc}(C); q = r$
31	$E1'_{pq} \rightarrow \epsilon$	$q = \text{checkVar}(C, p)$
32	$\text{ArgList}_n \rightarrow E_p \text{ArgList}'_m \{PARAM\}_{,,p}$	$n = m + 1$
33	$\text{ArgList}_n \rightarrow \epsilon$	$n = 0$
34	$\text{ArgList}'_n \rightarrow , E_p \text{ArgList}'_m \{PARAM\}_{,,p}$	$n = m + 1$
35	$\text{ArgList}'_n \rightarrow \epsilon$	$n = 0$

Транслирующая грамматика для основных конструкций языка MiniC

№	Правило	Семантическое определение
1	$\text{DeclareStmt} \rightarrow \text{Type}_p \text{ id}_{\text{name}} \text{ DeclareStmt}'_{qr}$	$q = p; r = \text{name}$
2	$\text{DeclareStmt}'_{pq} \rightarrow (\text{ ParamList}_{nC'}) \{ \text{ StmtList}_{C'} \} \{\text{RET}\},, '0'$	if $C > -1$: error (function definition inside function) else: $C' = \text{addFunc}(q, p, n)$
3	$\text{DeclareStmt}'_{pq} \rightarrow = \text{ num}_{\text{val}} \text{ DeclVarList}'_r ;$	$\text{addVar}(q, C, p, \text{val}); r = p$
4	$\text{DeclareStmt}'_{pq} \rightarrow \text{ DeclVarList}'_r ;$	$\text{addVar}(q, C, p); r = p$
5	$\text{Type}_p \rightarrow \text{char}$	$p = \text{char}$
6	$\text{Type}_p \rightarrow \text{int}$	$p = \text{int}$
7	$\text{DeclVarList}'_p \rightarrow , \text{ id}_{\text{name}} \text{ InitVar}_{rs} \text{ DeclVarList}'_t$	$r = p; s = \text{name}; t = p$
8	$\text{DeclVarList}'_p \rightarrow \epsilon$	
9	$\text{InitVar}_{pq} \rightarrow = \text{ num}_{\text{val}} \text{ ИЛИ } = \text{ chr}_{\text{val}}$	$\text{addVar}(q, C, p, \text{val})$
10	$\text{InitVar}_{pq} \rightarrow \epsilon$	$\text{addVar}(q, C, p)$
11	$\text{ParamList}_n \rightarrow \text{Type}_q \text{ id}_{\text{name}} \text{ ParamList}'_s$	$\text{addVar}(\text{name}, C, q); n = s + 1$
12	$\text{ParamList}_n \rightarrow \epsilon$	$n = 0$
13	$\text{ParamList}'_n \rightarrow , \text{ Type}_q \text{ id}_{\text{name}} \text{ ParamList}'_s$	$\text{addVar}(\text{name}, C, q); n = s + 1$
14	$\text{ParamList}'_n \rightarrow \epsilon$	$n = 0$
15	$\text{StmtList} \rightarrow \text{Stmt} \text{ StmtList}$	стартовый нетерминал
16	$\text{StmtList} \rightarrow \epsilon$	
17	$\text{Stmt} \rightarrow \text{DeclareStmt}$	if $C == -1$: error (operator should be inside function)
18	$\text{Stmt} \rightarrow \text{AssignOrCallOp}$	
19	$\text{Stmt} \rightarrow \text{WhileOp}$	
20	$\text{Stmt} \rightarrow \text{ForOp}$	
21	$\text{Stmt} \rightarrow \text{IfOp}$	
22	$\text{Stmt} \rightarrow \text{SwitchOp}$	
23	$\text{Stmt} \rightarrow \text{IOp}$	
24	$\text{Stmt} \rightarrow \text{OOp}$	
25	$\text{Stmt} \rightarrow \{ \text{ StmtList} \}$	
26	$\text{Stmt} \rightarrow \text{return } E_p \{\text{RET}\},, p ;$	
27	$\text{Stmt} \rightarrow ;$	
28	$\text{AssignOrCallOp} \rightarrow \text{AssignOrCall} ;$	
29	$\text{AssignOrCall} \rightarrow \text{ id}_{\text{name}} \text{ AssignOrCall}'_q$	$q = \text{name}$
30	$\text{AssignOrCall}'_p \rightarrow = E_q \{\text{MOV}\}_{q,r}$	$r = \text{checkVar}(C, p)$
31	$\text{AssignOrCall}'_p \rightarrow (\text{ ArgList}_n) \{\text{CALL}\}_{q,r}$	$q = \text{checkFunc}(p, n); r = \text{alloc}(C)$
32	$\text{WhileOp} \rightarrow \text{while } \{\text{LBL}\},, 11 (E_p) \{\text{EQ}\}_{p,0,12} \text{ Stmt} \{\text{JMP}\},, 11 \{\text{LBL}\},, 12$	$11 = \text{newLabel}(); 12 = \text{newLabel}()$
33	$\text{ForOp} \rightarrow \text{for} (\text{ ForInit} ; \{\text{LBL}\},, 11 \text{ ForExp}_p ; \{\text{EQ}\}_{p,0,14} \{\text{JMP}\},, 13 \{\text{LBL}\},, 12 \text{ ForLoop} \{\text{JMP}\},, 11) \{\text{LBL}\},, 13 \text{ Stmt} \{\text{JMP}\},, 12 \{\text{LBL}\},, 14$	$11 = \text{newLabel}(); 12 = \text{newLabel}()$ $13 = \text{newLabel}(); 14 = \text{newLabel}()$
34	$\text{ForInit} \rightarrow \text{Assign}$	
35	$\text{ForInit} \rightarrow \epsilon$	
36	$\text{ForExp}_p \rightarrow E_q$	$p = q$
37	$\text{ForExp}_p \rightarrow \epsilon$	$p = '1'$
38	$\text{ForLoop} \rightarrow \text{Assign}$	
39	$\text{ForLoop} \rightarrow ++ \text{ id}_{\text{name}} \{\text{ADD}\}_{p,p}$	$p = \text{checkVar}(C, \text{name})$

40	ForLoop $\rightarrow \epsilon$	
41	IfOp \rightarrow if (E _p) {EQ} _{p,0,11} Stmt {JMP} _{,,12} {LBL} _{,,11} ElsePart {LBL} _{,,12}	l1 = newLabel(); l2 = newLabel()
42	ElsePart \rightarrow else Stmt	
43	ElsePart $\rightarrow \epsilon$	
44	SwitchOp \rightarrow switch (E _p) { Cases _{q,end} } {LBL} _{,,end}	q = p; end = newLabel()
45	Cases _{p,end} \rightarrow ACase _{q,end1,def1} Cases' _{r,end2,def2}	q = r = p; end1 = end2 = end; def2 = def1
46	Cases' _{p,end,def} \rightarrow 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} $\rightarrow \epsilon$ {JMP} _{,,q}	if (def >= 0) q = def; else q = end
48	ACase _{p,end,def} \rightarrow case num _{val} {NE} _{p,val,next} : Stmt {JMP} _{,,end} {LBL} _{,,next}	next = newLabel(); def = -1
49	ACase _{p,end,def} \rightarrow default : {JMP} _{,,next} {LBL} _{,,def} Stmt {JMP} _{,,end} {LBL} _{,,next}	next = newLabel(); def = newLabel()
50	IOp \rightarrow in id _{name} ; {IN} _{,,p}	p = checkVar(C,name)
51	OOp \rightarrow out OOp' ;	
52	OOp' \rightarrow E _p {OUT} _{,,p}	
53	OOp' \rightarrow 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 (= 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, len)
Вызывается для объявления функции. Работает только в глобальном контексте. Если идентификатор name уже объявлен (как функция или переменная), то возвращает false. Работает аналогично setIdType, но устанавливает kind = func и количество параметров len. 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 score идентификатор с именем name; если не находит и score > -1, то повторяет поиск для score = -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 checkId(), 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.