NewBie Interpreter
Đồ án môn: Kiến trúc phần mềm

Phạm Lưu Trọng Nghĩa – 1612418

Phan Hải Bình – 1612041

Giáo viên: Trần Minh Triết

Trợ giảng: Nguyễn Đức Huy, Nguyễn Phạm Phương Nam, Trần Văn Quý

Contents

I. Introduction:

Gần đây có khá nhiều ngôn ngữ lập trình mới được sáng tạo (GOLANG – 2009, TypeScript – 2012) từ những ngôn ngữ lập trình cơ bản như: Javscript, C/C++. Để máy có thể hiểu được ngôn ngữ lập trình, chúng ta cần một "người đứng giữa" để dịch. Có 2 loại dịch:

- Biên dịch (Compile): Chuyển đổi từ ngôn ngữ cấp cao thành ngôn ngữ cấp thấp (thường là mã máy) qua trình biên dịch (compiler). Sử dụng biên dịch ta sẽ thấy thường có bước build. Các ngôn ngữ biên dịch hiện nay như: C++, C#, Golang, ngôn ngữ biên dịch thường là loại ngôn ngữ static type, thường có runtime speed nhanh hơn so với ngôn ngữ thông dịch.
- Thông dịch (Interpret): Không cần qua bước build, thông dịch là sử dụng ngôn ngữ cấp thấp để đọc từng dòng và chạy ngôn ngữ cấp cao, có thể xem như ngôn ngữ cấp thấp đó là trình thông dịch. Các ngôn ngữ thông dịch hiện nay như: Javascript, PHP, Python, là các loại ngôn ngữ dynamic type, thường có runtime speed chậm hơn so với ngôn ngữ biên dịch.

Để hiểu được cách hoạt động của compiler/interpreter, nhóm đã research và implement một ngôn ngữ mới cùng với các subset syntax cơ bản sẽ được giới thiệu ở phần tiếp theo. Ngôn ngữ mới của nhóm sẽ là ngôn ngữ thông dịch, có tên là **NewBie** sử dụng C# làm interpreter để thông dịch **NewBie**.

Vì sao nhóm chọn C# làm interpreter?

- Thứ nhất, C# là ngôn ngữ biên dịch, nên runtime speed nhanh hơn các ngôn ngữ thông dịch khác.
- Thứ hai, vì nhóm chỉ giành thời gian một tuần để thực hiện đồ án, C# là ngôn ngữ static type, nên việc debug/fixbug sẽ nhanh hơn trong quá trình development.
- Thứ ba, dẫu biết sử dụng C/C++ sẽ nhanh hơn, nhưng mà quá trình development sẽ mất thời gian hơn vì C/C++ là ngôn ngữ có độ phức tạp cao.

II. Language Rules:

1. Operators:

Phép gán:

Operator	Description	Example
= Assign	Gán giá trị cho một biến	a = b

Biểu diễn được các phép toán số học cơ bản, nếu a = 5, b = 10 ta có:

Operator	Description	Example
+ Addition	Phép cộng giữa 2 số	a + b = 15
– Subtraction	Phép trừ giữa 2 số	a – b = -5
* Multiplication	Phép nhân giữa 2 số	a * b = 50

/ Division	Phép chia thực giữa 2 số	a / b = 0.5
// Floor Division	Phép chia lấy phần nguyên giữa 2 số	b // a = 2
% Mod Division	Phép chia lấy dư giữa 2 số	b % a = 0

Biểu diễn được các phép so sánh cơ bản, nếu a = 5, b = 10 ta có:

Operator	Description	Example
== Equal	So sánh bằng giá trị của 2 số	a == b = false
!= Not Equal	So sánh khác giá trị của 2 số	a != b = true
> Greater than	So sánh lớn hơn giá trị của 2 số	a > b = false
< Less than	So sánh bé hơn giá trị của 2 số	a < b = true
>= Greater than or equal	So sánh lớn hơn bằng giá trị của 2 số	a >= b = false
<= Less than or equal	So sánh bé hơn bằng giá trị của 2 số	a <= b = true

2. Datatype:

Datatype	Keyword
Số nguyên	int
Số Thực	real
Chuỗi	string
Boolean	bool

3. Variable:

Khai báo biến:

```
var x : int;
var y, z : real;
var firstName, lastName: string;
var isOdd, isPowerOfTwo: bool;
```

Sử dụng biến trong expression

```
z = (x + y)*(x + y) - 2*x*y ;
```

4. Function:

Khai báo hàm, gọi và sử dụng hàm:

```
function IsOdd(x: int) {
    return x % 2 != 0;
}

odd = IsOdd(3);
```

Trình thông dịch NewBie cung cấp một Builtin function:

```
print(expr: expression);
```

5. Condition:

Sử dụng câu lệnh điều kiện, rẽ nhánh.

```
if(expression) {
    # statements
}
else {
    # statements
}
```

6. Loop:

Sử dụng câu lệnh vòng lặp. (Ex: chương trình print từ 0 → 9)

```
var i: int;
i = 0;
while(i < 10) {
    i = i + 1;
    print(i);
}
```

7. Scope:

Các variable được khai báo trong scope thì chỉ được sử dụng trong scope.

Các variable được khai báo ở scope cha thì scope con sẽ được sử dụng.

```
program TenChuongTrinh
{
    var h: bool;
    var s : string;
    var x, y: int;

    s = "This is an odd";
    x = 5;

    function PrintIfOdd(x: int) {
        var k: string;
        k = 5;

        if(x % 2 != 0) {
            print(s);
            print(x);
        }
    }

    PrintIfOdd(x);
    print(k); # error

}
```

8. Reserved keywords:

Reserved keywords sẽ là các keyword mà người dùng không được đặt tên biến trùng. Sau đây là các reserved keyword trong NewBie.

Keyword	Ý nghĩa
var	Dùng để khai báo biến
program	Dùng để khai báo chương trình
int	Dùng để khai báo biến kiểu số nguyên
real	Dùng để khai báo biến kiểu số thực
string	Dùng để khai báo biến kiểu chuỗi
bool	Dùng để khai báo biến boolean
function	Dùng để khai báo function

return	Dùng để trả về giá trị trong một function
if	Câu lệnh dùng trong điều kiện rẽ nhánh đúng
else	Câu lệnh dùng trọng điều kiện rẽ nhánh sai
while	Câu lệnh dùng trong khi muốn thực hiện điều kiện lặp
true	Boolean constant true
false	Boolean constant false

III. Structure:

1. Modules/Processing:

NewBie sẽ thông dịch source code như sau:

- B1: Đầu vào INPUT là một đoạn text, là source code của ngôn ngữ NewBie.
- B2: Phân tích text sang các lexemes (hay còn gọi là token) thông qua **Lexer**.
- B3: Xây dựng **Abstract Syntax Tree** (**AST**), cùng với việc phân tích lỗi Syntax từ các tokens thông qua **Parser**.
- B4: Xây dựng **Symbol Table** từ **AST** thông qua **Semantic Analyzer**.
- B5: **Semantic Analyzer** dựa vào **Symbol Table** để phân tích các lỗi semantic trong source code.
- B6: **Interpreter** sẽ thực hiện duyệt cây **AST**, và execute các câu lệnh.

2. Lexer:

a. Công dụng:

Công dụng chính của Lexer là chuyển hóa source code thành các token, ví dụ các token như: *int*, *real*, *program*, *function*.

Lexer là module có INPUT là raw text, OUTPUT sẽ là một stream các token và lần lượt được feed vào Parser.

b. Property:

```
private const char ENDCHAR = '@';       // Kí tự kết thúc
private string text;     // Chuỗi cần phân tích thành token
private int pos;         // Vị trí hiện tại đang phân tích
public char currentChar; // Kí tự hiện tại
private int lineno;      // Dòng hiện tại trong chương trình người dùng nhập
private int column;      // Cột hiện tại trong chương trình người dùng nhập
private static Dictionary<string, Token> RESERVED_KEYWORDS // Dictionary chứa các
reserved token
```

Công dụng của các property:

- ENDCHAR: Sử dụng để kiểm tra đã đến cuối file chưa.
- pos, currentChar: Được xem như là con trỏ của Lexer.
- lineno/ column: dòng/cột hiện tại mà con trỏ của Lexer đang đứng, dùng để in lỗi cụ thể cho người sử dụng biết lỗi ở đâu.

- RESERVED_KEYWORDS: Dùng để chứa các Reserved keyword của trình thông dịch.

c. Các hàm chính:

Hàm khởi tạo list reserved keywords của chương trình. Với key là tên reserved keywords dưới dạng string, value là dữ liệu Token. Hàm sẽ được gọi khi trình thông dịch được khởi tạo.

```
/// <summary>
/// Hàm tĩnh khởi tạo các RESERVED_KEYWORDS của chương trình
/// </summary>
public static void InitReservedKeywords()
{
     RESERVED_KEYWORDS.Add("var", new Token(TokenType.VAR, "VAR"));
     RESERVED_KEYWORDS.Add("program", new Token(TokenType.PROGRAM, "PROGRAM"));
     RESERVED_KEYWORDS.Add("int", new Token(TokenType.INTEGER, "INTEGER"));
     RESERVED_KEYWORDS.Add("real", new Token(TokenType.REAL, "REAL"));
     RESERVED_KEYWORDS.Add("string", new Token(TokenType.STRING, "STRING"));
     RESERVED_KEYWORDS.Add("bool", new Token(TokenType.BOOL, "BOOL"));
     RESERVED_KEYWORDS.Add("function", new Token(TokenType.PROCEDURE, "PROCEDURE"));
     RESERVED_KEYWORDS.Add("return", new Token(TokenType.RETURN, "RETURN"));
     RESERVED_KEYWORDS.Add("if", new Token(TokenType.IF, "IF"));
     RESERVED_KEYWORDS.Add("else", new Token(TokenType.ELSE, "ELSE"));
     RESERVED_KEYWORDS.Add("while", new Token(TokenType.WHILE, "WHILE"));
     RESERVED_KEYWORDS.Add("true", new Token(TokenType.BOOL_CONST, "TRUE"));
     RESERVED_KEYWORDS.Add("false", new Token(TokenType.BOOL_CONST, "FALSE"));
}
```

Hàm lấy token tiếp theo, hàm sẽ iterate qua các character và nhận diện token, sau đó trả về giá trị Token tiếp theo. Hàm sẽ được lớp Parser sử dụng để lấy Token tiếp theo. Các Token được lấy sẽ là các loại như: operators (+, –, *, /, …), số nguyên (12, 2323, 12345), số thực (1.23, 31.4), các identifiers (tất các reserved keywords, tên biến, tên hàm), và các kí tự khác.

```
/// <summary>
/// Lấy token tiếp theo
/// </summary>
/// <returns></returns>
public Token GetNextToken()
{   //Khi chưa hết file
    while (!this.IsEnd())
    {
        //Nếu là khoảng trắng thì bỏ qua
        if (Char.IsWhiteSpace(this.currentChar))
        {
            this.SkipWhiteSpace();
            continue;
        }
        //Nếu là comment thì bỏ qua
        else if (this.currentChar == '#')
        {
           this.AdvanceCurrentChar();
           this.SkipComment();
           continue;
        }
        //Nếu gặp kí tự “ thì trả về một chuỗi
        else if(this.currentChar == '"')
```

```
        {
            return this.GetStringToken();
        }
        // MORE IF ELSE ...
    }
}
```

3. Parser:

a. Công dụng:

Parser sẽ nhận từng Token từ **Lexer** thông qua hàm *GetNextToken()*. Sau đó Parser sẽ dựa trên Token.type đó để build Abstract Syntax Tree.

Trong quá trình build Abstract Syntax Tree parser đồng thời kiểm tra lỗi syntax và in lỗi ra nếu có lỗi. Lớp *Error* sẽ chịu trách nhiệm việc xử lý và in ra error.

Những thành phần trong AST như: Block, Decalration, Compound, Statement, Expression sẽ là những thành phần đóng vai trò quan trọng tạo nên NewBie.

b. Block:

Một Block sẽ bắt đầu từ dấu "{" đến dấu "}"

Một Block sẽ có thể không có Declaration hoặc có thể có nhiều Declaration, Declaration luôn nằm đầu tiên ở mỗi Block.

Cũng nằm trong Block, và nằm sau Declaration sẽ là Compound.

c. Declaration:

Declaration có 2 loại:

- Variable Declaration: Gồm thông tin là Variable, và Type của Variable đó.
- Function Declaration: Gồm thông tin là Function Name, Parameters, Block.

d. Compound:

Compound là tập hợp các Statement, một Compound luôn nằm trong một Block và luôn nằm sau các Declarations.

e. Statement:

Dựa vào Token hiện tại, Parser sẽ đưa một Statement về những loại sau:

- Compound
- Function Call
- Assignment Statement
- Return Statement
- If Statement
- While Statement
- Empty (Các trường hợp còn lại)

f. Expression:

Một Expression sẽ là tập hợp những phép tính toán học, phép tính logic.

Nhóm đã tách một Expression thành nhiều Term. Trong Term sẽ có các Factor.

Tóm lại Expression, Term, Factor sẽ được định nghĩa như sau:

- Expression: Tập hợp các phép tính cộng và trừ của nhiều Term.
- Term: Tập hợp các phép tính nhân và chia của nhiều Factor.
- Factor: Có thể là một trong những loại Token sau đây:
    - Operators: +, -
    - Số nguyên
    - Số thực
    - Chuỗi
    - Expression nằm trong 2 dấu "(" và ")"
    - Function call
    - Variable

4. Cây cú pháp - Abstract Syntax Tree(AST):

Cây AST sẽ chịu trách nhiệm chứa các Node ở trên của Parser parse ra như: Block, Declaration, Compound, Statement, …

Các class như Compound, Block, Declaration, … có thể sẽ chứa các cây con AST khác, và các class này sẽ cùng kế thừa cây AST. Cây AST có cấu trúc tương tự như mẫu Composite.

Để duyệt trên cây AST ta sẽ dùng mẫu Visitor, để khi Visit một node, ta sẽ lần lượt Visit tất cả các node con của node đó. Class NodeVisitor sẽ là class trung tâm, và method chính là Visit(AST node).

Method Visit sẽ cần một tham số truyền vào là một node trên cây AST, sau đó Visit sẽ lấy tên class của Node đó và thêm prefix "Visit" để gọi method. Các method "Visit{Class}" sẽ được viết trong Interpreter hoặc Semantic Analyzer kế thừa từ Node Visitor.

5. Symbol Table:

Symbol Table được xây dựng dựa trên AST. Symbol Table có tác dụng lưu các Symbol và scope của nó.

Một Symbol có thể là một biến, một function, hoặc builtin function (gồm tên và type).

Semantic Analyzer kế thừa từ Node Visitor từ đó duyệt trên cây AST và build ra Symbol table.

Semantic Analyzer sẽ phải viết lại tất cả các hàm "Visitor{Class}" để có thể chạy được.

6. Semantic Analyzer:

Semantic Analyzer nhiệm vụ chính là dùng để phân tích các lỗi semantic trước khi thực hiện interpreter. Các lỗi semantic mà NewBie đã cover được như:

- Chỉ sử dụng biến đã declaration.
- Khai báo biến trùng tên.

Trong quá trình duyệt trên AST và build Symbol Table Semantic Analyzer sẽ Visit lần lượt hết tất cả các lỗi semantic trong source code. Và dựa vào ErrorHandle để in lỗi.

Sau khi quét hết lỗi, trong trường hợp không có lỗi nào, NewBie sẽ bắt đầu xây dựng CallStack và Interpreter.

7. Call Stack:

Tương tự như Symbol Table, Call Stack được xây dựng dựa trên AST và bằng Interpreter.

Call Stack được sinh ra để lưu trữ các giá trị của identifier như: Variable, Function, Builtin Function, và cùng với đó là lưu scope của các identifier đó trong runtime.

Trong Call Stack sẽ chứa nhiều Activation Records, mỗi Activation Record sẽ là một scope.

Khi gọi hàm, một Activation Record sẽ được push vào call stack đồng thời những params, variable, function được declare trong hàm đó sẽ được lưu vào activation record đó. Khi kết thúc hàm, các variable, function sẽ được pop ra ngoài khỏi Call Stack.

8. Interpreter:

Sau khi đã phân tích được semantic error. Interpreter sẽ là là class cuối cùng của flow, Interpreter sẽ tương tác với Call Stack và AST thông qua hàm Visit().

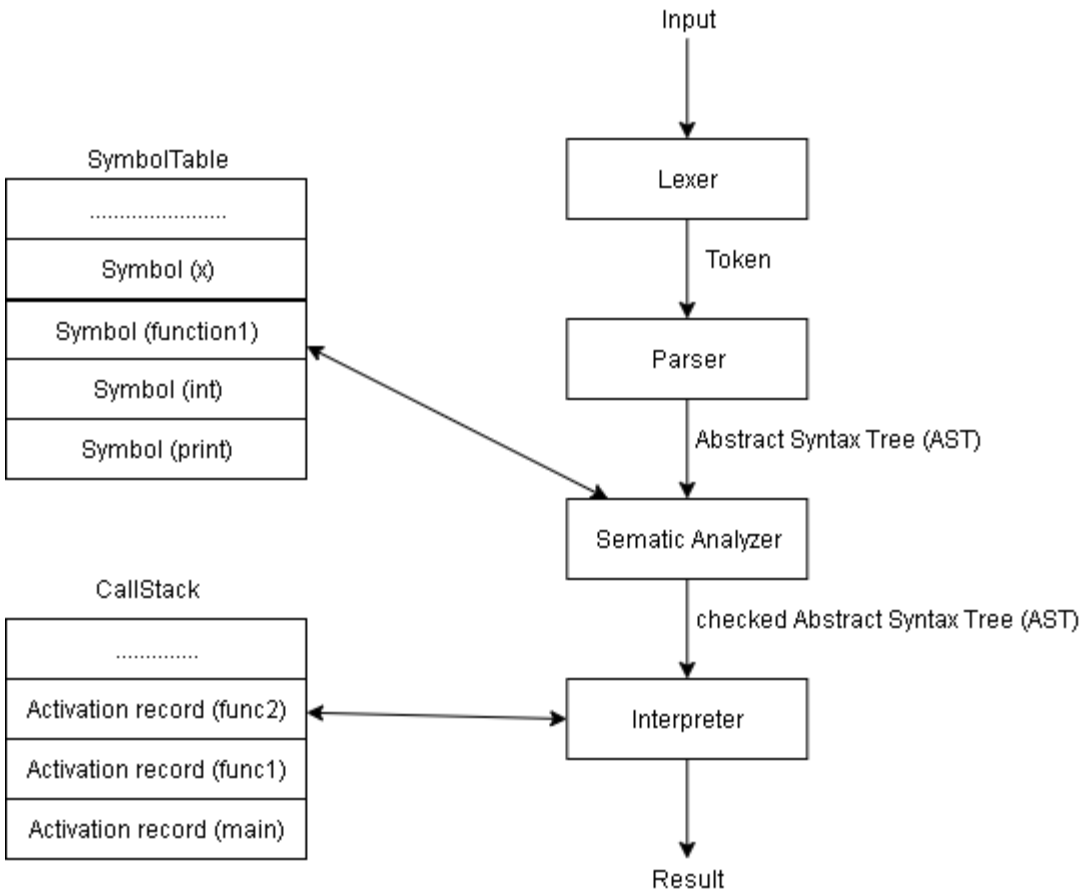
Interpreter kế thừa từ Node Visitor và implement lại các hàm “Visit{Class}”.

Và trong những hàm “Visit{Class}” đó, NewBie lấy các value của các Variable từ Call Stack và thực sự thực hiện các phép tính, lệnh print, các hàm, và cuối cùng là trả về kết quả.

9. Tổng kết:

Chương trình chính của NewBie sẽ được đọc từ 1 file và interpret source code của file đó.

```
        static void Main(string[] args)
        {
            var file = "";
            if (args.Length > 0) file = args[0];

            Token.AddSampleCompareOperator();
            Lexer.InitReservedKeywords();

            string text = Input.Read(file);

            Lexer lexer = new Lexer(text);
            Parser parser = new Parser(lexer);


            var tree = parser.Parse();
            var semanticAnalyzer = new SemanticAnalyzer();
            semanticAnalyzer.Visit(tree);

            Interpreter interpreter = new Interpreter(tree);
            int result = interpreter.Interpret();
        }
```



Hình 1 Full flow chạy của NewBie

IV. Tài liệu tham khảo

- https://ruslanspivak.com/lsbasi-part1/
- https://en.wikipedia.org/wiki/Visitor_pattern
- https://en.wikipedia.org/wiki/Composite_pattern
- https://www.tutorialspoint.com/compiler_design/compiler_design_overview.htm