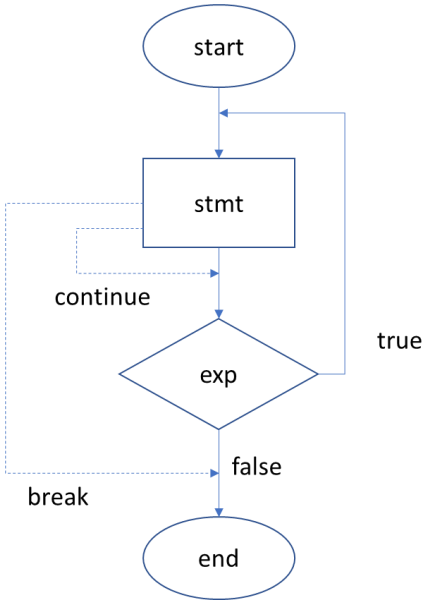
1) Do stmt while exp

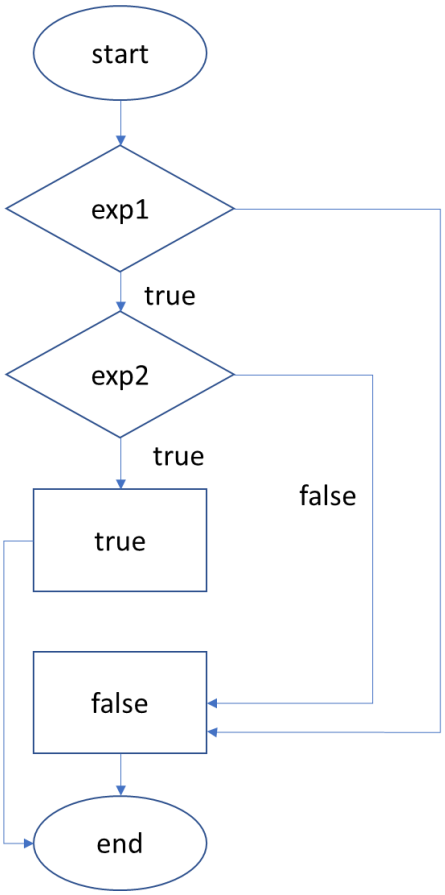


- Khi ta vào từng statement trong stmt, nếu ta gặp lệnh continue hoặc break thì nó sẽ gọi lệnh GOTO để đến nhãn của continue hoặc break tương ứng bên trong do while statement.

```
Def visitDowhile(Self, ast, frame):
        Frame.enterLoop()
        Label = frame.getNewLabel()
        contLb = frame.getContLabel()
        brkLb = frame.getBreakLabel()
        Self.emit.emitLABEL(Label, frame)
        [Self.visit(x) for x in ast.body]
        Self.emit.emitLABEL(contLb, frame)
        Self.visit(expr)
        Self.emit.emitIFTRUE(Label, frame)
        Self.emit.emitLABEL(brkLb, frame)
        Frame.exitLoop()
Def visitContinue(self, ast, frame)
```

Self.emit.emitGOTO(frame.getContLabel(), frame)

2) +, /, AND



Prototype:
Visit(exp1)
emitIFFALSE(false label)
Visit(exp2)
emitIFFALSE(false label)
emitPUSHICONST(1)
emitGOTO(end label)
emitLABEL(false label)
emitPUSHICONST(0)
emitLABEL(end label)

Short-circuit cho and

```
Def visitBinaryOP(self, ast, frame):
        Op = ast.op
        Left, typL = self.visit(ast.left, frame)
        Right, typR = self.visit(ast.right, frame)
        typBin = typL
        Res = “ “
        If op is ‘+’:
                If type(typL) is IntType and type(typR) is FloatType:
                        Left += self.emit.emitI2F(frame)
                        typBin = FloatType()
                Elif (type(typR) is IntType and type(typL) is FloatType:
                        Right += self.emit.emitI2F(frame)
                        typBin = FloatType()
                Res += Left + Right + Self.emit.emitADDOP(op, typL, frame)
```

```
elif op is '/':
    If type(typL) is IntType:
        Left += self.emit.emitI2F(frame)
    Elif (type(typR) is IntType:
        Right += self.emit.emitI2F(frame)
    Res += Left + Right +Self.emit.emitMULOP(op, FloatType(),frame)
    typBin = FloatType()
Elif op is 'AND':
    falseLb = frame.getNewLabel()
    endLb = frame. EndLabel()
    Res += Left
    Res += Self.emit.emitIFFALSE(falseLb, frame)
    Res += Right
    Res += Self.emit.emitIFFALSE(falseLb, frame)
    Res += Self.emit.emitPUSHICONST(1, frame)
    Res += Self.emit. emitGOTO(endLb, frame)
    Res += Self.emit. emitLABEL(falseLb, frame)
    Res += Self.emit.emitPUSHICONST(0, frame)
    Res += Self.emit. emitLABEL(endLb, frame)
Return Res, typBin
```

3) Kiểu tham khảo và kiểu con trỏ
   - Kiểu con trỏ là biến chứa giá trị gồm địa chỉ vùng nhớ và giá trị đặc biệt nil. Con trỏ tham khảo thông qua địa chỉ
   - Kiểu tham khảo là biến tham khảo thông qua đối tượng hoặc giá trị

Pointer	Reference
Int A; Int *pA = &A;	Int A; Int &rA = A;
*pA => A	rA => A
pA++	Không có increment
pA = &B	Không thể thay đổi giá trị tham khảo
pA = null	Không được phép null
Int *pA	Bắt buộc phải có giá trị khởi tạo

   - Alias là hiện tượng một đối trượng bị bao bởi 2 tên khác nhau trong cùng một thời gian:
     + Với kiểu con trỏ khi khai báo 2 con trỏ, cùng trỏ vào một địa chỉ trong bộ nhớ thì sẽ gây ra hiện tượng alias. Vd trên C++

```
Int main(){
    Int *p, *q;
    p = new int;
```

```
    *p = 1;
    q = p;
    *q = 2;
}
```

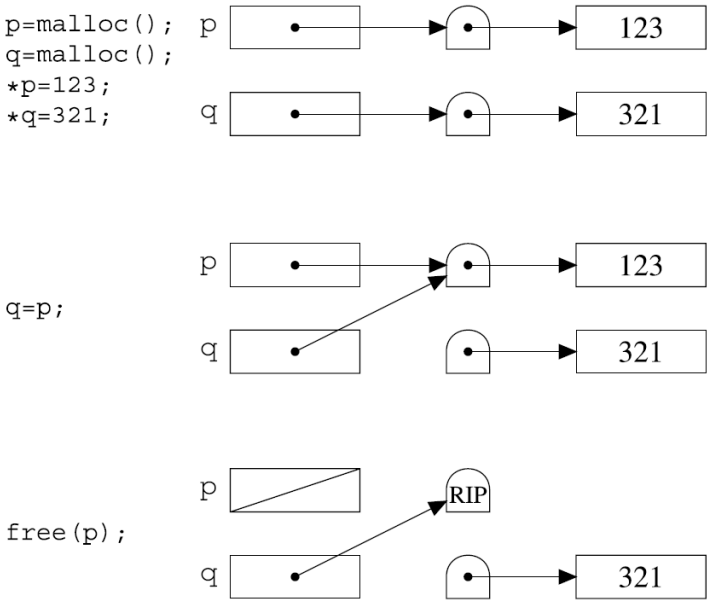
+ Với kiểu tham khảo khi 2 khai báo cùng tham khảo đến một đối tượng trong cùng thời điểm. vd trên C++

```
Int main(){
    Int a = 5;
    Int &rA1 = a ;
    *rA1 = 6;
    Int &rA2 = a;
    *rA2 = 7;
    }
```

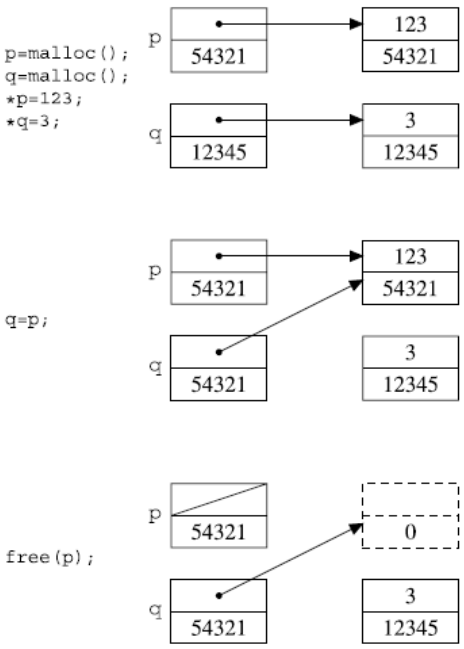
4) So sánh các cơ chế gọi chương trình con
   - Gọi trở về đơn giản:
     + Không gọi đệ qui
     + Lệnh gọi tường minh
     + Chỉ có duy nhất 1 điểm vào chương trình (entry point)
     + Truyền tức thời. Vd: A gọi B thì B thực thi tức thời, chương trình A tạm ngưng
     + Thực thi đơn. Trong 1 thời điểm chỉ có 1 chương trình chạy
   - Đệ qui
     + Có thể gọi đệ qui bằng cách trực tiếp hoặc gián tiếp
     + Còn lại y chang simple call
   - Biến cố
     + Không có lệnh gọi tường minh
     + Với xử lý sự kiện chỉ thực thi khi có sự kiện mà mình đã hiện thực trình xử lý thông qua trình điều khiển chung
     + Với xử lý lỗi, chương trình chỉ thực thi khi bắt được biến cố và ném ra lỗi -> trình xử lý biến cố
   - Trình cộng hành
     + Có nhiều điểm vào chương trình
     + Thực thi đơn
     + Luân phiên giữa các chương trình mà không bắt đầu lại từ đầu chương trình
     + Thời điểm chuyển đổi phụ thuộc người lập trình
   - Trình định thời
     + Không truyền tức thời. Vd: A gọi B ko chuyển lập tức mà mức độ trì hoãn phụ thuộc vào cách định thời

- Công tác
    + Thực thi đồng thời nhiều task
    + Việc chuyển task trên máy đơn nhân phụ thuộc vào máy dựa trên cơ chế time sharing

5) Tham chiếu treo
    - Tombstone: vùng nhớ heap sẽ tạo một cell trung gian là tombstone. Khi khai báo một biến con trỏ và gán địa chỉ cần tham chiếu, nó sẽ trỏ đến tombstone và từ tombstone mới trỏ đến vùng cần tham chiếu. Khi có tham chiếu treo xảy ra, thì tombstone sẽ được thay thế bằng giá trị nil khi được truy xuất từ con trỏ còn lại. Vd:



    - Lock and key: khi một đối tượng được tạo trong vùng nhớ heap, nó sẽ liên kết với lock tương ứng với một word trong bộ nhớ và lưu một giá trị ngẫu nhiên (thường tránh giá trị 0 và 1). Pointer khi được khởi tạo và tham chiếu tới đối tượng sẽ gồm một cặp địa chỉ và key (với giá trị của key tương ứng với giá trị lock của đối tượng). Khi một đối tượng bị hủy thì giá trị lock sẽ bị thay đổi thành giá trị mặc định (như 0 hoặc 1). Khi tham chiếu treo xảy, key của con trỏ còn lại sẽ không trùng khớp với lock của đối tượng đã bi hủy nữa sẽ báo lỗi. Vd:



6) Inference

1. Hàm H(x, f, h) là hàm gồm 3 thông số:
   H: T1 x T2 x T3 → T4 (1)
   Với x: T1
   f: T2
   h: T3
2. Hàm f(x) là hàm 1 thông số:
   f: T2 ≡ (T5 → T6) (2)
   mà ta có x là tham số truyền vào có kiểu là T1
   ⇨ T1 ≡ T5 (3)

   Ta có biểu thức điều kiện của if là Boolean

   ⇨ T6 ≡ Boolean (4)

   (2), (3), (4) => T2 ≡ (T1 → Boolean) (5)
3. Hàm h(x) là hàm 1 thông số:
   h: T3 ≡ (T7 → T8) (6)
   mà ta có x là tham số truyền vào có kiểu là T1
   ⇨ T1 ≡ T7 (7)

   (6), (7) => T3 ≡ (T1 → T8) (8)
4. Hàm h(h(x)) là hàm một thông số:
   h: T3 ≡ (T9 → T10) (9)
   mà ta có h(x) là tham số truyền vào có kiểu là T8

⇨ T8 ≡ T9 (10)

(9), (10) => T3 ≡ (T8 → T10) (11)

5. Hàm h phải trả về cùng 1 kiểu:
   => T1 ≡ T8 ≡ T10 (12)
   (6), (11), (12) => T3 ≡ (T1 → T1) (13)
6. Biểu thức sau return phải cùng kiểu trả về mà f(x) kiểu trả về là boolean:
   Mà kiểu trả về của h là T1 => T1 ≡ Boolean (14)
   (1), (5), (13), (14) => H: Boolean x (Boolean → Boolean) x (Boolean → Boolean) → Boolean

7) Scope
   a) Môi trường tham khảo

Hàm	Môi trường tham khảo
Main	a//1, b//1, c//1, Main, sub1
Sub1	a//2, b//1, c//1, Main, sub1, sub2, sub3
Sub2	a//3, b//1, c//3, f//3, Main, sub1, sub2, sub3
Sub3	a//2, b//4, c//1, sub1, sub2, sub3

   b) Bản ghi hoạt động
   - Khi Sub3 được gọi lần 1

Main

a	0
b	1
c	4

-> sub1

a	3

->sub2

a	1
c	2
f(a)	
f(b)	

->sub3

b	1

   - Sau khi gọi sub3 lần 1, ta tính được f(a) = 2 * 4 – 3 = 1 và cập nhật vào sub2
   - Sub2 gọi tiếp sub3 lần để tính ra kết quả f(b) = 2*4 – 3 = 5 và cập nhật tiếp vào sub2

->sub2

a	1
c	2
f(a)	1
f(b)	

->sub3

b	2

->sub2

a	1
c	2
f(a)	1
f(b)	5

-> Main

a	0
b	-8
c	4

- Trong sub1 sẽ tính ra được b = (1 – 5) * 2 = -8 và cập nhật vào biến b//1 ở Main

8) Pass through
   a) Truyền trị - kết quả
   - j -> rvalue(0, addr(j)) -> i với addr(j) là địa chỉ của j để trả kết quả về
   - A[j] -> rvalue(4, addr(A[0]) với addr(A[0]) là địa chỉ của A[0] để trả kết quả về

Khởi tạo ban đầu

static		Main		sumAndDecrease	
A[0]	4	s		a	4
A[1]	6			i	0
A[2]	14			s	0
j	0				
n	3				

Lần lặp 1

static		Main		sumAndDecrease	
A[0]	3	s		a	4
A[1]	6			i	1
A[2]	14			s	4
j	0				
n	3				

Lần lặp 2

static		Main		sumAndDecrease	
A[0]	2	s		a	4
A[1]	6			i	2
A[2]	14			s	8
j	0				
n	3				

Lần lặp 3

static		Main		sumAndDecrease	
A[0]	1	s		a	4
A[1]	6			i	3

A[2]	14
j	0
n	3

s	12

Lần lặp 4: kết thúc vòng lặp trả kết quả về

static Main

A[0]	4
A[1]	6
A[2]	14
j	3
n	3

s	12

- Khi kết thúc, kết quả của a được truyền về lại A[0] và i truyền về j, bảng hoạt động của sumAndDecrease cũng bị hủy
  ⇨ Kết quả in ra là 12 4 6 14

b) Truyền tham khảo

- j -> lvalue(0) -> i
- A[j] -> lvalue(A[0]) -> a
  ⇨ Kết quả thay đổi trực tiếp trên static như sau:

Khởi tạo ban đầu

static Main sumAndDecrease

A[0]	4
A[1]	6
A[2]	14
j	0
n	3

s	

a	addr(A[0]
i	addr(j)
s	0

Lần lặp 1

static Main sumAndDecrease

A[0]	3
A[1]	6
A[2]	14
j	1
n	3

s	

a	addr(A[0])
i	addr(j)
s	4

Lần lặp 2

static Main sumAndDecrease

A[0]	3	s		a	addr(A[0])
A[1]	5			i	addr(j)
A[2]	14			s	7
j	2				
n	3				

Lần lặp 3

static Main sumAndDecrease

A[0]	3	s		a	addr(A[0])
A[1]	5			i	addr(j)
A[2]	13			s	10
j	3				
n	3				

Lần lặp 4: kết thúc vòng lặp trả kết quả

static Main

A[0]	3	s	10
A[1]	5		
A[2]	13		
j	3		
n	3		

⇨ Khi kết thúc, bảng hoạt động của sumAndDecrease bị hủy và kết quả in ra là 10 3 5 13

c) Truyền bằng tên

- j -> code tính j -> i
- A[j] -> code tính A[j] -> a
- Lúc này ta có i ≡ j và a ≡ A[j]

Khởi tạo ban đầu

static Main sumAndDecrease

A[0]	4	s		a	A[j] = A[0] = 4
A[1]	6			i	j = 0
A[2]	14			s	0

j	0
n	3

Lần lặp 1

static		Main		sumAndDecrease	
A[0]	3	s		a	A[j] = A[1] = 6
A[1]	6			i	j = 1
A[2]	14			s	4
j	0				
n	3				

Lần lặp 2

static		Main		sumAndDecrease	
A[0]	3	s		a	A[j] = A[2] = 14
A[1]	5			i	j = 2
A[2]	14			s	10
j	0				
n	3				

Lần lặp 3

static		Main		sumAndDecrease	
A[0]	3	s		a	A[j] = A[3]
A[1]	5			i	j = 3
A[2]	13			s	24
j	0				
n	3				

Lần lặp 4: kết thúc vòng lặp trả kết quả

static		Main	
A[0]	3	s	24
A[1]	5		
A[2]	13		
j	3		
n	3		

- Kết thúc chương trình sumAndDecrease thì sẽ cập nhật lại giá trị và hủy bảng hoạt động

⇨ Kết quả in ra sẽ là 24 3 5 13