💶

PPL

1. Camera
    a. 2 camera: 1 trực tiếp, 1 bên hông
    b. Nên sử dụng 2 camera
2. Lấy đề: BKEXam
3. Không sử dụng giấy nháp, nếu cần nháp thì nháp thẳng lên bài làm và gạch bỏ
4. Làm trên giấy rồi chụp lại
5. Nội dung thi
    ▼ Câu 1: Viết AST Generation
        1. Cho 1 đoạn grammar, dựa trên đoạn grammar viết AST
        2. vardecl có s, vardecl *
        3. các tín hiệu kết thúc, không kết thúc.
    ▼ Câu 2: Static checker - ass3
        Như programing code, có thể ngắn hơn cả programing code
        Có ít quy định nhưng phải tuân theo các quy định đó
        Đọc đề kỹ và viết cho kỹ
        GIỐNG programing code - làm name hoặc type
    ▼ Câu 3 - Mã Jasmine
        Thầy sẽ cho 1 đoạn mã, Java, C++, Python ⇒ yêu cầu viết mã Jasmine tương đương
        Ví dụ: a = 5;

Mình sẽ có, vị dụ a nằm ở index số 1 thì cái a = 5 sẽ tương đương với dòng gì? iconst, istore1,....

Nó có thể là mảng, có thể là hàm

▼ Câu 4 Suy diễn kiểu

Quiz đã làm, đề năm ngoái đã làm

▼ Câu 5 Cơ chế truyền tham số, cơ chế gọi chương trình con

Rơi vào 2 cái: (hoặc là truyền tham số, hoặc là cơ chế gọi chương trình con)

- Thư nhất: Cơ chế truyền tham số (Lưu ý, ra đề mở, thường là sẽ hỏi trong ngôn ngữ C++, Python). Một đoạn code như vậy đó thì đó là cơ chế gì, giải thích rõ ràng cụ thể cơ chế đó ra
- Thứ 2: Các cơ chế gọi trả về chương trình con. Gọi trả về đơn giản, schedule, exception. (các cơ chế gọi chương trình con)

6. Phải chụp cho rõ

Ôn tập

▼ Câu 1 AST

Câu 1: Cho một đoạn ngữ pháp được mô tả bằng ANTLR như sau:
program: vardecl+ EOF;
vardecl: LET idlist TDOTS typlist (EQ explist)?;
idlist: ID CM idlist | ID;
typlist: typ (CM typ)*;
explist: uexp (CM uexp)*;
uexp: US | exp;
trong đó, LET, TDOTS, EQ, CM, ID, US lần lượt là ký tự kết thúc cho các từ khóa hoặc danh hiệu, … uexp là một ký tự không kết thúc đại diện cho một biểu thức (nếu gán cho biến tương ứng) hoặc US (nếu biến không được khởi tạo).
Viết các phương thức visitVardecl, visitIdlist, visitExplist trong lớp ASTGeneration để sinh ra cây AST tương ứng với các lớp AST như sau:
class Program(ABC): #decl: List[VarDecl]
class VarDecl(ABC): #id: str, typ: Type, exp: Expr if initialized, otherwise None
class Type(ABC): pass
class Expr(ABC): pass
Ví dụ: let a, b, c … int, float, int = 15, _, 15 + 8 thì sinh ra Program([VarDecl(a, IntType(), IntLit(15)), VarDecl(b, FloatType(), None), VarDecl(c, IntType(), BinExp(+, IntLit(15), IntLit(8)))]
Các phương thức khác, sinh viên phải viết giả định với kết quả phù hợp với các phương thức sinh viên được viết.

Câu 1: Cho một đoạn ngữ pháp được mô tả bằng ANTLR như sau:
program: vardecl+ EOF;
vardecl: LET idlist TDOTS typlist (EQ explist)?;
idlist: ID CM idlist | ID;
typlist: typ (CM typ)*;
explist: uexp (CM uexp)*;
uexp: US | exp;
trong đó, LET, TDOTS, EQ, CM, ID, US lần lượt là ký tự kết thúc cho các từ khóa hoặc danh hiệu, … uexp là một ký tự không kết thúc đại diện cho một biểu thức (nếu gán cho biến tương ứng) hoặc US (nếu biến không được khởi tạo).
Viết các phương thức visitVardecl, visitIdlist, visitExplist trong lớp ASTGeneration để sinh ra cây AST tương ứng với các lớp AST như sau:
class Program(ABC): #decl: List[VarDecl]
class VarDecl(ABC): #id: str, typ: Type, exp: Expr if initialized, otherwise None
class Type(ABC): pass
class Expr(ABC): pass
Ví dụ: let a, b, c … int, float, int = 15, _, 15 + 8 thì sinh ra Program([VarDecl(a, IntType(), IntLit(15)), VarDecl(b, FloatType(), None), VarDecl(c, IntType(), BinExp(+, IntLit(15), IntLit(8)))]
Các phương thức khác, sinh viên phải viết giả định với kết quả phù hợp với các phương thức sinh viên được viết.

▼ Câu 3

Note: Suy diễn kiểu hay là biểu thức kiểu

def foo(x,f) = f(f(x))

What is the type of function foo? Note that if a variable type is used, its name is T.

The type of function foo is T

Làm lại | Hiện đáp án

Your score is 0/1.

1. foo is a function so its type is T1->T2 where T1 is input type and T2 is output type

2. This function has 2 parameter, so T1 = T3*T4 where T3 is the type of x (2a) and T4 is the type of f (2b)

3. In the body of the function, there is expression f(x), so f is a function and its type is T5 -> T6 (3a) and x is passed to function f and from (2a) => T5 = T3 (3b)

4. there is also expression f(f(x)) and (3a) => T5 = T6 (4)

5. The result of expression is also the result of function foo and (1) and (3a) => T2 =T6 (5)

6. From (1) and (2a) => type of foo is T3*T4 -> T2 (6)

7. From (6) and (3a) => type of foo is (T3*(T5->T6)) -> T2 (7)

8. From (7) and (3b), (4), (5) => type of foo is (T3*(T3->T3))->T3

There is only one variable type in the result so we can write the type of foo is (T*(T->T))-> T

▼ Câu 4 Jasmine

Trong ngôn ngữ Python, ta có đoạn mã sau:
o = {‘a’: 15, ‘b’: 16} def foo(a):
a[‘a’] = a[b] + a[‘a’] foo(o)
print(o[‘a’], o[‘b’])
Hãy cho biết kết quả và giải thích cơ chế truyền tham số kể trên.

UndeclaredType(name) sẽ được quăng ra. Ví dụ với khai báo biến cuối cùng sử dụng kiểu vd đã được khai báo trước đó nên nó hợp lệ. Nhưng nếu không có kiểu vd được khai báo trước đó, lỗi UndeclaredType(vd) được quăng ra.

Câu 3: Hãy chuyển đổi đoạn mã sau đây trong C/C++ sang mã Jamin tương ứng:

```
int a[7] = {1, 2, 3, 4, 5, 6, 7};
int b = 1;
a[b * 2 + 3] = a[b * 3 + 2] + 2;
```

Biết rằng, a và b có chỉ số tương ứng là 1 và 2.

Câu 4: Trong ngôn ngữ Python, ta có đoạn mã sau:

```
o = {'a': 15, 'b': 16}
def foo(a):
    a['a'] = a[b] + a['a']
foo(o)
print(o['a'], o['b'])
```

Hãy cho biết kết quả và giải thích cơ chế truyền tham số kể trên.

▼ Câu 5 Control Abstraction

▼ Râu ria

▼ 3 cấp điều khiển trình tự:

- Biểu thức
- Phát biểu
- Cấp đơn vị: Điều khiển trình tự của việc gọi các chương trình con => Trừu tượng hóa điều khiển.

▼ ĐỊNH NGHĨA CHƯƠNG TRÌNH CON

Bao gồm:

- Đặc tả: tên chương trình, thông số:
  - input, ouput,
  - thứ tự thông số,
  - kiểu thông số,
  - cơ chế truyền thông số.
  - Hành vi:

- Hiện thực:
  - Viết thân của một chương trình con:
  - Dữ liệu cục bộ
  - Tổng hợp các statement.

▼ SUBPROGRAM ACTIVATION

Định nghĩa chương trình con - bảng hoạt động:
+ Tạo ra mỗi lần chương trình con được gọi
+ Khi thực thi chuognw trình con hoàn tất thì bảng hoạt động đó bị xóa đi
+ Có nhiều bảng hoạt động, tồn tại ở nhiều thời điểm khác nhau
+ Trong trường hợp gọi đệ quy thì nhiều bảng hoạt động của chương trình con tồn tại cùng một lúc.

▼ Bảng hoạt động chương trình con:

+ Static part: Phần mã của chương trình con, là code mình viết.
+ Dynamic: Bảng ghi hoạt động - Activation Record
- thông số
- local data
- địa chỉ trả về - return address
- other links

Quiz:

```
val timeOut = new javax.swing.AbstractAction() {
def actionPerformed(e: java.awt.event.ActionEvent) = op
}
```

##params của actionPerformed bao gồm this và e.

```
def apply(interval: Int, repeats: Boolean = true)
(op: => Unit) {
```

##apply bao gồm interval, repeats, op

▼ CƠ CHẾ GỌI CHƯƠNG TRÌNH CON

▼ `Gọi trả về đơn giản - Single Call return`

- Không cho phép gọi đệ quy: Trong ngôn ngữ lập trình mã máy.
- Trong một số ngôn ngữ lập trình mã máy Assembly
- Không có đệ quy để việc gọi trả về đơn giản hơn
- `Có lệnh gọi rõ ràng, tường minh.`
- Chỉ có một điểm vào của chương trình. Mỗi khi gọi là nó sẽ bắt đầu vào chính điểm đó.
- khi mà được gọi thì chương trình được gọi sẽ được thực thi tức thời
  - Truyền 1 cách tức thời: Ví dụ khi chương trình A gọi B thì B sẽ chạy ngay tức thoiwf, chương trình A sẽ nhường cho chương trình B chạy.
  - A đang chạy, A gọi B ⇒ B sẽ chạy, A nhường cho B
- Trong một thời điểm chỉ có 1 chương trình chạy ⇒ Single Execution

▼ `Đệ quy`

- Gọi trực tiếp
  - Chương trình sẽ gọi lại chính nó
- Gọi gián tiếp
  - Gọi thông qua những chuương trình khác.
- 4 thuộc tính còn lại giống Single Call-return: Lệnh gọi tường minh, có 1 điểm vào, thực thi đơn, thực thi tức thời
- Chỉ khác với trả về đơn là gọi đệ quy

▼ `Exception - Biến cố và xử lí biến cố`

- Không có lệnh gọi tường minh. Không có lệnh gọi chương trình.
- Được dùng:

- Trong lập trình hướng sự kiện: Được thực thi khi có một sự kiện gì đó xảy ra.
- Ví dụ như định nghĩa 1 chương trình: khi người ta click chuột thì CT đó sẽ làm gì?
  Người ta nhấn A thì sẽ làm gì,...
  Viết những trình, mà trình đó sẽ thực thi khi có 1 sự kiện gì đó xảy ra
  Chương trình xử lí các sự kiện xảy ra
- Exception

- Error Handler: Xử lý biến cố: nén vào biến cố. Khi mà bắt biến cố sẽ thực thi xử lý biến cố.

  `Throw exception`

- Các ngôn ngữ phải đặc tả
  - Những biến cố nào sẽ được xử lý, mà nó được định nghĩa như thế nào
    Các biến cố đã do ngôn ngữ lập trình định nghĩa sẵn, hoặc người dùng có thể tự tạo thêm các biến cố.
  - Làm cách nào để các exception được raised lên
    Raising exception
    Click, MouseMove, TextChange
    Hệ điều hành ném ra các biến cố
    Các object(Timer)
    By programmer (throw)
  - Làm thế nào để các exception được xử lý

    Định nghĩa protected block: quan sát xem các biến cố có xảy ra hay không ⇒ bắt và xử lý biến cố

    ▼ Termination Semantic

    non-resumable + stack unwinding:

    - Khi xử lý biến cố xong sẽ không quay lại nơi đã xảy ra biến cố

- Kết hợp với stack unwinding: Gọi CT A, B ⇒ đặt bảng hoạt động của A vào stack, tương ứng đặt B vào stack và BHĐ của C vào stack
  Khi C xảy ra biến cố, mà biến cố đó ko được xử lý trong C thì sẽ quay về xử lý trong B ⇒ hủy BHĐ của C ra, tương ứng bỏ B, rồi quay về A xử lý ở A

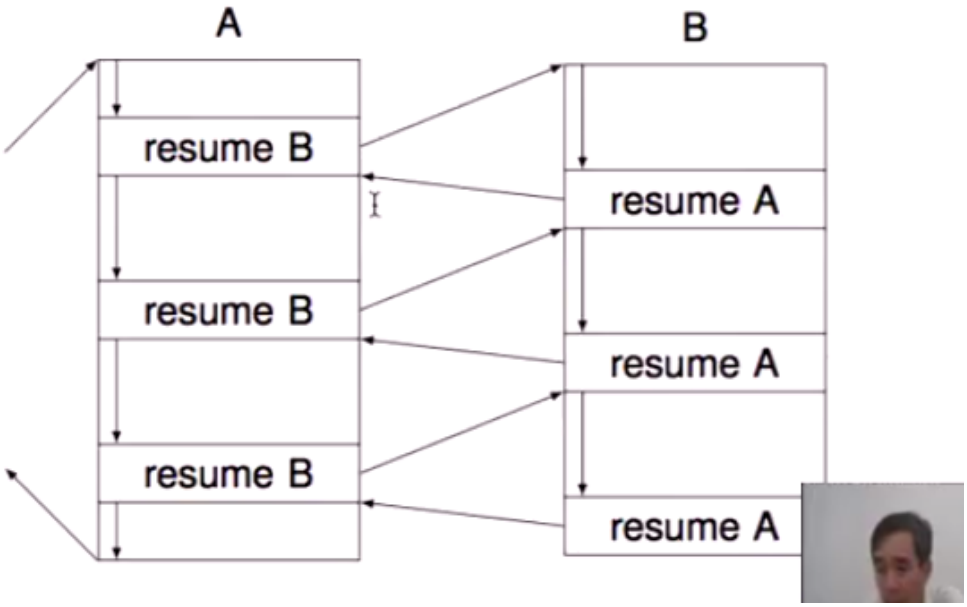
Resumable xử lý biến cố xong rồi quay lại:

- ngay chỗ xảy ra biến cố
- sau khi xảy ra biến cố

▼ `Coroutines - Trình cộng hành`

- Có nhiều điểm vào

A coroutine may postpone its execution and control is back to caller. Its execution later is resumed at the place it postponed.



- Điểm chuyển qua lại giữa A, B là do người lập trình xác định

▼ `Cơ chế phân tasks`

- Thực thi đồng thời nhiều tasks khác nhau
- Dựa trên các máy có thể đa xử lý
- Máy đơn xử lý có thể xử dụng time sharing. Chia thời gian, mỗi thời gian chay tasks này, rồi dành cho tassks kia
- Tương tự như coroutines tuy nhiên không phải do người lập trình quyết định
- Chuyển trình dựa vào cơ chế. người lập trình ko quyết định được.
- chạy trong máy đa xử lý.
- Một thời điểm có nhiều chương trình con chạy đồng thời với nhau.

▼ Tuy nhiên, sẽ dẫn đến nhiều vấn đề

Các vấn đề về đồng bộ

- Race condition
- Deadlock

Các vấn đề truyền dữ liệu giữa cá tasks

▼ `Cơ chế định thời:`

- Khác vói cơ chế trả về đơn giản: Ngay tức thời và không tức thời
- Khi được gọi thì chương trình con sẽ không được thực thi ngay lập tức, mà phỉa dựa vào bộ định thời
- bộ định thời quyết định chuognw trình nào sẽ chạy, được định thời bởi 2 cách
  - Định thời bởi thời gian
  - Định thời bởi quyền ưu tiên

- Controlled by a scheduler

TRUYỀN THAM SỐ

`Khai báo thông số:`

- `Thông số hình thức:`
  - Ví dụ: int foo(float x, bool& Y) ⇒ 3 thông số
    2 thông số truyền vào đó là x và y
    1 thông số truyền ra đó là int

  Là thông số khai báo
  Chỉ là một tên đơn, không khởi tạo
  Giống khai báo biến, thường có thêm kí hiệu

- `Thông số thực:` Là argument truyền vào, là một expression

  Khi gọi 1 hàm, thì cái gì mình truyền vào đó thì gọi là thông số thực

  Thường là biểu thức (expression)

`Formal - Actual:` Tương ứng giữa thông số thực và thông số hình thức

- `Truyền bằng vị trí:` Có bao nhiêu thông số thì phải truyền đủ

  Ví dụ int foo(int a, int b) ⇒ foo(1,2)

- `Truyền bằng tên:` Phải nhớ tên của thông số hình thức. Để truyền vào cho nó. Không quan trọng thứ tự. Biết được tên thông số nên biết truyền vào đúng vào cái gì
  nhược: phải nhớ tên các thông số
  ví dụ: int foo(int a, int b) ⇒ foo(b = 1, a = 2)

CÁC PHƯƠNG PHÁP TRUYỀN THÔNG SỐ

Có ba nhóm:

`Input - output:`

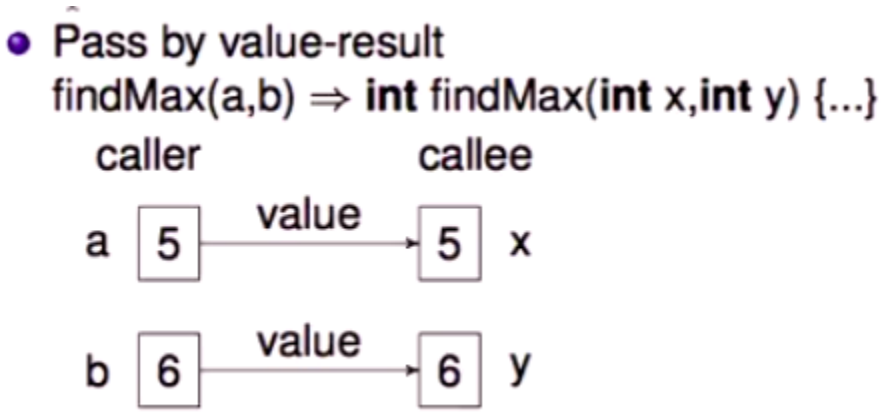
Những thông số dùng vừa để truyền vào, vừa đề truyền ra, truyền vào cho chương trình con, sau đó chương

trình con truyền ra.

- `Truyền bằng trị kết quả` - pass by value-result:

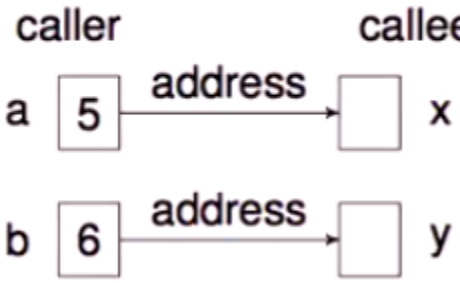
  a = 5, b = 6. truyền a và b.
  findMax(a, b). tương đương a truyền cho x và b truyền cho y.
  Khi gọi, thông số thực truyền giá trị cho thông số hình thức ⇒ x nhận giá trị 5, y nhận giá trị 6

  - khi thực thi, các giá trị được truyền vào x và y thay đôi, tuy nhiên vẫn độc lập với a và b
  - Khi chương trình kết thúc, sẽ chép nguược lại giá trị của x vào a, và y vào b. `x = 7, y = 8 ⇒ a = 7, b = 8`
  - Sau đó thông số hình thức pass ngược lại cho thông số giá
    trị.



- `Truyền bằng tham khảo = pass by reference`:

- Pass by reference
  findMax(a,b) ⇒ **int** findMax(**int&** x,**int&** y) {...}

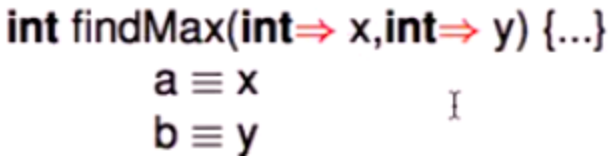


- Truyền địa chỉ vào, ⇒ x và y sẽ là các bí danh alias của a và b.
- X và y thay đổi thì a và b cũng thay đổi theo
- xác định được địa chỉ của thông số thực. Mối liên kết giữa formal và actual khong thay đổi.
- Khi x và y thay đổi thì a và b cũng thay đổi theo.
- Thực hiện theo kiểu realtime. Khi x, y thay đổi thì a và b cũng thay đổi theo.
- Khi chương trình kết thúc, thì x y bị hủy bỏ ⇒ a và b cũng đã được thay đổi
- Còn pass bay value-result thì sau khi xong tất cả rồi mới pass.

▼ `Pass by name - truyền nguyên cái tên vào:`

không truyền địa chỉ vào, không truyền giá trị vào mà truyền nguyên thông số thực

- Pass by name
  findMax(a,b) ⇒ **int** findMax(**int⇒** x,**int⇒** y) {...}

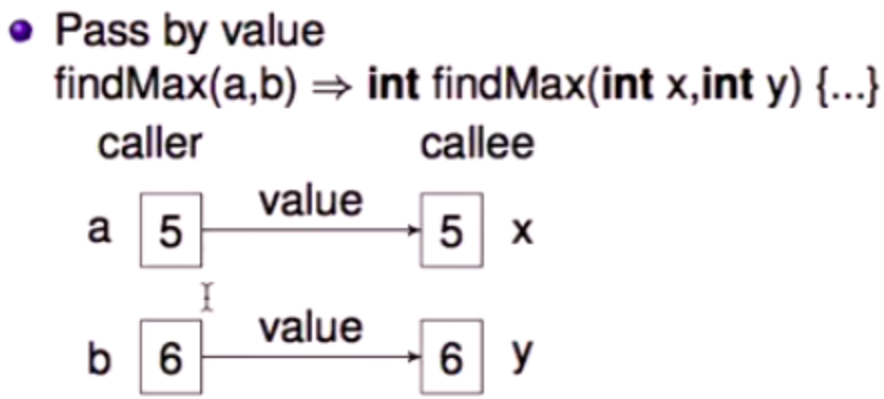


Nếu là a + 1 hoặc b/2 thì nó sẽ truyền nguyên cái expression đó vào. Việc tính toán a + 1, b/2 sẽ được thực thi khi mà chúng ta tham khảo vào x, y trong chương trình findMax

- Input only:

  Truyền vào chương trình con.

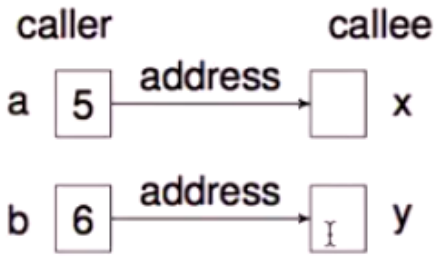
  - `Pass by value - truyền bằng trị:`



    - là 1 nửa của truyền kết quả
    - chỉ thực hiện quá trình truyền vào, không thực hiện quá trình truyền ra
    - Sao chép giá trị của a và b, sự thay đổi của x, y không ảnh hưởng gì đến a và b.
    - Sau khi kết thúc thì sẽ hủy x và y, a và b không bị thay đổi.
    - Khác với truyền bằng trị kết quả.

  - `Pass by constant reference - truyền bằng tham khảo hằng:`

- Pass by constant reference
  findMax(a,b) ⇒ **int** findMax(**const int&** x,**const int&** y) {...}



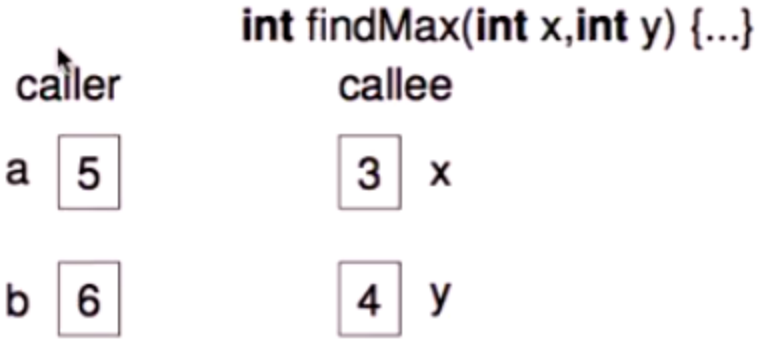
  - Truyền địa chỉ thông số thực vào thông số hình thức
  - Thông số hình thức được coi như 1 hằng, ⇒ findMax không thể làm thay đổi x và y ⇒ a và b ko thay đổi
  - vào Đối với thông số có giá trị lớn, chỉ truyền địa chỉ vào thì sẽ tốn ít tài nguyên hơn.

▼ Output only:

Thông số từ bên trong chưa trình con trả ra

▼ `Pass by result`:

- Pass by result



Chép ngược giá trị kết quả từ chương trình.

Tuy nhiên giá trị của a và b thì không được truyền vào.

Chỉ là nhận mà thôi

▼ `As a result of a function:`

int foo()...return 0;

Không có thông số thực, no parameter: `foo() + 1`

▼ Note các kiểu pass trong C++ và Python

Python: the parameter passed in is actually a reference to an object (but the reference is passed by value)

5️⃣ Câu 5

Note all

▼ Câu 1

Coi lại đề ví dụ +

```
program: exp EOF;
exp: (term ASSIGN)* term;
term: factor COMPARE factor | factor;
factor: operand (ANDOR operand)*;
operand: ID | INTLIT | BOOLIT | '(' exp ')';
INTLIT: [0-9]+ ;
BOOLIT: 'True' | 'False' ;
ANDOR: 'and' | 'or' ;
ASSIGN: '+=' | '-=' | '&=' | '|=' | ':=' ;
COMPARE: '=' | '<>' | '>=' | '<=' | '<' | '>' ;
ID: [a-z]+ ;
```

```
and AST classes as follows:
class Expr(ABC):
class Binary(Expr):  #op:string;left:Expr;right:Expr
class Id(Expr): #value:string
class IntLiteral(Expr): #value:int
class BooleanLiteral(Expr): #value:boolean



program: vardecl+ EOF;
vardecl: mptype ids ';' ;
mptype: INTTYPE | FLOATTYPE;
ids: ID (',' ID)*;
INTTYPE: 'int';
FLOATTYPE: 'float';
ID: [a-z]+ ;
and AST classes as follows:
class Program:#decl:list(VarDecl)
class Type(ABC): pass
class IntType(Type): pass
class FloatType(Type): pass
class VarDecl: #variable:Id; varType: Type
class Id: #name:str



program: exp EOF;
```

exp: term ASSIGN exp | term;

term: factor COMPARE factor | factor;

factor: factor ANDOR operand | operand;

operand: ID | INTLIT | BOOLIT | '(' exp ')';

INTLIT: [0-9]+ ;

BOOLIT: 'True' | 'False' ;

ANDOR: 'and' | 'or' ;

ASSIGN: '+=' | '-=' | '&=' | '|=' | ':=' ;

COMPARE: '=' | '<>' | '>=' | '<=' | '<' | '>' ;

ID: [a-z]+ ;

and AST classes as follows:

▼ Câu 3 Jasmine

int arr

arr

gọi hàm 2 biến

▼ Câu 4

foo(x, y, z) = if else

coi trong scorm

def foo(x,y,z) = if y(x) then z(x) else x + 1

1. Dựa vào điều kiện dấu cộng chỉ sử dụng cho số nguyên, ta có kiểu của x là Integer
2. Trong cùng 1 nhánh, kiểu đầu ra của z(x) cũng là Integer
3. Từ 1 và 2, kiểu của z(x) là Integer ⇒ Integer
4. y(x) nằm trong câu điều kiện, vậy kiểu đầu ra của y(x) là boolean ⇒ kiểu của y(x) là Integer ⇒ Boolean

5. Câu if else trả về kiểu Integer do đó kiểu trả về của foo cũng là Integer
6. Kiểm của foo(x, y, z): Integer * (Integer ⇒ Boolean) * (Integer ⇒ Integer) ⇒ Integer

1. Gọi T1 là kiểu của x, T2 là kiểu của y, T3 là kiểu của z
2. Dựa vào..... kiểu của x là Integer ⇒ T1 = Integer
3. z(x) sẽ trả về kiểu Integer do đó kiểu của z(x), T3 = T1 ⇒ Integer
4. y(x) trong câu điều kiện, sẽ trả về boolean vậy biểu thức kiểu của y(x), T2 = Integer ⇒ Boolean
5. Vế phải có kiểu trả về là Integer, do đó kiểu trả về của foo(x,y,z) là Integer
6. Biểu thức kiểu của foo: T1 * T2 * T3 ⇒ Integer
7. Từ 2, có T1 = Integer
   T2 = Integer ⇒ Boolean
   T3 = T1 ⇒ Integer = Integer ⇒ Interger

def foo(x,y,z)=y(z(x)+1)

1. Dựa vào phép cộng chỉ dành riêng cho số nguyên
   Kiểu của z(x) là kiểu nguyên
2. Gọi T1 là kiểu của x, T2 là kiểu của y
3. hàm foo: T1*T2*Integer ⇒ T2

1. Gọi T1 là kiểu của x
2. Dựa vào điều kiện chỉ có số nguyên mới có thể sử dụng phép toán + do đó đầu ra của z(x) là Integer: vậy kiểu của z(x) là T1 ⇒ Integer
3. Gọi T2 là kiểu đầu ra của y(z(x)+1), kiểu của y(z(x)=1) là Integer ⇒ T2

4. T2 là kiểu đầu ra của y(z(x) + 1), do đó cũng là kiểu đầu ra của hàm foo
5. Quay về với hàm foo, ta có kiểu của x là T1, kiểu của y là Integer ⇒ T2, kiểu của z là T1 ⇒ Integer
6. T1*(Integer ⇒ T2) * (T1 ⇒ Integer) ⇒ T2

quiz

▼ Câu 5

▼ C++ so sánh swap giữa * và &

Nêu định nghĩa giữa 2 cách so sánh và cho ví dụ

So sánh giữa 2 cách

Câu 4: Trong ngôn ngữ C, do không có kiểu truyền tham số bằng tham chiếu nên người ta dùng con trỏ như sau:
int swap(int* a, int* b) { int c = *a; *a = *b; *b = c; }
Hãy viết một ví dụ sử dụng hàm swap kể trên và giải thích bản chất của kiểu truyền tham số như trên.

▼ C++ so sánh truyền * và constant int

Nêu định nghĩa giữa 2 cách so sánh và cho ví dụ

So sánh giữa 2 cách

▼ Exception

So sánh bắt ngoại lệ giữa C++ và Python

Đưa ví dụ bắt Exception trong Python

Midterm Thin noted