%{

#include <iostream>

#include "classesnodes.h"

void yyerror(char const\* s);

extern int yylex(void);

using namespace std;

%}

/\* удален union и типы \*/

%token BOOL CHAR FLOAT INT STRING

%token TRUE FALSE CHAR\_LITERAL INT\_LITERAL FLOAT\_LITERAL STRING\_LITERAL RAW\_STRING\_LITERAL

%token ID

%token FOR LOOP WHILE IN CONTINUE

%token IF ELSE

%token LET MUT CONST

%token FN ENUM STRUCT TRAIT

%token IMPL SELF BIG\_SELF PUB SELF\_REF MUT\_SELF\_REF MUT\_REF MOD SUPER

%token ';' RIGHT\_ARROW

/\* BREAK и RETURN в документации почему-то присуствует в приоритетах операций. Стоит наверное с этим разобраться \*/

%nonassoc BREAK RETURN

%nonassoc '{' '}'

%right ':'

%right '='

%nonassoc RANGE /\* .. \*/

%left OR /\* || \*/

%left AND /\* && \*/

%left '<' '>' EQUAL NOT\_EQUAL LESS\_EQUAL GREATER\_EQUAL // == != <= >=

%left '+' '-'

%left '\*' '/' '%'

%left '!' '&' UMINUS USTAR /\* - \* \*/

%nonassoc '?'

%left '.' '[' DOUBLEDOTS

%nonassoc '(' ')'

%start Program

%%

/\* ---------------------- PROGRAM --------------------------- \*/

Program: ItemListEmpty { $$ = new ProgramNode($1); }

/\* Необходимо уточнить, надо ли как-то обозначить, что Stmt все должны быть Item, иначе программа не заработает \*/

/\* ----------------------------- STATEMENT ----------------------------- \*/

StmtList: Stmt { $$ = new StmtListNode($1); }

| StmtList Stmt { $$ = StmtListNode::Append($1, $2); }

;

Stmt: ';' { $$ = new StmtNode(StmtNode::semicolon, 0, 0, 0); }

| LetStmt { $$ = new StmtNode(StmtNode::let, 0, 0, $1); }

| ExprStmt { $$ = new StmtNode(StmtNode::exprstmt, $1);}

;

ItemListEmpty: /\*empty\*/ { $$ = 0; }

| ItemList { $$ = new ItemListNode($1); }

;

ItemList: Item { $$ = new ItemListNode($1); }

| ItemList Item { $$ = ItemListNode::Append($1, $2); }

;

Item: SimpleItem { $$ = new ItemNode(self, $1);}

| Visibility SimpleItem { $$ = new ItemNode($1, $2);}

;

SimpleItem: FuncStmt { $$ = ItemNode::DeclarationFunction(self, $1); }

| StructStmt { $$ = ItemNode::DeclarationStruct(self, $1); }

| EnumStmt { $$ = ItemNode::DeclarationEnum(self, $1); }

| ImplStmt { $$ = ItemNode::DeclarationImpl(self, $1); }

| TraitStmt { $$ = ItemNode::DeclarationTrait(self, $1); }

| ConstStmt { $$ = ItemNode::DeclarationConst(self, $1); }

| ModuleStmt { $$ = ItemNode::DeclarationModule(self, $1); }

;

/\* ---------- Function ------------ \*/

FuncStmt: DecFuncStmt { $$ = $1; }

| ImplFuncStmt { $$ = $1; }

;

DecFuncStmt: FN ID '(' FuncParamListEmpty ')' ';' { $$ = new FuncStmtNode($2, 0, $4, 0); }

| FN ID '(' FuncParamListEmpty ')' RIGHT\_ARROW Type ';' { $$ = new FuncStmtNode($2, $7, $4, 0); }

;

ImplFuncStmt: FN ID '(' FuncParamListEmpty ')' BlockExpr { $$ = new FuncStmtNode($2, 0, $4, $6); }

| FN ID '(' FuncParamListEmpty ')' RIGHT\_ARROW Type BlockExpr { $$ = new FuncStmtNode($2, $7, $4, $8); }

;

FuncParamListEmpty: /\* empty \*/ { $$ = FuncParamListNode::FunctionParamsFinal(FuncParamListNode::associated, 0); }

| FuncParamList { $$ = FuncParamListNode::FunctionParamsFinal(FuncParamListNode::associated, $1); }

;

FuncParamList: SELF { $$ = FuncParamListNode::FunctionParamsFinal(FuncParamListNode::self, 0); }

| SELF\_REF { $$ = FuncParamListNode::FunctionParamsFinal(FuncParamListNode::self\_ref, 0); }

| MUT\_SELF\_REF { $$ = FuncParamListNode::FunctionParamsFinal(FuncParamListNode::mut\_self\_ref, 0); }

| FuncParam { $$ = new FuncParamListNode($1); }

| FuncParamList ',' FuncParam { $$ = FuncParamListNode::Append($1, $3); }

;

FuncParam: ID ':' Type { $$ = new FuncParamNode($1, $3, FuncParamNode::noMut); }

| MUT ID ':' Type { $$ = new FuncParamNode($2, $4, FuncParamNode::mut); }

| ID ':' MUT\_REF Type { $$ = new FuncParamNode($1, $4, FuncParamNode::mut\_ref); }

| ID ':' '&' Type { $$ = new FuncParamNode($1, $4, FuncParamNode::link); }

;

/\* ========== Struct =========== \*/

StructStmt: StructStruct { $$ = $1; }

| TupleStruct { $$ = $1; }

;

StructStruct : STRUCT ID '{' StructFieldListEmpty '}' { $$ = new StructStructNode($2, $4); }

| STRUCT ID ';' { $$ = new StructStructNode($2, 0); }

;

StructFieldListEmpty: /\* empty \*/ { $$ = 0; }

| StructFieldList { $$ = new StructFieldListNode($1); }

| StructFieldList ',' { $$ = new StructFieldListNode($1); }

;

StructFieldList: StructField { $$ = new StructFieldListNode($1); }

| StructFieldList ',' StructField { $$ = StructFieldListNode::Append($1, $3); }

;

StructField: ID ':' Type { $$ = new StructFieldNode($1, $3, self); }

| Visibility ID ':' Type { $$ = new StructFieldNode($2, $4, $1); }

;

/\*--- TupleStruct ----\*/

TupleStruct: STRUCT ID '(' TupleFieldListEmpty ')' { $$ = new StructStructNode($2, $4); }

;

TupleFieldListEmpty: /\* empty \*/ { $$ = 0; }

| TupleFieldList { $$ = new StructFieldListNode($1); }

| TupleFieldList ',' { $$ = new StructFieldListNode($1); }

;

TupleFieldList: Type { $$ = new StructFieldListNode(new StructFieldNode(0, $1, self)); }

| Visibility Type { $$ = new StructFieldListNode(new StructFieldNode(0, $2, $1)); }

| TupleFieldList ',' Type { $$ = StructFieldListNode::Append($1, new StructFieldNode(0, $3, self)); }

| TupleFieldList ',' Visibility Type { $$ = StructFieldListNode::Append($1, new StructFieldNode(0, $4, $3)); }

;

/\* ============= ENUM ================ \*/

EnumStmt: ENUM ID '{' EnumItemListEmpty '}' { $$ = new EnumStmtNode($2, $4); }

;

EnumItemListEmpty: /\* empty \*/ { $$ = 0; }

| ',' { $$ = 0; }

| EnumItemList { $$ = new EnumItemListNode($1); }

| EnumItemList ',' { $$ = new EnumItemListNode($1); }

;

EnumItemList: EnumItem { $$ = new EnumItemListNode($1); }

| EnumItemList ',' EnumItem { $$ = EnumItemListNode::Append($1, $3); }

;

EnumItem: ID { $$ = new EnumItemNode($1, self, 0, 0); }

| Visibility ID { $$ = new EnumItemNode($2, $1, 0, 0); }

| ID '=' ExprWithBlock { $$ = new EnumItemNode($1, self, NULL, $3); } /\* В таком случае ID должен быть всегда только целочисленным числом. Нельзя на парсере определить такое\*/

| ID '=' ExprWithoutBlock { $$ = new EnumItemNode($1, self, 0, $3); }

| Visibility ID '=' ExprWithBlock { $$ = new EnumItemNode($2, $1, 0, $4); }

| Visibility ID '=' ExprWithoutBlock { $$ = new EnumItemNode($2, $1, 0, $4); }

| Visibility ID '{' StructFieldListEmpty '}' { $$ = new EnumItemNode($2, $1, $4, 0); }

| ID '{' StructFieldListEmpty '}' { $$ = new EnumItemNode($1, self, $3, 0); }

;

/\* =========== IMPL ================ \*/

ImplStmt: IMPL Type '{'AssociatedItemListEmpty '}' { $$ = new ImplStmtNode(ImplStmtNode::inherent, $2, 0, $4); }

| IMPL ID FOR Type '{'AssociatedItemListEmpty '}' { $$ = new ImplStmtNode(ImplStmtNode::trait, $4, $2, $6); }

;

AssociatedItemListEmpty: /\* empty \*/ { $$ = 0; }

| AssociatedItemList { $$ = new AssociatedItemListNode($1); }

;

AssociatedItemList: AssociatedItem { $$ = new AssociatedItemListNode($1); }

| AssociatedItemList AssociatedItem { $$ = AssociatedItemListNode::Append($1, $2); }

;

/\* Необходима еще проверка для Impl то что FuncStmt является именно реализацией \*/

AssociatedItem: FuncStmt { $$ = new AssociatedItemNode(self, $1, 0); }

| ConstStmt { $$ = new AssociatedItemNode(self, 0, $1); }

| Visibility FuncStmt { $$ = new AssociatedItemNode($1, $2, 0); }

| Visibility ConstStmt { $$ = new AssociatedItemNode($1, 0, $2); }

;

/\* ============ TRAIT ================ \*/

TraitStmt: TRAIT ID '{' AssociatedItemListEmpty '}' { $$ = new TraitNode($2, $4); }

;

/\* ============ CONST =============== \*/

ConstStmt: CONST ID ':' Type '=' ExprWithBlock ';' { $$ = ConstStmtNode::ConstStmt($2, $4, $6); }

| CONST ID ':' Type '=' ExprWithoutBlock ';' { $$ = ConstStmtNode::ConstStmt($2, $4, $6); }

| CONST ID ':' Type ';' { $$ = ConstStmtNode::ConstStmt($2, $4, 0); }

;

/\* =========== Module ================= \*/

ModuleStmt: MOD ID ';' { $$ = new ModuleStmtNode(ModuleStmtNode::empty, $2, 0); }

| MOD ID '{' ItemListEmpty '}' { $$ = new ModuleStmtNode(ModuleStmtNode::empty, $2, $4); }

;

/\* ========= LetStmt ============ \*/

LetStmt: LET ID '=' ExprWithBlock ';' { $$ = new LetStmtNode($2, 0, LetStmtNode::noMut, $4); }

| LET ID '=' ExprWithoutBlock ';' { $$ = new LetStmtNode($2, 0, LetStmtNode::noMut, $4); }

| LET ID ':' Type '=' ExprWithBlock ';' { $$ = new LetStmtNode($2, $4, LetStmtNode::noMut, $6); }

| LET ID ':' Type '=' ExprWithoutBlock ';' { $$ = new LetStmtNode($2, $4, LetStmtNode::noMut, $6); }

| LET MUT ID ';' { $$ = new LetStmtNode($3, 0, LetStmtNode::mut, 0); }

| LET MUT ID ':' Type ';' { $$ = new LetStmtNode($3, $5, LetStmtNode::mut, 0); }

| LET MUT ID '=' ExprWithBlock ';' { $$ = new LetStmtNode($3, 0, LetStmtNode::mut, $5); }

| LET MUT ID '=' ExprWithoutBlock ';' { $$ = new LetStmtNode($3, 0, LetStmtNode::mut, $5); }

| LET MUT ID ':' Type '=' ExprWithBlock ';' { $$ = new LetStmtNode($3, $5, LetStmtNode::mut, $7); }

| LET MUT ID ':' Type '=' ExprWithoutBlock ';' { $$ = new LetStmtNode($3, $5, LetStmtNode::mut, $7); }

;

/\* === Expression Statement === \*/

ExprStmt: ExprWithoutBlock ';' {$$ = new StmtNode(StmtNode::expression, $1, 0, 0);}

| ExprWithBlock ';' {$$ = new StmtNode(StmtNode::expression, $1, 0, 0);}

;

/\*----------------------- EXPRESSION ---------------------- \*/

ExprListEmpty: /\*empty\*/ { $$ = 0; }

| ExprList ',' { $$ = new ExprListNode($1); }

| ExprList { $$ = new ExprListNode($1); }

;

ExprList: ExprWithBlock { $$ = new ExprListNode($1); }

| ExprWithoutBlock { $$ = new ExprListNode($1); }

| ExprList ',' ExprWithBlock { $$ = ExprListNode::Append($1, $3); }

| ExprList ',' ExprWithoutBlock { $$ = ExprListNode::Append($1, $3); }

;

ExprWithoutBlock: CHAR\_LITERAL { $$ = ExprNode::ExprFromCharLiteral(ExprNode::char\_lit, $1); }

| STRING\_LITERAL { $$ = ExprNode::ExprFromStringLiteral(ExprNode::string\_lit, $1); }

| RAW\_STRING\_LITERAL { $$ = ExprNode::ExprFromStringLiteral(ExprNode::raw\_string\_lit, $1); }

| INT\_LITERAL { $$ = ExprNode::ExprFromIntLiteral(ExprNode::int\_lit, $1); }

| FLOAT\_LITERAL { $$ = ExprNode::ExprFromFloatLiteral(ExprNode::float\_lit, $1); }

| TRUE { $$ = ExprNode::ExprFromBoolLiteral(ExprNode::bool\_lit, $1); }

| FALSE { $$ = ExprNode::ExprFromBoolLiteral(ExprNode::bool\_lit, $1); }

| ExprWithoutBlock '+' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::plus, $1, $3); }

| ExprWithoutBlock '+' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::plus, $1, $3); }

| ExprWithBlock '+' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::plus, $1, $3); }

| ExprWithBlock '+' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::plus, $1, $3); }

| ExprWithoutBlock '-' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::minus, $1, $3); }

| ExprWithoutBlock '-' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::minus, $1, $3); }

| ExprWithBlock '-' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::minus, $1, $3); }

| ExprWithBlock '-' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::minus, $1, $3); }

| ExprWithoutBlock '/' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::div\_expr, $1, $3); }

| ExprWithoutBlock '/' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::div\_expr, $1, $3); }

| ExprWithBlock '/' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::div\_expr, $1, $3); }

| ExprWithBlock '/' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::div\_expr, $1, $3); }

| ExprWithoutBlock '\*' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::mul\_expr, $1, $3); }

| ExprWithoutBlock '\*' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::mul\_expr, $1, $3); }

| ExprWithBlock '\*' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::mul\_expr, $1, $3); }

| ExprWithBlock '\*' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::mul\_expr, $1, $3); }

| ExprWithoutBlock '%' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::mod, $1, $3); }

| ExprWithoutBlock '%' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::mod, $1, $3); }

| ExprWithBlock '%' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::mod, $1, $3); }

| ExprWithBlock '%' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::mod, $1, $3); }

| ExprWithoutBlock AND ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::and\_, $1, $3); }

| ExprWithoutBlock AND ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::and\_, $1, $3); }

| ExprWithBlock AND ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::and\_, $1, $3); }

| ExprWithBlock AND ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::and\_, $1, $3); }

| ExprWithoutBlock OR ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::or\_, $1, $3); }

| ExprWithoutBlock OR ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::or\_, $1, $3); }

| ExprWithBlock OR ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::or\_, $1, $3); }

| ExprWithBlock OR ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::or\_, $1, $3); }

| ExprWithoutBlock '=' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::asign, $1, $3); }

| ExprWithoutBlock '=' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::asign, $1, $3); }

| ExprWithBlock '=' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::asign, $1, $3); }

| ExprWithBlock '=' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::asign, $1, $3); }

| ExprWithoutBlock EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::equal, $1, $3); }

| ExprWithoutBlock EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::equal, $1, $3); }

| ExprWithBlock EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::equal, $1, $3); }

| ExprWithBlock EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::equal, $1, $3); }

| ExprWithoutBlock NOT\_EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::not\_equal, $1, $3); }

| ExprWithoutBlock NOT\_EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::not\_equal, $1, $3); }

| ExprWithBlock NOT\_EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::not\_equal, $1, $3); }

| ExprWithBlock NOT\_EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::not\_equal, $1, $3); }

| ExprWithoutBlock '>' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater, $1, $3); }

| ExprWithoutBlock '>' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater, $1, $3); }

| ExprWithBlock '>' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater, $1, $3); }

| ExprWithBlock '>' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater, $1, $3); }

| ExprWithoutBlock '<' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::less, $1, $3); }

| ExprWithoutBlock '<' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::less, $1, $3); }

| ExprWithBlock '<' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::less, $1, $3); }

| ExprWithBlock '<' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::less, $1, $3); }

| ExprWithoutBlock GREATER\_EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater\_equal, $1, $3); }

| ExprWithoutBlock GREATER\_EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater\_equal, $1, $3); }

| ExprWithBlock GREATER\_EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater\_equal, $1, $3); }

| ExprWithBlock GREATER\_EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::greater\_equal, $1, $3); }

| ExprWithoutBlock LESS\_EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::less\_equal, $1, $3); }

| ExprWithoutBlock LESS\_EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::less\_equal, $1, $3); }

| ExprWithBlock LESS\_EQUAL ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::less\_equal, $1, $3); }

| ExprWithBlock LESS\_EQUAL ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::less\_equal, $1, $3); }

| '-' ExprWithoutBlock %prec UMINUS { $$ = ExprNode::OperatorExpr(ExprNode::uminus, $2, 0); }

| '-' ExprWithBlock %prec UMINUS { $$ = ExprNode::OperatorExpr(ExprNode::uminus, $2, 0); }

| '!' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::negotation, $2, 0); }

| '!' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::negotation, $2, 0); }

| ExprWithoutBlock '?' { $$ = ExprNode::OperatorExpr(ExprNode::question, $1, 0); }

| ExprWithBlock '?' { $$ = ExprNode::OperatorExpr(ExprNode::question, $1, 0); }

| '\*' ExprWithoutBlock %prec USTAR { $$ = ExprNode::OperatorExpr(ExprNode::ustar, $2, 0); }

| '\*' ExprWithBlock %prec USTAR { $$ = ExprNode::OperatorExpr(ExprNode::ustar, $2, 0); }

| '&' ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::link, $2, 0); }

| '&' ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::link, $2, 0); }

| '[' ExprListEmpty ']' { $$ = ExprNode::ArrExprFromList(ExprNode::array\_expr, $2); }

| '[' ExprWithoutBlock ';' ExprWithoutBlock ']' { $$ = ExprNode::ArrExprAutoFill(ExprNode::array\_expr\_auto\_fill, $2, $4); }

| '[' ExprWithoutBlock ';' ExprWithBlock ']' { $$ = ExprNode::ArrExprAutoFill(ExprNode::array\_expr\_auto\_fill, $2, $4); }

| '[' ExprWithBlock ';' ExprWithoutBlock ']' { $$ = ExprNode::ArrExprAutoFill(ExprNode::array\_expr\_auto\_fill, $2, $4); }

| '[' ExprWithBlock ';' ExprWithBlock ']' { $$ = ExprNode::ArrExprAutoFill(ExprNode::array\_expr\_auto\_fill, $2, $4); }

| ExprWithoutBlock '[' ExprWithoutBlock ']' { $$ = ExprNode::OperatorExpr(ExprNode::index\_expr, $1, $3); }

| ExprWithoutBlock '[' ExprWithBlock ']' { $$ = ExprNode::OperatorExpr(ExprNode::index\_expr, $1, $3); }

| ExprWithBlock '[' ExprWithoutBlock ']' { $$ = ExprNode::OperatorExpr(ExprNode::index\_expr, $1, $3); }

| ExprWithBlock '[' ExprWithBlock ']' { $$ = ExprNode::OperatorExpr(ExprNode::index\_expr, $1, $3); }

| ExprWithoutBlock '.' INT\_LITERAL { $$ = ExprNode::TupleExpr(ExprNode::tuple\_expr, $1, $3); }

| ExprWithBlock '.' INT\_LITERAL { $$ = ExprNode::TupleExpr(ExprNode::tuple\_expr, $1, $3); }

| CONTINUE { $$ = ExprNode::OperatorExpr(ExprNode::continue\_expr, 0, 0); }

| BREAK { $$ = ExprNode::OperatorExpr(ExprNode::break\_expr, 0, 0); }

| RANGE { $$ = ExprNode::RangeExpr(ExprNode::range\_right, 0, 0); }

| RANGE ExprWithoutBlock { $$ = ExprNode::RangeExpr(ExprNode::range\_right, $2, 0); }

| RANGE ExprWithBlock { $$ = ExprNode::RangeExpr(ExprNode::range\_right, $2, 0); }

| ExprWithoutBlock RANGE { $$ = ExprNode::RangeExpr(ExprNode::range\_left, $1, 0); }

| ExprWithBlock RANGE { $$ = ExprNode::RangeExpr(ExprNode::range\_left, $1, 0); }

| ExprWithoutBlock RANGE ExprWithoutBlock { $$ = ExprNode::RangeExpr(ExprNode::range\_expr, $1, $3); }

| ExprWithoutBlock RANGE ExprWithBlock { $$ = ExprNode::RangeExpr(ExprNode::range\_expr, $1, $3); }

| ExprWithBlock RANGE ExprWithoutBlock { $$ = ExprNode::RangeExpr(ExprNode::range\_expr, $1, $3); }

| ExprWithBlock RANGE ExprWithBlock { $$ = ExprNode::RangeExpr(ExprNode::range\_expr, $1, $3); }

| RETURN { $$ = ExprNode::OperatorExpr(ExprNode::return\_expr, 0, 0); }

| RETURN ExprWithoutBlock { $$ = ExprNode::OperatorExpr(ExprNode::return\_expr, $2, 0); }

| RETURN ExprWithBlock { $$ = ExprNode::OperatorExpr(ExprNode::return\_expr, $2, 0); }

| ExprWithoutBlock '.' ID { $$ = ExprNode::CallAccessExpr(ExprNode::field\_access\_expr, $3, $1, 0); }

| ExprWithBlock '.' ID { $$ = ExprNode::CallAccessExpr(ExprNode::field\_access\_expr, $3, $1, 0); }

| ExprWithoutBlock '.' ID '(' ExprListEmpty ')' { $$ = ExprNode::CallAccessExpr(ExprNode::method\_expr, $3, $1, $5); }

| ExprWithBlock '.' ID '(' ExprListEmpty ')' { $$ = ExprNode::CallAccessExpr(ExprNode::method\_expr, $3, $1, $5); }

| PathCallExpr { $$ = $1 }

| PathCallExpr '(' ExprListEmpty ')' { $$ = ExprNode::StaticMethod(ExprNode::static\_method, $1, $3); }

| PathCallExpr '{' StructExprFieldListEmpty '}' { $$ = ExprNode::FieldListAccess(ExprNode::static\_method, $1, $3); }

| '(' ExprWithBlock ')' { $$ = $2; }

|'(' ExprWithoutBlock ')' { $$ = $2; }

;

PathCallExpr: ID { $$ = ExprNode::CallAccessExpr(ExprNode::id\_, $1, 0, 0); }

| SUPER { $$ = ExprNode::CallAccessExpr(ExprNode::super\_expr, new string("super"), 0, 0); }

| SELF { $$ = ExprNode::CallAccessExpr(ExprNode::self\_expr, new string("self"), 0, 0); }

| PathCallExpr DOUBLEDOTS ID { $$ = ExprNode::PathCallExpr(ExprNode::path\_call\_expr, $3, $1); }

;

StructExprFieldListEmpty: /\*empty\*/ { $$ = 0; }

| StructExprFieldList { $$ = new ExprListNode($1); }

;

StructExprFieldList: ',' StructExprField { $$ = new ExprListNode($2); }

| StructExprFieldList ',' StructExprField { $$ = ExprListNode::Append($1, $3); }

;

StructExprField: /\*empty\*/ { $$ = 0; }

| ID ':' ExprWithoutBlock { $$ = ExprNode::ExprFromStructField(ExprNode::struct\_field\_expr, $1, $3); }

| ID ':' ExprWithBlock { $$ = ExprNode::ExprFromStructField(ExprNode::struct\_field\_expr, $1, $3); }

;

ExprWithBlock: BlockExpr { $$ = $1; }

| LoopExpr { $$ = $1; }

| IfExpr { $$ = $1; }

;

BlockExpr: '{' StmtList '}' { $$ = ExprNode::BlockExpr(ExprNode::block\_expr, 0, $2); }

| '{' ExprWithoutBlock '}' { $$ = ExprNode::BlockExpr(ExprNode::block\_expr, $2, 0); }

| '{' StmtList ExprWithoutBlock '}' { $$ = ExprNode::BlockExpr(ExprNode::block\_expr, $3, $2); }

| '{' '}' { $$ = ExprNode::BlockExpr(ExprNode::block\_expr, 0, 0); }

;

LoopExpr: InfiniteLoopExpr { $$ = $1; }

| PredicateLoopExpr { $$ = $1; }

| IteratorLoopExpr { $$ = $1; }

;

InfiniteLoopExpr: LOOP BlockExpr { $$ = ExprNode::CycleExpr(ExprNode::loop\_expr, 0, $2, 0); }

;

PredicateLoopExpr: WHILE '(' ExprWithBlock ')' BlockExpr { $$ = ExprNode::CycleExpr(ExprNode::loop\_while, $3, $5, 0); }

| WHILE '(' ExprWithoutBlock ')' BlockExpr { $$ = ExprNode::CycleExpr(ExprNode::loop\_while, $3, $5, 0); }

;

IteratorLoopExpr: FOR '(' ID IN ExprWithBlock ')' BlockExpr { $$ = ExprNode::CycleExpr(ExprNode::loop\_for, $5, $7, $3); }

| FOR '(' ID IN ExprWithoutBlock ')' BlockExpr { $$ = ExprNode::CycleExpr(ExprNode::loop\_for, $5, $7, $3); }

;

IfExpr: SimpleIfElseExpr { $$ = $1; }

| SimpleIfElseExpr ELSE BlockExpr { $$ = ExprNode::AddElseBlock($1, $3) }

;

SimpleIfElseExpr: SimpleIfExpr { $$ = $1; }

| SimpleIfElseExpr ELSE SimpleIfExpr { $$ = ExprNode::AddIfBlock($1, $3); }

;

SimpleIfExpr: IF '(' ExprWithoutBlock ')' BlockExpr { $$ = ExprNode::IfExpr(ExprNode::if\_expr, $3, $5); }

| IF '(' ExprWithBlock ')' BlockExpr { $$ = ExprNode::IfExpr(ExprNode::if\_expr, $3, $5); }

;

/\*-------------------------TYPE -------------------------- \*/

Type: BOOL { $$ = new TypeNode(TypeNode::bool\_); }

| CHAR { $$ = new TypeNode(TypeNode::char\_); }

| FLOAT { $$ = new TypeNode(TypeNode::float\_); }

| INT { $$ = new TypeNode(TypeNode::int\_); }

| STRING { $$ = new TypeNode(TypeNode::string\_); }

| ID { $$ = new TypeNode(TypeNode::id\_); }

| '[' Type ';' ExprWithBlock ']' { $$ = new TypeNode(TypeNode::array\_, $2, $4); }

| '[' Type ';' ExprWithoutBlock ']' { $$ = new TypeNode(TypeNode::array\_, $2, $4); }

;

/\*---------------------- VISIBILITY ------------------------- \*/

Visibility: PUB { $$ = pub; }

| PUB '(' SUPER ')' { $$ = self; }

| PUB '(' SELF ')' { $$ = super; }

;

%%

void yyerror(char const \*s)

{

printf("%s\n",s);

}