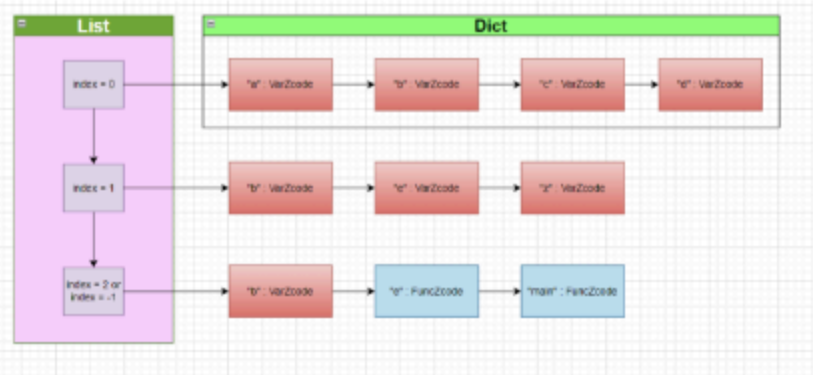
1 Lý thuyết BTL3

1.1 Thiết kế param

Ta có 2 phần là toàn cục và cục bộ :

- Toàn cục sẽ bao hôm các hàm và các biến khai báo ngoài hàm
- Cục bộ thì sẽ vào các biến được khai báo trong các block của hàm, for, if, block và param của hàm




```
# Tạo danh sách
my_list = ['Var:Zcode', 'Func:Zcode', 'Var:Zcode']

# Tạo từ điển
my_dict = {
    'a': 'Var:Zcode',
    'b': 'Var:Zcode'
}

# cấu trúc dữ liệu param được thiết kế
param = [
    {'a': 'Var:Zcode', 'b': 'Var:Zcode'},
    {'a': 'Var:Zcode', 'c': 'Var:Zcode'},
    {'a': 'Var:Zcode', 'c': 'Func:Zcode'}
]
```

- Để dễ tìm kiếm ta thiết kế Dict đối với một tầm vực (block hay toàn cục)
- List lưu trữ từng tầm vực của biến được lưu trữ với index 0 đến index cuối thì tầm vực sẽ từ cục bộ ra tới toàn cục
- tại index cuối thì sẽ là toàn cục gồm khai báo biến toàn cục và hàm
- các index còn lại sẽ là các biến cục bộ
- Khi tìm kiếm thì ta tìm kiếm từ cục bộ đến toàn cục nghĩa là từ index 0 đến index cuối

1.2 Class Zcode

```
class FuncZcode(Zcode):
    def __init__(self, param = [], typ = None, body = False):
        self.param = param
        self.typ = typ
        self.body = body

class VarZcode(Zcode):
    def __init__(self, typ = None):
        self.typ = typ
```

- **param** là danh sách các *type* của từng param được truyền vào
- **typ của hàm** là kiểu dữ liệu trả về nếu chưa xác định thì để *None*
- **body** kiểm tra xem declaration-only part hay không nếu có body thì true nếu không thì là false
- **typ của biến** là kiểu dữ của biến nếu chưa xác định thì để *None*

```
func VoTien(number a, string b)
    return 1.1
-> FuncZcode([NumberType(), StringType()], NumberType(), True)

func VoTien()
-> FuncZcode([], None, True)

number VoTien
-> VarZcode(NumberType())

dynamic VoTien
-> VarZcode(None)
```

1.3 Cách dùng Prama List[Dict]

https://www.scaler.com/topics/list-of-dictionaries-in-python/ https://www.geeksforgeeks.org/type-isinstance-python/

1. Tìm kiếm tên có ở hàng đầu tiên cục bộ gần nhất hay không

```
// name có tồn tại trong dict đầu tiên trong list
if param[0].get(name):
    return True
else:
    return Flase
```

2. Tìm kiếm tên có ở tầm vực gần nhất

```
for item in param:
    Id = item.get(name)
    if Id is not Node:
        return True
return false
```

3. sử dụng type để kiểm tra đang là hàm hay biến

```
type(item) is VarZcode # kiểm tra có phải là khai báo biến không

type(item) is FuncZcode # kiểm tra có phải là khai báo hàm không
```

2 khởi tạo

2.1 init

```
self.ast = ast
self.BlockFor = 0
self.function = None
self.io = [{
        "readNumber" : FuncZcode([], NumberType(), True),
        "readBool" : FuncZcode([], BoolType(), True),
        "readString" : FuncZcode([], StringType(), True),
        "writeNumber" : FuncZcode([NumberType()], VoidType(), True),
        "writeBool" : FuncZcode([BoolType()], VoidType(), True),
        "writeString" : FuncZcode([StringType()], VoidType(), True)
        }]
```

- **self.ast** cây đã được xây ở BTL2
- **self.BlockFor** Hiện tại có ở trong vòng for hay không nếu ở thì đang ở vòng for thứ mấy
- **self.function** hàm hiện tại đang xử lí dùng biến FuncZcode
- **self.io** này dùng thư viện IO

Hàm để kiểm tra thử thành thông hay không với gọi hàm self.visit(self.ast, self.io) với ast là cây ở BTL2 và io là param đầu tiên mặc định của BTL này

```
    def check(self):
        self.visit(self.ast, self.io)
        return None
```

2.2 Các hàm phụ trong việc so sánh Type

```
def comparType(self, LHS, RHS):
    pass

def comparListType(self, LHS, RHS):
    pass
```

- **comparType(self, LHS, RHS)** truyền vào 2 *Type* sẽ có thể là **NumberType, BoolType, StringType, VoidType, ArrayType** chú ý ArrayType
- **comparListType(self, LHS, RHS):** truyền vào 2 danh sách có kích thước khác nhau

3 Các lỗi cần nén

- **2.1 Redeclared Variable/ Parameter/ Function** The declaration must be unique in its scope which is formally described as in ZCode specification. Otherwise, the exception Redeclared(<kind>, <identifier>) is released, where <kind> is the kind (Variable/Parameter/Function) of the identifier in the second declaration

  1. **Ý tưởng :** nếu khai báo tên trùng nhau trong một scope thì sẽ bị lỗi do trùng trên sẽ có 3 loại là **Variable/ Parameter/ Function** tên hàm và biến hay parameter không được trùm nhau trong một tầm vực
  2. **Gợi ý :** ta cần xét trong tầm vực gần nhất là **prama[0]** kiểm tra xem có nào trùng tên hay không, đối với **func** thì cần kiểm tra có khai báo một phần trước hay không
  3. **Kiến thức : Python Dictionaries**
  4. **Ví dụ**

```
# Ví Dụ 1
number a
string a -> Redeclared Variable

# Ví Dụ 2
func a() return
string a -> Redeclared Variable

# Ví Dụ 3
func a() return
func a() -> Redeclared Function

# Ví Dụ 4
number a
func a() -> Redeclared Function

# Ví Dụ 5
func a(number b, number b) -> Redeclared Parameter
```

- **2.2 Undeclared Identifier/Function**
  - The exception Undeclared(Identifier(), <identifier-name>) is released when there is an identifier is used but its declaration cannot be found. The identifier can be a variable or a parameter.
  - Undeclared(Function(), <function-name>) is released if there do not exist any function with that name. The function usage (as the function call) could not be allowed before its declaration.

  1. **ý tưởng :** nếu sử dụng mà **Identifier/Function** không được khai báo trước thì không được dùng sẽ gây ra lỗi
  2. **Gợi ý :** ta cần xét hết **prama** để kiểm tra **name** có tồn tại hay không, sau đó xét tới **type** đang là **FuncZcode** hay **FuncZcode**
  3. **Kiến thức : Python Dictionaries**
  4. **Ví dụ**

```
# Ví Dụ 1
number a <- b -> Undeclared Identifier

# Ví Dụ 2
func a() return 1
```

```
number b <- b() -> Undeclared Function

# Ví Dụ 3
func a() return 1
func b() begin
    number a
    var c <- a() -> Undeclared Function
end
```

- **2.6 No definition for a function** In ZCode, a function could be introduced as a declaration without the body (as the definition). If exists a function without the definition in the program, the exception NoDefinition(<name>), <name> is the name of function with the first appearance is not defined.
  1. **ý tưởng :** nếu khai báo một hàm mà chưa được triển khai gọi là khai bao một phần thì lúc sau mà không triển khai phần còn lại thì sẽ lỗi này
  2. **Gợi ý :** Kiểm tra trong **param[-1]** có tồn tại **FuncZcode** nào chưa khai báo **Body** hay không
  3. **Kiến thức : Python Dictionaries**
  4. **Ví dụ**

```
# Ví Dụ 1
func foo()
-> NoDefinition

# Ví Dụ 2
func foo()
func foo() return -> true
```

- **2.7 Break/Continue not in loop** A break/continue statement must be inside directly or indirectly a loop otherwise the exception MustInLoop(<statement>) must be thrown.
  ý tưởng : Break/Continue chỉ được phép xử dụng trong vòng **For**
- **2.8 No entry point** There must be a function whose name is main without any parameter and return nothing in a ZCode program. Otherwise, the exception NoEntryPoint() is released.
  ý tưởng : Hàm main không được khai báo đúng hay không khai báo, các tham số có đúng hay không
- **2.7 Break/Continue not in loop** A break/continue statement must be inside directly or indirectly a loop otherwise the exception MustInLoop(<statement>) must be thrown.
  ý tưởng : Break/Continue chỉ được phép xử dụng trong vòng **For**
- **2.3 Type Mismatch In Expression** An expression must conform the type rules for expressions, otherwise the exception TypeMismatchInExpression(<expression>) is released. The type rules for expression are as follows:
  - For an array subscripting (index operator) E1[E2], E1 must be in array type and E2 must be a list of number.
    1. **ý tưởng :** Kiểm tra array có phải đúng kiểu hay không và danh sách bên trong phải là NumberType hay không
    2. **Gợi ý :** Visit cả E1, và danh sách E2 ta sẽ thu được **type** và kiểm tra có phải đúng kiểu hay không
    3. **Kiến thức : isinstance**
    4. **Ví dụ**

```
# Ví Dụ 1
number a
var b <- a[1] -> TypeMismatchInExpression

# Ví Dụ 2
number a[2]
var b <- a["1"] -> TypeMismatchInExpression

# Ví Dụ 3
dynamic a
var b <- a[1] -> TypeMismatchInExpression không thể suy diễn kiểu được
```

- For a binary and unary expression, the type rules are described in the ZCode specification.
  1. **ý tưởng :** xem kiểu cho phép trong bảng mô tả BTL
  2. **Gợi ý :** visit các phần tử rồi kiểm tra type của từng phần tử
  3. **Kiến thức :** **isinstance**
  4. **Ví dụ**

```
# Ví Dụ 1
var a <- 1 + "1" -> TypeMismatchInExpression

# Ví Dụ 2
var a < - "1" -> TypeMismatchInExpression
```

- For a function call <function-name>(<args>), the callee <method name> must have non-void as return type (The VoidType is a class representing no return anything in a function). The type rules for arguments and parameters are the same as those mentioned in a procedure call.
  1. **ý tưởng :** Kiểm tra danh sách arguments có đủ phần tử không, các kiểu có tương thích hay không, kiểu trả về có phải là non-void không
  2. **Gợi ý :** Tìm kiếm *FuncZcode* trong **prama** rồi kiểm tra **2 list prama** và kiểm tra *typeReturn*
  3. **Kiến thức :** **isinstance**
  4. **Ví dụ**

```
# Ví Dụ 1
func foo() return 1
var a <- foo(1) -> TypeMismatchInExpression

# Ví Dụ 2
func foo(number a) return 1
var a <- foo("1") -> TypeMismatchInExpression

# Ví Dụ 3
func foo(number a) return 1
var a <- foo(1) -> TypeMismatchInExpression
```

- **2.5 Type Mismatch In Statement** A statement must conform the corresponding type rules for statements, otherwise the exception TypeMismatchInStatement(<statement>) is released. The type rules for statements are as follows:

- The type of a conditional expression in an if or in a for statement must be boolean.
  1. **ý tưởng :** Kiểm tra trong **if và for**
  2. **Gợi ý :** visit từng thằng thôi if bao gồm **expr, các expr trong elif**, for bao gồm **name, condExpr, updExpr**
  3. **Kiến thức :** **isinstance**
  4. **Ví dụ**

```
# Ví Dụ 1
if (1) return -> TypeMismatchInStatement

# Ví Dụ 2
if (true) return  -> TypeMismatchInStatement
elif (1)

# Ví Dụ 3 -> true
dynamic a
dynamic b
dynamic c
for a until b by c return
# a,c kiểu number và b kiểu bool

# Ví Dụ 3 -> TypeMismatchInStatement
dynamic a <- true
dynamic b
dynamic c
for a until b by c return
# a,c kiểu number và b kiểu bool
```

- For an assignment statement, the left-hand side can be in any type except void type. The right-hand side (RHS) is either in the same type as that of the LHS or in the type that can coerce to the LHS type. When LHS is in an array type, RHS must be in array type whose size is the same and whose element type can be either the same or able to coerce to the element type of LHS.
- The type of a conditional expression in an if or in a for statement must be boolean.
  1. **ý tưởng :** Kiểm tra về LHS và RHS có chung kiểu hay không
  2. **Gợi ý :** visit sau đó dùng hàm **comparType**
  3. **Kiến thức :** **isinstance**
  4. **Ví dụ**

```
# Ví Dụ 1
number a[1,2,3]
number b[1,2,4]
a <- b -> TypeMismatchInStatement
```

- For a call statement <method name>(<args>), the callee must have VoidType as return type. The number of arguments and the number of parameters must be the same. In addition, the type of each argument must be the same as the corresponding parameter.
  1. **ý tưởng :** giống function call
  2. **Gợi ý :** giống function call

3. **Kiến thức :** isinstance

- For a return statement, if the return type of the enclosed function is VoidType, the expression in the return statement must be empty. Otherwise, the type of the return expression must be the same as the return type of the function.
- For a declaration, if there is the initialization expression in a variable declaration, the type of the declaration and initialization expression must conform the type rule for an assignment described above.

1. **ý tưởng :** giống assignment statement
2. **Gợi ý :** giống assignment statement
3. **Kiến thức :** isinstance

- **2.4 Type Cannot Be Inferred** Này khá dễ hiểu là nếu LHS or RHS đều không xác định kiểu thì trả về lỗi, nếu 1 trong hay bên xác định được thì gán cho bên còn lại, trong expr cũng tương tự

```
# Ví Dụ 1
number a[1,2,3]
var b <- a -> bên phải đã biết kiểu nên b cũng có kiểu tương tự

# Ví Dụ 2
dymanic a
var b <- a + 1 -> vì + -> a là numbertype và b cũng là numbertype vì biến dùng cả kết quả nu

# Ví Dụ 2
func main() begin
    dymanic a
    return a -> Type Cannot Be Inferre
end
```