Nguyên Lý Ngôn Ngữ Lập Trình

Bài Tập Lớn

ZCODE

nhóm thảo luận CSE
https://www.facebook.com/groups/211867931379013

Tp. Hồ Chí Minh, Tháng 1/2024

Mục lục

1 Kế hoạch làm bài

Phần Làm	Bắt đầu	Thời gian	Kết thúc	Kiến thức	Nộp lại discord anh
BTL 2	29/1	6 ngày	23:59 5/2	AST+OOP+FP	BTL2 + Trắc nghiệm

Một số quy định của nhóm

- Nếu mấy bạn vô trễ thì thời gian làm có thể kéo dài theo ngày với thời gian Bắt đầu sẽ là ngày hôm đó
- Sau *deadline* thì anh không chỉ BTL phần đó nữa 
- Hỏi lý thuyết thì hỏi trong nhóm cho tiện
- Hỏi code thì nhắn riêng anh để bảo mật cho mấy bạn
- Nộp bài thì gửi file code *zcode.g4* đối btl1 và *ASTGeneration.py* đối btl2
- Trắc nghiệm thì hỏi lý thuyết *GK* hoặc *harmony*
- **Phần test case sẽ không cung cấp hết mà giữ lại để các bạn nộp anh check test case nếu sai anh gửi test sai cho mấy bạn fix**
- yêu cầu BTL2 là nộp file *ID_BTL2.py*(ASTGeneration.py) + *ID_BTL1.g4* trên mỗi hàm thì cần comment dòng sytax đã viết vào như ví dụ

```
#* program: (COMMENTS NEWLINE | NEWLINE)* list_declared EOF;
def visitProgram(self, ctx:ZCodeParser.ProgramContext):
    return Program(self.visit(ctx.list_declared()))
```

2 AST *Zcode*

2.1 Kiến thức Python cơ bản

- Lệnh if else trong python

```
# lệnh if else cơ bản
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")

# if else tối giản giống toán tử 3 ngôi ? trong c++
print("A") if a > b else print("B")
```

- Đệ quy mảng. Ví dụ cho mảng array hãy nhân các phần tử trong mảng lênh 2 lần

```
def rec(array):
    if len(array) == 0: return []
    return [array[0] * 2] + rec(array[1:])
```

- Chuyển *string* về *int* dùng lện *int*("123")
- *contructor* trong python và cách khởi tạo đối tượng đó

```
class VarDecl(Decl,Stmt):
    name : Id
    varType : Type = None # None if there is no type
    modifier: str = None # None if there is no modifier
    varInit : Expr = None # None if there is no initial

# khởi tạo đối tượng các phần tử không khởi tạo thì mặt định là None
person1 = VarDecl(Id("VoTien"), BoolType(), None, None)
person1 = VarDecl(Id("VoTien"), BoolType())
person1 = VarDecl(Id("VoTien"), BoolType(), None, NumberLiteral(1))
```

2.2 File AST

1. Loại class *Type*
   - *NumberType*, *BoolType*, *StringType* các loại type cơ bản không có tham số
   - *ArrayType* thì có 2 tham số một là *size* danh sách *numberLit* để khởi tạo mảng và *eleType* kiểu dữ liệu của mảng đó
2. Loại class *Expressions*
   - *BinaryOp*, *UnaryOp* là toán tử 2 ngôi và một ngôi nhận vào *op* toán tử dạng chuỗi và *Expressions*
   - *Id* tên của biến nhận vào chuỗi
   - *ArrayCell* là toán tử index hay trong expr7 hiện thực nhận vào *arr* là một biểu thức đệ quy và *idx* là dánh sách $Index_oper$
   - *NumberLiteral*, *StringLiteral*, *BooleanLiteral* thì nhận vào các giá trị tương ứng float, string, bool
   - *ArrayLiteral* khởi tạo array bên trong *value* là danh sách các *expr*
3. Loại class *Statements*
   - *Assign* nhận vào vế trái và vế phải tưng ứng *LHS* và *expr*
   - *Block* nhận vào danh sách các *Stmt*
   - *For* đầu tiên là biến chạy *init* tiếp theo là 2 biểu thức điều kiện và cập nhật cuối cùng biểu thức
   - *Break*, *Continue* không nhận tham số đầu vào
   - *Return* có thể nhận một expr hoặc None
   - *Call* nhận vào 2 tham số đầu là tên và sau đó là danh sách các tham số truyền vào hàm *args*
4. Loại class *Declarations*
   - *VarDecl* nhận vào tên, *type* chỉ có nếu là khai báo các *keyword* nguyên thủy còn *Implicit* thì cho là *None*, *modifier* cứ cho là *None* đi vẫn chưa hiểu ý thầy lắm, *varInit* giá trị *expr* của khai báo khi khởi tạo
   - *FuncDecl* nhận vào 3 tham số là tên, danh sách *param* nếu có cuối cùng là biểu thức nếu có
5. class *Program* dùng khởi tạo ban đầu cho trương trình

2.3 Bài tập lớn

Các bước Thực hiện trước khi code

Bước 1: Tải BTL2 của thầy về

Bước 2: Copy *AST.py* vào các *folder astgen* và *test* cho dễ code

Bước 3: Chạy lệnh **python run.py gen** thì 2 file *ZCodeParser* và *ZCodeVisitor* nằm trong folder *target* có thể chạy lệnh **python run.py test LexerSuite** thì 2 file *ZCodeParser* và *ZCodeVisitor* nằm trong folder *parse*

Bước 4: Tìm file *ZCodeVisitor.py* copy tất cả các hàm trong *class ZCodeVisitor* sang cho *ASTGeneration* đang viết, chỉ các hàm thôi nha

Bước 5: Copy 2 file *ZCodeParser* và *ZCodeVisitor* vào folder *astgen* ngang hàng với file *AST* và *ASTGeneration*

Bước 6: Mở file *AST.py* xem cấu trúc cây của từng loại

Bước 7: Code trong file *ASTGeneration.py* với các hàm vừa copy

Các loại hàm hay dùng trong antlr

- *ctx.* < *Name* *Parse* > () Kiểm tra xem Tree có con tên đó hay không.

  ```
  list_declared: declared list_declared | declared;
  ```

  VD: Kiểm tra xem cây này có con tên đó hay không *ctx.declared()* tồn tại nên không phải *None* -> bool(*ctx.declared()*) = *true*, còn *ctx.ID()* không tồn tại nên là *None* -> bool(*ctx.ID()*) = *false*

- *ctx.getChildCount()* Kiểm tra xem Tree này có mấy con.

  ```
  list_declared: declared list_declared | declared;
  ```

  VD: cây này có 2 con *declared list_declare* hoặc là 1 con *declared* nên *ctx.getChildCount()* = 2, *ctx.getChildCount()* = 1

- *ctx.* < *Name* *Parse* > ().*getText*() Lấy ra chuỗi *Tokens* chỉ có các *Tokens* trong *lexer* mới gọi hàm này như *ID*, *NUMBER_LIT*, *STRING_LIT*, KeyWord, Operators, Separators

- *self.visit* gọi đệ quy xuống hàm của con trong *parse* này chạy theo cơ chế nó tìm ra hàm phù hợp với tên và prama để gọi đến.

-

Các loại hay dùng

1. Biểu thức dạng trả về danh sách

   ```
   list_declared: declared  list_declared | declared;
   ```

   Ta muốn lấy ra một mảng list_declared dùng đệ quy

   ```
   def visitList_declared(self, ctx:MT22Parser.List_declaredContext):
       if ctx.list_declared():
           return [self.visit(ctx.declared())] + self.visit(ctx.list_declared())
       return [self.visit(ctx.declared())]
   ```

2. Biểu thức dạng trả về một đối tượng

   ```
   implicit_var: VAR ID ASSIGNINIT expression;
   ```

   Xem danh sách các tham số trong file *AST* của class *ImplicitVarDecl* trong hàm *init* để truyền vào

   ```
   def visitImplicit_var(self, ctx:ZCodeParser.Implicit_varContext):
       return VarDecl(Id(ctx.ID().getText()), None, None, self.visit(ctx.expression()))
   ```

3. Biểu thức dạng trả về nhiều kiểu

```
type_prime: BOOL | NUMBER | STRING;
```

Thực hiện if else để kiểm tra thuộc loại nào rồi trả về *Type* của loại đó

```
def visitType_prime(self, ctx:ZCodeParser.Type_primeContext):
    if ctx.BOOL():
        return BoolType()
    elif ctx.NUMBER():
        return NumberType()
    return StringType()
```

4. Biểu thức dạng expr

```
expression: expression1 CONCAT expression1  | expression1;
expression2: expression2 (AND | OR) expression3 | expression3
```

nếu trong một biểu thức có nhiều *expression* chung tên thì sẽ là một mảng nên cần lấy từng phần tử, nếu một trong hai bên của | có nhiều phân tử thì bên còn lại tuy có một vẫn tử nhưng vẫn mặc định là mảng

```
def visitExpression(self, ctx:ZCodeParser.ExpressionContext):
    if ctx.getChildCount() == 1:
        return self.visit(ctx.expression1()[0])

    op = ctx.CONCAT().getText()
    left = self.visit(ctx.expression1()[0])
    right = self.visit(ctx.expression1()[1])
    return BinaryOp(op, left, right)

def visitExpression2(self, ctx:ZCodeParser.Expression2Context):
    if ctx.getChildCount() == 1:
        return self.visit(ctx.expression3())

    op = ''
    if ctx.AND():
        op = ctx.AND().getText()
    elif ctx.OR():
        op = ctx.OR().getText()

    left = self.visit(ctx.expression2())
    right = self.visit(ctx.expression3())
    return BinaryOp(op, left, right)
```

5. Phần xử lí hơi rắc rối hơn

```
#* expression7: (ID | ID LPAREN index_operators? RPAREN)
#*   LBRACKET index_operators  RBRACKET  | expression8;
    if ctx.getChildCount() == 1:
        return self.visit(ctx.expression8())
    elif ctx.getChildCount() == 4:
        return ArrayCell(Id(ctx.ID().getText()), self.visit(ctx.index_operators()[0]))
    elif len(ctx.index_operators()) == 2:
        return ArrayCell(CallExpr(Id(ctx.ID().getText()),
        self.visit(ctx.index_operators()[0])), self.visit(ctx.index_operators()[1]))
    return ArrayCell(CallExpr(Id(ctx.ID().getText()), []),
    self.visit(ctx.index_operators()[0]))
```

6. dùng *tuple* trong python xử lí với return a, b thì trả về một tuple (a,b) ta có thể gán c, b = (a,b) cũng được hoặc có thể phân rã nó ra dùng toán tử *

```
    #* callStmt: func -> statement
    def visitCallStmt(self, ctx:ZCodeParser.CallStmtContext):
        return CallStmt(*self.visit(ctx.func()))

    #* funcCall: func -> expression
    def visitFuncCall(self, ctx:ZCodeParser.FuncCallContext):
        return FuncCall(*self.visit(ctx.func()))

    #* func: ID LPAREN index_operators? RPAREN
    def visitFunc(self, ctx:ZCodeParser.FuncContext):
        if ctx.index_operators():
            return Id(ctx.ID().getText()), self.visit(ctx.index_operators())
        return Id(ctx.ID().getText()), []
```

7. xử lí ở biểu thức *IF*

```
#* if_statement : IF expression statement elif_list (ELSE statement)?;
def visitIf_statement(self, ctx:ZCodeParser.If_statementContext):
    if not ctx.ELSE():
        return If(self.visit(ctx.expression()),
                        self.visit(ctx.statement()[0]),
                        self.visit(ctx.elif_list()),None)
    return If(self.visit(ctx.expression()),
                self.visit(ctx.statement()[0]),
                self.visit(ctx.elif_list()),self.visit(ctx.statement()[1]))

#* elif_list: ELIF expression ignore? statement elif_list | ;
def visitElif_list(self, ctx:ZCodeParser.Elif_listContext):
    if ctx.getChildCount() == 0: return []
    return [(self.visit(ctx.expression()), self.visit(ctx.statement()))]
            + self.visit(ctx.elif_list())
```

8. Phần *visitIgnore* thì bỏ qua hoặc return none cũng được không ảnh hưởng

3 Cập nhật ngày 17/2

1. ép kiểu từ string về float cho $NumberLiteral(float(ctx.NUMBER\_LIT().getText()))$
2. $modifier$ trong lớp $VarDecl$ là $modifier = "var"$ đối với kiểu *var* và $modifier = "dynamic"$ đối với kiểu *dynamic*
3. Dã thêm hết tất cả các test case nên các bạn làm nộp bkel luôn nha không cần nộp lại anh vậy là chúng ta hoành thành 2 btl rồi.
4. khi nào mở BTL3,4 thì anh sẽ báo sau do anh đang code chắc cuối tháng này -> tìm hiểu phần scope và type

4 Các Khóa Học HK232

nhóm thảo luận CSE

https://www.facebook.com/groups/211867931379013

- Lớp BTL1 + GK + LAB + Lý thuyết + Harmony của môn DSA HK232
- Lớp BTL2 + CK + LAB + Lý thuyết + Harmony của môn DSA HK232
- Lớp BTL1 + LAB + Lý thuyết + Harmony của môn KTLT HK232
- Lớp BTL2 + LAB + Lý thuyết + Harmony của môn KTLT HK232
- Lớp BTL1 + BTL2 + GK + Harmony của môn PPL HK232
- Lớp BTL3 + BTL4 + CK + Harmony của môn PPL HK232

CHÚC CÁC EM HỌC TỐT