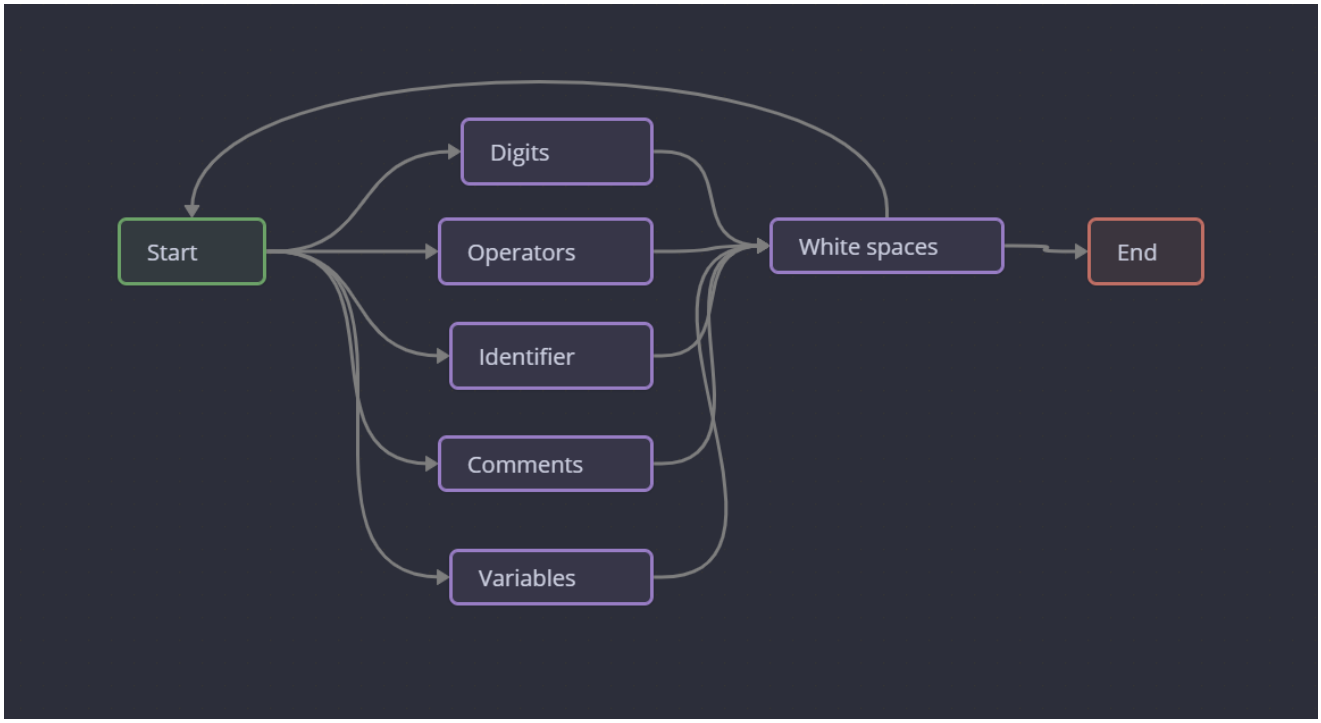
داک پروژه کامپایلر

اعضای گروه: سپهر شیرازی، سهیل سلیمی

فاز ۱

variables : `[a-z,A-Z_] ([0-9] | [a-z,A-Z_])\* | whitespace | end`
digits: `([0-9] | operators) ([0-9])\* | end`
identifiers: `bool | break | char | continue | else | false | for | if | int | print | return | true | and | or | not | whitespace | end`
comments: `// | /\* /w \*/ | whitespace | end`
whitespace: `space | tab | \n`
operators: `+ | - | * | / | % | > | < | = | <= | >= | != | == | ( | ) | [ | ] | { | } | ; | , | && | || | whitespace | end`
end: `\0`

ماشین گذار



فاز ۲

Grammar

```
Program -> Declarations
Declarations -> Declaration Declarations
Declarations -> ''
Declaration -> Type VarOrFunc
VarOrFunc -> Identifier VarOrFuncRest
VarOrFuncRest -> FunctionRest
VarOrFuncRest -> VarDeclRest
VarDeclRest -> Initialization MoreIdentifiers T_Semicolon
FunctionRest -> T_LP Parameters T_RP Block
Type -> T_Int
Type -> T_Bool
Type -> T_Char
MoreIdentifiers -> T_Comma Identifier Initialization MoreIdentifiers
MoreIdentifiers -> ''
Initialization -> T_Assign Expression
Initialization -> T_LB IntegerLiteral T_RB Initialization
Initialization -> ''
Functions -> Function Functions
Functions -> ''
Function -> Type Identifier FunctionRest
Parameters -> ParameterList
Parameters -> ''
ParameterList -> Parameter MoreParameters
Parameter -> Type Identifier
MoreParameters -> T_Comma Parameter MoreParameters
MoreParameters -> ''
Block -> T_LC Statements T_RC
Statements -> Statement Statements
Statements -> ''
Statement -> Declaration
Statement -> Assignment T_Semicolon
Statement -> IfStatement
Statement -> ForStatement
Statement -> PrintStatement T_Semicolon
Statement -> ReturnStatement T_Semicolon
Statement -> BreakStatement T_Semicolon
Statement -> ContinueStatement T_Semicolon
Assignment -> Identifier T_Assign Expression
IfStatement -> T_If T_LP Expression T_RP Block ElseIfs ElseBlock
ElseIfs -> ElseIf ElseIfs
ElseIfs -> ''
ElseIf -> T_Else T_If T_LP Expression T_RP Block
ElseBlock -> T_Else Block
ElseBlock -> ''
ForStatement -> T_For T_LP ForInit ForCondition T_Semicolon ForUpdate T_RP
Block
ForInit -> Assignment
ForInit -> Declaration
ForInit -> ''
```

```
ForCondition -> Expression
ForCondition -> ''
ForUpdate -> Assignment
ForUpdate -> ''
PrintStatement -> T_Print T_LP PrintArguments T_RP
PrintArguments -> StringLiteral MorePrintArguments
PrintArguments -> Expression
MorePrintArguments -> T_Comma Expression MorePrintArguments
MorePrintArguments -> ''
ReturnStatement -> T_Return Expression
BreakStatement -> T_Break
ContinueStatement -> T_Continue
Expression -> LogicalOr
LogicalOr -> LogicalAnd LogicalOrPRE
LogicalOrPRE  -> T_LOp_OR LogicalAnd LogicalOrPRE
LogicalOrPRE  -> ''
LogicalAnd -> Equality LogicalAndPRE
LogicalAndPRE  -> T_LOp_AND Equality LogicalAndPRE
LogicalAndPRE  -> ''
Equality -> Relational EqualityPRE
EqualityPRE ->  T_ROp_E Relational EqualityPRE
EqualityPRE  -> T_ROp_NE Relational EqualityPRE
EqualityPRE  -> ''
Relational -> Additive RelationalPRE
RelationalPRE ->  T_ROp_L Additive RelationalPRE
RelationalPRE ->  T_ROp_LE Additive RelationalPRE
RelationalPRE ->  T_ROp_G Additive RelationalPRE
RelationalPRE ->  T_ROp_GE Additive RelationalPRE
RelationalPRE ->  ''
Additive -> Multiplicative AdditivePRE
AdditivePRE  -> T_AOp_PL Multiplicative AdditivePRE
AdditivePRE  -> T_AOp_MN Multiplicative AdditivePRE
AdditivePRE  -> ''
Multiplicative -> Unary MultiplicativePRE
MultiplicativePRE ->  T_AOp_ML Unary MultiplicativePRE
MultiplicativePRE ->  T_AOp_DV Unary MultiplicativePRE
MultiplicativePRE ->  T_AOp_RM Unary MultiplicativePRE
MultiplicativePRE ->  ''
Unary -> T_LOp_NOT Unary
Unary -> Primary
Primary -> Identifier
Primary -> IntegerLiteral
Primary -> BooleanLiteral
Primary -> CharacterLiteral
Primary -> StringLiteral
Primary -> T_LP Expression T_RP
Identifier -> T_Id
IntegerLiteral -> T_Decimal
IntegerLiteral -> T_Hexadecimal
```

```
BooleanLiteral -> T_True
BooleanLiteral -> T_False
CharacterLiteral -> T_Character
StringLiteral -> T_String
```

- **Program**: اشاره دارد که `Declarations` نقطه شروع گرامر است و به معنای کل برنامه است. این قاعده به `Declaration` می‌تواند شامل یک یا چند باشد.
- **Declarations**: مجموعه‌ای از تعاریف است که می‌تواند شامل متغیرها، توابع و سایر اعلان‌ها باشد.
- **Declaration**: یک تعریف خاص است که می‌تواند یک `Type` و `VarOrFunc` باشد. `Type` می‌تواند `T_Int`, `T_Bool` یا `T_Char` باشد.
- **VarOrFunc**: تعیین می‌کند که آیا `Declaration` به یک متغیر یا تابع اشاره دارد.
- **VarDeclRest** و **FunctionRest**: تعیین می‌کنند که تعریف به یک تابع با پارامترها و بلاک کد یا یک اعلان متغیر با مقداردهی اولیه و سایر شناسه‌ها اشاره دارد.
- **Block**: بلاک کد است که می‌تواند شامل چندین `Statement` باشد.
- **Statement**: یک عملیات خاص در برنامه است که می‌تواند شامل اعلان‌ها، اختصاص‌ها، شرط‌ها، حلقه‌ها و غیره باشد.

و غیره

first and follow and predict table

ما از سایت hackingoff برای تبدیل استفاده کردیم

first

Non-Terminal Symbol	First Set
ε	ε
;	;
(	(
)	)
int	int
bool	bool
char	char
,	,
=	=
[	[
]	]
{	{

Non-Terminal Symbol	First Set
}	}
if	if
else	else
for	for
print	print
return	return
break	break
continue	continue
&	&
!	!
<	<
>	>
+	+
-	-
*	*
/	/
%	%
id	id
number	number
true	true
false	false
character	character
string	string
Declarations	ε, int, bool, char
VarDeclRest	;, =, [, ε, ,
FunctionRest	(
Type	int, bool, char
MoreIdentifiers	,, ε
Initialization	=, [, ε
Functions	ε, int, bool, char
Parameters	ε, int, bool, char
MoreParameters	,, ε
Block	{
Statements	ε, int, bool, char, if, print, break, id, return, for, continue

Non-Terminal Symbol	First Set
IfStatement	if
ElseIfs	ε, else
ElseIf	else
ElseBlock	else, ε
ForStatement	for
ForInit	ε, id, int, bool, char
ForCondition	ε, !, (, id, true, false, string, number, character
ForUpdate	ε, id
PrintStatement	print
MorePrintArguments	,, ε
ReturnStatement	return
BreakStatement	break
ContinueStatement	continue
LogicalOrPRE	ε, !, (, id, true, false, string, number, character
LogicalAndPRE	&, ε
EqualityPRE	=, !, ε
RelationalPRE	<, >, ε
AdditivePRE	+, -, ε
MultiplicativePRE	*, /, %, ε
Unary	!, (, id, true, false, string, number, character
Primary	(, id, true, false, string, number, character
Identifier	id
IntegerLiteral	number
BooleanLiteral	true, false
CharacterLiteral	character
StringLiteral	string
Declaration	int, bool, char
VarOrFunc	id
VarOrFuncRest	(, ;, =, [, ε, ,
Function	int, bool, char
Parameter	int, bool, char
Assignment	id
PrintArguments	string
ParameterList	int, bool, char

Non-Terminal Symbol	First Set
Statement	int, bool, char, if, print, break, id, return, for, continue
Program	ε, int, bool, char
Multiplicative	!, (, id, true, false, string, number, character
Additive	!, (, id, true, false, string, number, character
Relational	!, (, id, true, false, string, number, character
Equality	!, (, id, true, false, string, number, character
LogicalAnd	!, (, id, true, false, string, number, character
LogicalOr	!, (, id, true, false, string, number, character
Expression	!, (, id, true, false, string, number, character

follow

Non-Terminal Symbol	Follow Set
Program	$
Declarations	$
Declaration	int, bool, char, ;, if, print, break, id, return, for, continue, $, }
VarOrFunc	int, bool, char, ;, if, print, break, id, return, for, continue, $, }
VarOrFuncRest	int, bool, char, ;, if, print, break, id, return, for, continue, $, }
VarDeclRest	int, bool, char, ;, if, print, break, id, return, for, continue, $, }
FunctionRest	int, bool, char, ;, if, print, break, id, return, for, continue, $, }
Type	id
MoreIdentifiers	;
Initialization	,, ;
Functions	
Function	int, bool, char
Parameters	)
ParameterList	)
Parameter	,,)
MoreParameters	)
Block	!, (, id, true, false, string, number, character, else, int, bool, char, ;, if, print, break, return, for, continue, $, }
Statements	}
Statement	int, bool, char, if, print, break, id, return, for, continue, }

Non-Terminal Symbol	Follow Set
Assignment	;,)
IfStatement	int, bool, char, if, print, break, id, return, for, continue, }
ElseIfs	else, int, bool, char, if, print, break, id, return, for, continue, }
ElseIf	else, int, bool, char, if, print, break, id, return, for, continue, }
ElseBlock	int, bool, char, if, print, break, id, return, for, continue, }
ForStatement	int, bool, char, if, print, break, id, return, for, continue, }
ForInit	;
ForCondition	;
ForUpdate	)
PrintStatement	;
PrintArguments	)
MorePrintArguments	)
ReturnStatement	;
BreakStatement	;
ContinueStatement	;
Expression	), ,, ;
LogicalOr	), ,, ;
LogicalOrPRE	), ,, ;
LogicalAnd	!, (, id, true, false, string, number, character,), ,, ;
LogicalAndPRE	!, (, id, true, false, string, number, character,), ,, ;
Equality	&, !, (, id, true, false, string, number, character,), ,, ;
EqualityPRE	&, !, (, id, true, false, string, number, character,), ,, ;
Relational	=, !, &, (, id, true, false, string, number, character,), ,, ;
RelationalPRE	=, !, &, (, id, true, false, string, number, character,), ,, ;
Additive	<, >, =, !, &, (, id, true, false, string, number, character,), ,, ;
AdditivePRE	<, >, =, !, &, (, id, true, false, string, number, character,), ,, ;
Multiplicative	+, -, <, >, =, !, &, (, id, true, false, string, number, character,), ,, ;
MultiplicativePRE	+, -, <, >, =, !, &, (, id, true, false, string, number, character,), ,, ;
Unary	*, /, %, +, -, <, >, =, !, &, (, id, true, false, string, number, character,), ,, ;
Primary	*, /, %, +, -, <, >, =, !, &, (, id, true, false, string, number, character,), ,, ;
Identifier	=, (, [, ;, ,, *, /, %, +, -, <, >, !, &, id, true, false, string, number, character,)

Non-Terminal Symbol	Follow Set
IntegerLiteral	], *, /, %, +, -, <, >, =, !, &, (, id, true, false, string, number, character,), ,, ;
BooleanLiteral	*, /, %, +, -, <, >, =, !, &, (, id, true, false, string, number, character,), ,, ;
CharacterLiteral	*, /, %, +, -, <, >, =, !, &, (, id, true, false, string, number, character,), ,, ;
StringLiteral	,, *, /, %, +, -, <, >, =, !, &, (, id, true, false, string, number, character,), ;

predict table

Nonterminal	T_Semicolon	T_LP	T_R
Program			
Declarations			
Declaration			
VarOrFunc			
VarOrFuncRest	VarOrFuncRest -> VarDeclRestVarOrFuncRest -> VarDeclRest	VarOrFuncRest -> FunctionRestVarOrFuncRest -> VarDeclRest	
VarDeclRest	VarDeclRest -> Initialization MoreIdentifiers T_SemicolonVarDeclRest -> Initialization MoreIdentifiers T_Semicolon	VarDeclRest -> Initialization MoreIdentifiers T_Semicolon	
FunctionRest		FunctionRest -> T_LP Parameters T_RP Block	
Type			
MoreIdentifiers	MoreIdentifiers -> ''		
Initialization	Initialization -> ''	Initialization -> ''	

Nonterminal	T_Semicolon	T_LP	T_R
Functions			
Function			
Parameters			Par
ParameterList			
Parameter			
MoreParameters			Mor
Block			
Statements			
Statement			
Assignment			
IfStatement			
ElseIfs			

Nonterminal	T_Semicolon	T_LP	T_F
ElseIf			
ElseBlock			
ForStatement			
ForInit	ForInit -> ''	ForInit -> ''	
ForCondition	ForCondition -> ''	ForCondition -> Expression	
ForUpdate			For
PrintStatement			
PrintArguments		PrintArguments -> Expression	
MorePrintArguments			Mor -> ''
ReturnStatement			
BreakStatement			
ContinueStatement			
Expression		Expression -> LogicalOr	
LogicalOr		LogicalOr -> LogicalAnd LogicalOrPRE	

Nonterminal	T_Semicolon	T_LP	T_R
LogicalOrPRE	LogicalOrPRE -> ''	LogicalOrPRE -> ''	Log
LogicalAnd		LogicalAnd -> Equality LogicalAndPRE	
LogicalAndPRE	LogicalAndPRE -> ''	LogicalAndPRE -> ''	Log
Equality		Equality -> Relational EqualityPRE	
EqualityPRE	EqualityPRE -> ''	EqualityPRE -> ''	Equ
Relational		Relational -> Additive RelationalPRE	
RelationalPRE	RelationalPRE -> ''	RelationalPRE -> ''	Rel
Additive		Additive -> Multiplicative AdditivePRE	
AdditivePRE	AdditivePRE -> ''	AdditivePRE -> ''	Add
Multiplicative		Multiplicative -> Unary MultiplicativePRE	
MultiplicativePRE	MultiplicativePRE -> ''	MultiplicativePRE -> ''	Mul ''
Unary		Unary -> Primary	

Nonterminal	T_Semicolon	T_LP	T_F
Primary		Primary -> T_LP Expression T_RP	
Identifier			
IntegerLiteral			
BooleanLiteral			
CharacterLiteral			
StringLiteral			

با استفاده از جدول های ساخته شده می توانیم انها را وارد کد کرده و استفاده کنیم

فاز ۳

کد شامل دو بخش اصلی است:

1. **تعریف ساختار Sem**: شامل تعریف ساختار `Sem` که درخت نمادین (AST) و جدول شناسه‌ها (ID Table) را نگهداری می‌کند.
2. **توابع و متدها**: شامل توابع و متدهای مختلف برای انجام عملیات تجزیه و تحلیل معنایی

ساختار `Sem` شامل دو عضو است:

- درخت `ast` : درخت نمادین (AST) که ساختار نحوی کد منبع را نگهداری می‌کند.
- جدول `ids_table` : جدول شناسه‌ها که نوع شناسه‌ها و پارامترهای آن‌ها را نگهداری می‌کند.

متد `parser` :

این متد تحلیل معنایی کد را انجام می‌دهد. ابتدا از تابع `post_order_traversal` برای پیمایش درخت نمادین و پر کردن جدول شناسه‌ها استفاده می‌کند. سپس جدول شناسه‌ها را چاپ کرده و بررسی می‌کند که آیا تابع اصلی (main) با نوع صحیح و بدون پارامتر تعریف شده است یا خیر.

متد `post_order_traversal` :

این تابع به صورت بازگشتی درخت نمادین را به صورت post-order پیمایش می‌کند. در هر گره، عملیات‌های مختلفی بر اساس نوع گره انجام می‌دهد که شامل بررسی و ثبت شناسه‌ها، بررسی تطابق انواع و بررسی تعداد پارامترها می‌شود.

توابع کمکی

کد شامل چندین تابع کمکی است که به تحلیل معنایی کمک می‌کنند:

- `find_types` : برای پیدا کردن نوع شناسه‌ها در گره‌های مختلف.

- `find_the_op` : برای پیدا کردن نوع عملیات در گره‌های مختلف.
- `count_prams` : برای شمارش تعداد پارامترهای یک تابع.
- `find_prams` : برای پیدا کردن پارامترهای یک تابع.

بررسی و افزایش شماره بلوک

```
if let SymbolTree::Token(t) = symbole {     if t.token == TokenType::T_LC {
*block_num = *block_num + 1;     } }
```

اگر گره فعلی یک توکن `LC` باشد، شماره بلوک را افزایش می‌دهد. این کار برای مدیریت بلوک‌های کد (مثل بلوک‌های توابع یا حلقه‌ها) است.

پردازش گره‌های غیرترمینال

```
if let SymbolTree::NonTerminal(data) = symbole {
        if data == &NonTerminal::Declaration {  /* پردازش گره‌های اعلان */  }
else if data == &NonTerminal::VarDeclRest {/*پردازش گره‌های باقیمانده اعلان
متغیر */ }
        else if data == &NonTerminal::ReturnStatement {         /* پردازش
گره‌های دستور بازگشت */    } }
```

اگر گره فعلی یک گره غیرترمینال باشد، بر اساس نوع گره عملیات مختلفی انجام می‌دهد.

پردازش گره‌های اعلان(Declaration)

```
if data == &NonTerminal::Declaration {
        let mut choil = node.children();
        let n = choil.next().unwrap();
        let n = n.last_child().unwrap();
        let name = if let SymbolTree::Token(t) =
n.first_child().unwrap().data() {t}                else {
unimplemented!();     };
        let n = choil.next().unwrap();
        let n = n.last_child().unwrap();
        let types = if let SymbolTree::Token(t) = n.data() {   t   }
                    else {         unimplemented!();     };
        if ids_table.contains_key(&(name.literal.clone(),
block_num.clone())) {               println!("two same var in a block {:?}",
&name);
        }
        else {
                let mut prams = vec![];
                find_prams(&node, &mut prams);
```

```
                prams.reverse();
                let mut typer = None;
                find_the_op(&node, &mut typer);
                if typer.is_some() && typer.unwrap() != types.token &&
n.parent().unwrap().parent().unwrap().parent().unwrap().data()
== &SymbolTree::NonTerminal(NonTerminal::Statement) {
println!("types dont match {:?}", types);
                }
            ids_table.insert((name.literal.clone(), block_num.clone()),
(types.token.clone(), prams),);
        }
}
```

این بخش گره‌های اعلان متغیرها و توابع را پردازش می‌کند:

1. ابتدا نام و نوع متغیر یا تابع را استخراج می‌کند.
2. بررسی می‌کند که آیا شناسه‌ای با همین نام در همان بلوک وجود دارد یا خیر.
3. پارامترهای تابع را پیدا کرده و معکوس می‌کند.
4. نوع عملیات‌ها را پیدا می‌کند و اگر نوع‌ها تطابق نداشته باشند، پیام خطا چاپ می‌کند.
5. شناسه و نوع آن را در جدول شناسه‌ها ثبت می‌کند.

پردازش گره‌های دستور بازگشت

```
else if data == &NonTerminal::ReturnStatement {
        let node = node.first_child().unwrap();
        let typr = if node.data() ==
&SymbolTree::NonTerminal(NonTerminal::Expression) {
            if let SymbolTree::Token(token) =
node.last_child().unwrap().data() {             ids_table.iter().find(|f|
f.0 .0 == token.literal).unwrap().1.0.clone()         }
            else {
               let mut g = None;
               find_the_op(&node, &mut g);
               g.unwrap().clone()
            }
        }
        else { unimplemented!(); };
        fn find_typr(node: NodeRef<SymbolTree>) -> SymbolTree {
           if node.data() ==
&SymbolTree::NonTerminal(NonTerminal::Declaration) {
node.last_child().unwrap().first_child().unwrap().data().clone()
}
           else { find_typr(node.parent().unwrap()) }
        }
        if let SymbolTree::Token(t) = find_typr(node) {
           if typr.clone() != t.token {
```

```
                println!("return type doesnt match {:?}", t);
            }
        }
}
```

این بخش گره‌های دستور بازگشت را پردازش می‌کند:

1. نوع عبارت بازگشتی را پیدا می‌کند.
2. بررسی می‌کند که آیا نوع بازگشتی با نوع اعلان تابع مطابقت دارد یا خیر.

بررسی شناسه‌ها

```
if let SymbolTree::Token(token) = symbole {
   if token.token == TokenType::T_Id &&
      ids_table.keys().find(|f| f.0 == token.literal).is_none() &&
node.parent().unwrap().parent().unwrap().data() ==
&SymbolTree::NonTerminal(NonTerminal::VarOrFunc) {
         println!("var or func not declaration {:?}", { token })
    }
}
```

این بخش بررسی می‌کند که آیا شناسه‌ها (متغیرها یا توابع) قبل از استفاده اعلان شده‌اند یا خیر.