ÔN TẬP PPL

A. Phần đề

I. Phần câu hỏi lập trình

1. Vòng lặp:

Viết phương thức
+ visitDowhile(ast:Dowhile, o : Context)
+ visitFor(ast:For, o : Context)
+ visitContinue(ast:Continue, o : Context)
+ visitBreak(ast:Break, o : Context)

để sinh mã Jasmin cho một phát biểu Dowhile, For, Continue và Break. Và giải thích làm sao một phát biểu continue cũng như break nằm ở sâu bên trong thân của phát biểu Dowhile, cũng như For có thể chuyển điều khiển đến vị trí thích hợp. Cho biết lớp của các phát biểu trên trong AST được định nghĩa như sau :

class Dowhile(Stmt): sl:List[Stmt] exp: Expr	**class** For(Stmt): expr1:Expr expr2:Expr expr3:Expr loop:Stmt
class Break(Stmt):	**class** Continue(Stmt):

Một số phương thức của Emitter có thể sử dụng (ko giới hạn)	Một số phương thức Frame có thể sử dụng(ko giới hạn)
• emitIFTRUE(self, label:int, frame): • emitIFFALSE(self, label:int, frame)	• enterLoop(self) • exitLoop(self)

• emitGOTO(self, label:int, frame) • emitLABEL(self, label:int, frame) • emitPUSHICONST(self, in_:int, frame) • emitANDOP(self, frame) • emitADDOP(self, lexeme:string, in_:type, frame): • emitMULOP(self, lexeme:string, in_:type, frame):	• getNewLabel(self) • getBreakLabel(self) • getContinueLabel(self) • getStartLabel(self) • getEndLabel(self)

2. Toán tử:

* Giả sử biểu thức nhị phân có các phép toán +, /, AND, OR, trong đó, các phép toán +, / có thể có các toán hạng ở kiểu IntType hoặc FloatType, trong khi phép toán AND, OR chỉ cho phép các toán hạng ở kiểu BoolType. Khi các toán hạng khác kiểu, phải sinh mã chuyển đổi sang kiểu FloatType. Kết quả của phép AND và OR có cùng kiểu toán hạng, trong khi phép + / trả về kiểu FloatType nếu một trong 2 toán hạng kiểu FloatType. Hãy viết

+ visitBinaryOp(ast:BinaryOp, o:Context)

để sinh mã Jasmin cho biểu thức nhị phân. Chú ý phải sinh mã cho phép rút ngắn tính toán (short-circuit) cho AND, OR thì mới được trọn điểm câu này. Nhắc lại khai báo lớp BinaryOp trên cây AST như sau:

```
class BinaryOp(Expr):
    def __init__(self ,op:String,left:Expr,right: Expr)
```

Các phương thức của Emitter và Frame đã nêu ở câu trên

II. Phần câu hỏi tự luận

(Đề 2018)

3. Hãy trình bày về kiểu con trỏ (pointer) và kiểu tham khảo (reference)? Cho ví dụ cho mỗi kiểu? Nêu các điểm khác biệt của hai loại kiểu này? Hãy giải thích vì sao các loại kiểu này gây ra hiện tượng alias?
4. Hãy nêu điểm khác biệt chính của các cơ chế gọi chương trình con: đệ qui (recursive), biến cố (exception), trình cộng hành (coroutine), trình định thời (scheduled subroutine) và công tác (task) so với cơ chế Gọi-Trở về đơn giản (simple call-return)?
5. Hãy giải thích các phương pháp (lock-and-key, tombstone) tránh tham chiếu treo (dangling reference)? Nêu rõ cách truy xuất trong trường hợp bình thường, khi huỷ bỏ một đối tượng và cách phát hiện khi có tham chiếu treo.

6. Hãy thực hiện (viết các phương trình thể hiện các ràng buộc kiểu) suy diễn kiểu để suy ra kiểu của hàm sau:

```
1. H(x ,f, h) {
        if (f(x)) return h(x); else return f(x);
}
```

```
2. H(x ,f, h) {
        if (f(x)) return h(h(x)); else return f(x);
}
```

Biểu thức điều kiện của phát biểu if phải có kiểu boolean. Kiểu của biểu thức sau return phải cùng kiểu trả về của hàm.

7. Cho đoạn mã sau được viết trên ngôn ngữ Scala dùng qui tắc tầm vực tĩnh (static-scope rule). Nhắc lại, trên Scala, từ khoá var để khai báo biến, def để khai báo hàm, Int=>Int là kiểu hàm nhận vào 1 giá trị nguyên trả về 1 giá trị nguyên, giá trị của biểu thức cuối cùng trong một hàm là giá trị trả về của hàm.

```
def main = {
        var a = 0; var b = 1; var c = 4                                    //1
        def sub1(a:Int) = {                                                //2
                def sub2(a : Int, c : Int, f : Int => Int) = ( f(a) − f(c) ) ∗ 2   // 3
                def sub3(b:Int)  =  b ∗ c − a;                             // 4
                b = sub2(1 , 2, sub3)
        }
        sub1(3)
        print(b)                                                           // 5
}
```

a. Cho biết môi trường tham khảo tĩnh (static referencing environment) của các hàm main, sub1, sub2, sub3 (với các tên a, b và c phải ghi kèm // dòng khai báo của tên, ví dụ a//1).

b. Hãy vẽ các bản ghi hoạt động của các chương trình con còn đang tồn tại khi hàm main được gọi và chương trình thực thi đến dòng lệnh trong thân hàm sub3 // 4 lần thứ nhất? Trình bày tiếp sự thực thi cho đến khi in giá trị b của bản hoạt động main ở dòng // 5?

8. (LO.2) Cho một đoạn chương trình được viết trên một ngôn ngữ tựa C như sau:

```
int A[3] = {4 ,6 ,14};//
int j=0;
int n = 3;
int sumAndDecrease(int a, int i){
    int s = 0;
    for ( ; i<n ; i = i + 1 ){
        s = s + a;
        A[j] = A[j] - 1;
    }
    return s;
}
void main(){
    int s = sumAndDecrease(A[j], j);
    cout << s << A[0] << A[1] << A[2] ; //1
}
```

Hãy cho biết và giải thích kết quả in ra của chương trình trong các trường hợp sau:

a. nếu a và i được truyền bằng **trị-kết quả** .

b. nếu a và i được truyền bằng **tham khảo.**

c. nếu a và i được truyền bằng **tên**.

(Đề 2017)

9. Cho một cấu trúc dữ liệu được định nghĩa trên Scala như sau:

```
trait Element
case class Many(value:List[Element]) extends Element
case class Inte ( value : Int ) extends Element
case class Flt ( value : Float ) extends Element
```

và hàm `sum(lst:List[T],f:T=>Int):Int` trả về tổng các giá trị nguyên được ánh xạ từ các phần tử của danh sách lst qua hàm f. Ví dụ sử dụng hàm sum để tính tổng của một danh sách: `sum(List(1,3,4),(x:Int)=>x)` sẽ trả về 8.

Hãy viết lệnh gọi hàm sum trên để đếm số phần tử Inte trực tiếp (phần tử của danh sách) và gián tiếp (đệ qui trong các phần tử của danh sách) trong một danh sách các Element? Ví dụ:

```
val lst = List(Many(List(Inte(3),Many(List(Flt(2.1),Inte(5))),Flt(1.0),Inte(2))),Inte(1))
```

và `sum(lst,...)` sẽ trả về 4, với ... là nội dung cần phải thực hiện cho câu này.

10. Hãy nêu các đặc điểm chính của cơ chế Gọi - Trở về đơn giản? Đối với mỗi đặc điểm, hãy cho biết cơ chế gọi chương trình con nào khác biệt với cơ chế này?

11. Hãy viết lại biểu thức ở dạng trung tố (infix) sau sang dạng biểu thức tiền tố (prefix) Cambridge Polish:

a-b*c*d-e+f

Biết rằng độ ưu tiên và tính kết hợp của các phép toán trong biểu thức như thông thường (đều kết hợp trái và * có ưu tiên cao hơn +, -). Trong biểu thức tiền tố nhiều toán hạng, thứ tự tính toán cũng từ trái sang phải.

Yêu cầu: Biểu thức dạng tiền tố Cambridge Polish phải thỏa các yêu cầu sau:

- Thứ tự xuất hiện các toán hạng trong biểu thức dạng tiền tố phải có cùng thứ tự xuất hiện như trong biểu thức trung tố.
- Số dấu (và) là ít nhất.
- Có cùng thứ tự tính toán các phép toán với thứ tự đó trong biểu thức dạng trung tố.

12. Giải thích đặc điểm của kiểu Union ? Cho một ví dụ sử dụng kiểu Union và giải thích vì sao phải dùng kiểu Union trong ví dụ của bạn.

* Bonus :

13. Chuyển đổi số 44.45 sang dạng IEEE-754 16bit với 7 bit mũ

14. Giả sử có một struct như sau :

```
struct MyStruct {
    char a;
    int b;
    char c;
    float d;
    double e;
    float f;
}
```

Biết rằng kiểu char, int, float, double lần lượt có kích thước là 1,2,4,8, tìm kích thước của MyStruct và sắp xếp lại thứ tự khai báo biến để kích thước MyStruct là nhỏ nhất.

B. Phần giải

I. Phần câu hỏi lập trình

1. Vòng lặp:

```
def visitDowhile(ast:Dowhile, o:Context):
        code = ""
        frame = o.frame
        frame.enterLoop()

        bodyLabel = frame.getNewLabel()
        continueLabel = frame.getContinueLabel()
        breakLabel = frame.getBreakLabel()

        code += self.emit.emitLABEL(bodyLabel, frame)
        for stmt in ast.sl:
                code += self.visit(stmt, o)

        code += self.emit.emitLABEL(continueLabel, frame)
        exp, expType = self.emit.visit(ast.exp, Access(o.frame, o.sym, False, True))
        code += exp
        code += self.emit.emitIFTRUE(bodyLabel, frame)
        code += self.emit.emitLABEL(breakLabel, frame)
        frame.exitLoop()
        return code
```

```
def visitFor(ast:For, o:Context):
        code = ""
        frame = o.frame
        frame.enterLoop()
        bodyLabel = frame.getNewLabel()
        conditionLabel = frame.getNewLabel()
        continueLabel = frame.getContinueLabel()
        breakLabel = frame.getBreakLabel()

        exp1, exp1Type = self.visit(ast.expr1, Access(o.frame, o.sym, False, True))
        code += exp1
        code += self.emit.emitGOTO(conditionLabel, frame)

        code += self.emit.emitLABEL(bodyLabel, frame)
        code += self.visit(ast.loop, o)

        code += self.emit.emitLABEL(continueLabel, frame)
        exp3, exp3Type = self.visit(ast.expr3, Access(o.frame, o.sym, False, True))
        code += exp3

        code += self.emit.emitLABEL(conditionLabel, frame)
        exp2, exp2Type += self.visit(ast.expr2, Access(o.frame, o.sym, False, True))
        code += exp2
        code += self.emit.emitIFTRUE(bodyLabel,frame)
        code += self.emit.emitLABEL(breakLabel,frame)
```

```
        frame.exitLoop()
        return code
```

```
def visitContinue(ast:Continue, o:Context):
        frame = o.frame
        return self.emit.emitGOTO(frame.getContinueLabel(), frame)
```

```
def visitBreak(ast:Break, o:Context):
        frame = o.frame
        return self.emit.emitGOTO(frame.getBreakLabel(), frame)
```

* Giải thích continue và break khi nằm sâu bên trong for, do while vẫn có thể chuyển điều khiển đến vị trí thích hợp : Khi bắt đầu For hay Do while, ta đều bắt đầu bằng lệnh gọi Frame.enterLoop(), khi đó trong Frame sẽ sinh ra 2 label mới là continueLabel và breakLabel và đặt chúng vào continueStack cũng như breakStack. Ta đưa các label đó vào trong mã lệnh thực thi của chúng ta bằng lệnh gọi Frame.getContinueLabel và Frame.getBreakLabel, khi đó sẽ trả về được 2 label đó. Trong thân vòng lặp của chúng ta, mỗi khi chúng ta gọi continue hay break tức chúng ta sẽ emitGOTO(frame.getContinueLabel()) (tương tự cho break) lúc đó điều khiển sẽ được chuyển đến mã lệnh thực thi của label continue và break ta khai báo. Khi ra khỏi vòng lặp, ta sử dụng lệnh Frame.exitLoop() để xoá 2 nhãn continue và break ra khỏi stack của chúng.

- Link video : https://youtu.be/oJV7_75tahY?t=569

2. Toán tử:

```
def visitBinaryOp(ast:BinaryOp, o:Context):
        op = ast.op
        frame = o.frame
        sym = o.sym
        expStr = ""
        expType = None
        leftCode, leftType = self.visit(ast.left, Access(frame, sym, False, True))
        rightCode, rightType = self.visit(ast.right, Access(frame, sym, False, True))
        if type(leftType) != type(rightType):
                if type(leftType) is FloatType:
                        leftCode += self.emit.emitI2F(frame)
                        leftType = FloatType()
                else:
                        rightCode += self.emit.emitI2F(frame)
                        rightType = FloatType()
        if op in ['+','/']:
                expStr = leftCode + rightCode
                if op == '+'
                        expStr += self.emit.emitADDOP(op, leftType, frame)
                else:
                        expStr += self.emit.emitMULOP(op, leftType, frame)
```

```
                expType = leftType
        else:
                isAnd = op == "&&"
                expStr = leftCode
                label1 = frame.getNewLabel()
                label2 = frame.getNewLabel()
                expStr += self.emit.emitIFFALSE(label1, frame) if isAnd else
self.emit.emitIFTRUE(label1, frame)
                expStr += right
                expStr += self.emit.emitGOTO(label2, frame)
                expStr += self.emit.emitLABEL(label1, frame)
                expStr += self.emit.PUSHICONST(0 if isAnd else 1, frame)
                expStr += self.emit.emitLABEL(label2, frame)
                expStr = leftType
        return expStr, expType
```

II. Phần câu hỏi tự luận

3.

* Kiểu dữ liệu con trỏ :

- Là kiểu dữ liệu có miền giá trị bao gồm các địa chỉ của bộ nhớ và một giá trị đặc biệt là nil (con trỏ không trỏ đến vùng nhớ nào cả)
- Giúp truy xuất vùng nhớ gián tiếp, giúp quản lý bộ nhớ động cho phép lưu trữ trong vùng nhớ heap.
- Có 2 toán tử là phép gán và phép truy xuất.
    - Phép gán để gán giá trị của con trỏ đến vùng nhớ hữu ích.
    - Phép truy xuất lấy giá trị được lưu trữ tại vùng nhớ con trỏ trỏ đến.
- Ví dụ trong C:

  ```
  int *p, *q;
  p = q;
  *p = 5;
  *q = *p;
  ```

* Kiểu dữ liệu tham chiếu:

- Kiểu tham chiếu cho phép tham chiếu đến đối tượng hoặc giá trị.
- Sau khi khởi tạo, biến tham chiếu sử dụng giống hoàn toàn như biến nó tham chiếu tới.
- Không hỗ trợ thêm toán tử.
- Ví dụ:

  ```
  int b;
  int &a = b;
  ```

a = 5;
b++;
printf(“%i”,a) //6

- So sánh kiểu con trỏ và kiểu tham chiếu:

Kiểu con trỏ	Kiểu tham chiếu
Chứa địa chỉ của một biến	Là bí danh (alias) của một biến
Sử dụng thông qua toán tử * và =	Sử dụng trực tiếp như biến mà nó tham chiếu
Có thể thay đổi ô nhớ nó trỏ đến	Không thể thay đổi biến tham chiếu
Có thể trỏ về null	Không thể null
Có thể không có khởi tạo ban đầu	Phải có khởi tạo ban đầu

- Dựa vào việc truy xuất đến ô nhớ của con trỏ hay tham chiếu của biến tham chiếu có thể tạo ra hiện tượng alias.

4.

➢ Đệ quy : gọi lại gián tiếp hay trực tiếp, giống như việc gọi hàm đệ quy
➢ Biến cố : không có lệnh gọi tường minh, được gọi khi có sự kiện hoặc ngoại lệ xảy ra.
➢ Trình cọng hành : có nhiều điểm gọi hàm (entry point), cho phép trì hoãn việc thực thi và chuyển giao điều khiển lại cho chương trình gọi nó (caller). Việc thực thi của nó sau này sẽ được tiếp tục tại điểm mà nó trì hoãn.
➢ Công tác : thực thi đồng thời với những task khác, thực thi trên máy đa xử lý hoặc có tích hợp cơ chế time sharing.
➢ Trình định thời : có sự trì hoãn trong việc chuyển giao điều khiển cho chương trình con:
    ○ Theo thời gian: gọi sau một thời gian
    ○ Theo độ ưu tiên: gọi khi nào các chương trình có độ ưu tiên cao hơn hoàn thành.

5.

➢ Phương pháp lock-and-key: sử dụng cặp khoá-địa chỉ(key-address). Biến vùng nhớ động được đại diện bởi vùng nhớ kèm theo với nó là giá trị khoá.

- Khi một vùng nhớ được khởi tạo, giá trị khoá sẽ được đặt vào cả biến vùng nhớ động cũng như con trỏ.
- Các truy xuất vào vùng nhớ sẽ so sánh 2 giá trị khoá đó, và sẽ cho phép khi và chỉ khi chúng trùng nhau.
- Khi huỷ cấp phát, giá trị lock trên biến vùng nhớ sẽ bị thay đổi, sẽ khiến cho tất cả các key trên con trỏ khác sẽ không còn phù hợp và sẽ được báo lỗi.

➢ Phương pháp tombstones : tạo thêm một ô nhớ để lưu địa chỉ của vùng nhớ được cấp phát. Con trỏ sẽ trỏ vào tombstone

- Khi truy xuất vùng nhớ, từ pointer sẽ truy xuất tombstone, từ tombstone sẽ truy xuất giá trị của vùng nhớ
- Khi huỷ cấp phát, giá trị tombstone sẽ được gán thành Null
- Phát hiện tham chiếu treo khi tombstone là Null

6.

➢ Câu 1

H là 1 hàm có 3 tham số : H : T1 x T2 x T3 -> T4 (1) x : T1 f : T2 h : T3
f(x) : f là 1 hàm có một tham số: f : T5 -> T6 x truyền vào cho f => T1 ≡ T5 => f : T1 -> T6 (2)
if (f(x)) : biểu thức điều kiện của if phải có kiểu boolean: => T6 ≡ boolean (3)
Từ (2) và (3) => f : T1 -> boolean (4)
h(x) : h là 1 hàm có một tham số : h : T7 -> T8 x truyền vào cho h => h: T1 -> T8 (5) Vì h(x) và f(x) là 2 vế của mệnh đề trả về if nên chúng cùng kiểu trả về

Từ (4) và (5) => h : T1 -> boolean (6)
Từ (1), (4) => T2 ≡ T1 -> boolean (7)
Từ (1), (6) => T3 ≡ T1 -> boolean (8)
Kiểu trả về của hàm H chính là kiểu trả về của hàm if => T4 ≡ boolean (9)
Từ (1), (6), (7), (8), (9) => H : (T1 x (T1 -> boolean) x (T1 -> boolean)) -> boolean

➢ Câu 2 giữ nguyên 4 ý đầu của câu 1

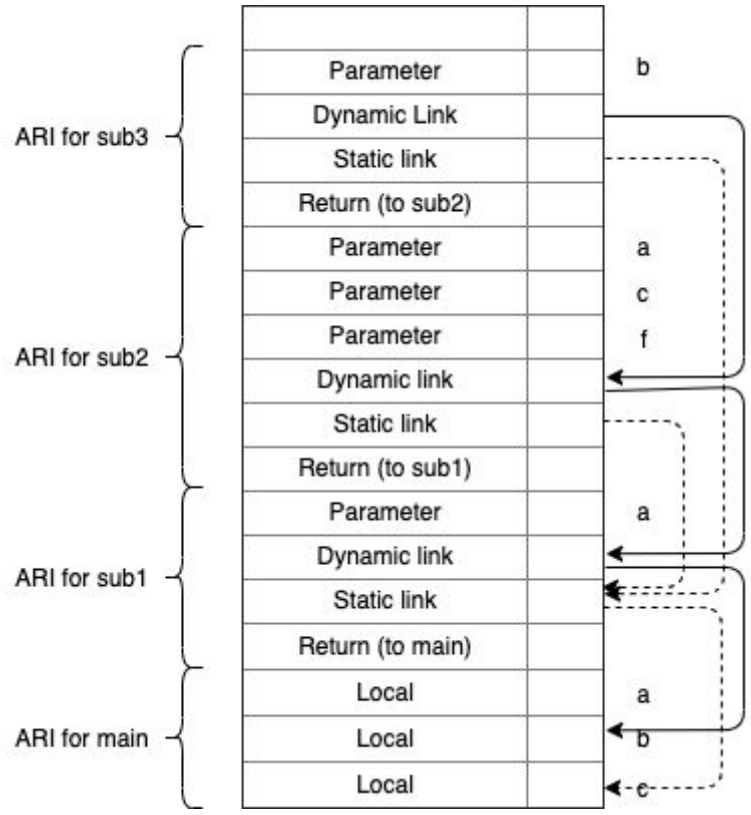
h(x) : h là 1 hàm có một tham số : h : T7 -> T8 x truyền vào cho h => T1 ≡ T7 => h: T1 -> T8 (5) h(h(x)) : giá trị trả về của hàm h truyền ngược lại vào cho nó => T7 ≡ T8 (5*) vậy h : T8 -> T8 (6) Từ (5) và (5*) => T1 ≡ T8 (7) Vì h(x) và f(x) là 2 vế của mệnh đề trả về if nên chúng cùng kiểu trả về => T8 ≡ boolean (8)
Từ (6), (7) và (8) ta có: T1 ≡ boolean (9) h: boolean -> boolean (10)
Từ (1), (4), (9) => T2 ≡ boolean -> boolean (11)
Từ (1), (10) => T3 ≡ boolean -> boolean (12)
Kiểu trả về của hàm H chính là kiểu trả về của hàm if => T4 ≡ boolean (13)
Từ (1), (9), (11), (12), (13) suy ra H : (boolean x (boolean -> boolean) x (boolean -> boolean)) -> boolean

7.

a)

Name	Referencing environment
main	main, sub1, a//1, b//1, c//1
sub1	main, sub1, sub2, sub3, a//2, b//1, c//1
sub2	main, sub1, sub2, sub3, a//3, b//1,c//3,f//3
sub3	main, sub1, sub2, sub3, a//2, b//4, c//1

b)



Sau lệnh gọi f(a) tức sub3(a) đầu tiên, trong hàm sub2 tiếp tục thực thi lệnh f(c) tức sub3(c), sau đó tính toán giá trị (f(a)-f(c))*2 và trả về kết quả cho hàm sub1. Hàm sub1 sau khi nhận được giá trị từ việc gọi sub2 sẽ gán vào cho b và trả điều khiển lại cho hàm main, hàm main sau đó sẽ thực thi tiếp tục đến lệnh print(b) và in ra giá trị của b tính được từ việc gọi sub1 ở trên.

8.

a. Nếu a và i được truyền bằng **trị - kết quả** :

- Lời gọi hàm sumAndDescrease sẽ truyền tương ứng giá trị A[0] và j vào cho 2 biến a và i, là 4 và 0.
- Kết thúc hàm sumAndDescrease ta được giá trị s trả về là 12, giá trị biến a không đổi, giá trị biến i bằng 3, dãy A ban đầu thành {1,6,14}

- Sau đó giá trị của a sẽ được gán lại cho A[0], giá trị của i gán lại cho j. Khi đó dãy A là {4,6,14} và j bằng 3
- Kết quả in ra là 12 4 6 14

b. Nếu a và i được truyền bằng tham khảo:

- Lời gọi hàm sumAndDescrease sẽ truyền tương ứng ô nhớ của A[0] và j vào cho 2 biến a và i, tức bây giờ các truy xuất tới a và i sẽ ảnh hưởng trực tiếp đến A[0] và j.
- Vào trong thân vòng lặp, vì truyền theo kiểu tham khảo nên kết quả sau vòng lặp là : s = 10, i = j = 3, a = 3, A = {3, 5, 13}
- Kết quả in ra là 10 3 5 13

c. Nếu a và i được truyền bằng tên:

- Lời gọi hàm sumAndDescrease sẽ truyền thay thế tên biến a thành A[j] và tên biến i thành j.
- Vào trong thân vòng lặp, vì truyền theo kiểu tên nên kết quả sau vòng lặp là : s = 24, j = 3 , A = {3, 5, 13}
- Kết quả in ra là 24 3 5 13

(2017)

9.

```
def func(e : Element): Int = {
  e match {
     case _: Inte => 1
     case _: Flt => 0
     case _:Many => sum(e.asInstanceOf[Many].value, func)
  }
}

sum(lst, (e:Element) => func(e))
```

* Bonus hàm sum để test:

```
def sum[T](lst:List[T],f:T=>Int): Int = {
  var i=0
  var alto = 0
  while (i < lst.length) {
     alto += f(lst(i))
     i += 1
  }
  alto
```

}

10.

* Các đặc điểm chính của chương trình gọi-trả về đơn giản :
  - Không đệ quy (khác với chương trình đệ quy)
  - Chỉ có một điểm đầu (khác với cơ chế)
  - Có lệnh gọi tường minh (khác với cơ chế biến cố-ngoại lệ)
  - Chuyển điều khiển ngay lập tức (khác với trình định thời)

11.

* Vẽ cây :

(+(-a(*bcd)e)f)

12.

➢ Union là kiểu dữ liệu đặc biệt giúp ta có thể định nghĩa lưu trữ nhiều loại dữ liệu khác nhau vào cùng một vùng nhớ. Ta có thể định nghĩa kiểu Union cho nhiều kiểu dữ liệu nhưng tại một thời điểm chỉ có một loại dữ liệu được lưu trữ xuống vùng nhớ.

➢ Union giúp cho việc tối ưu hoá bộ nhớ vì một vùng nhớ có thể sử dụng cho nhiều mục đích, ngoài ra nó còn hỗ trợ giả đa hình trong các ngôn ngữ lập trình không hỗ trợ hướng đối tượng.

➢ Ví dụ :

```
struct Node {
        struct Node * next;
        union Num{int i; float f;char str[5];} data;
}
```

Ta có thể định nghĩa nhiều cấu trúc dữ liệu cho một node như int, float hay array của char mà không cần thêm biến và bộ nhớ sẽ được set động theo từng loại dữ liệu giúp cho việc lưu trữ sé tối ưu hơn.

13.

bias = $2^{7-1} - 1 = 63$ (với 7 là số bit mũ đề cho, nếu đề cho số bit fraction là độ chính xác thì số bit mũ = số bit tổng - số bit fraction -1)

44	22	11	5	2	1

22	11	5	2	1	0
0	0	1	1	0	1

<<<----------

=> 44 = 101100_2

0.45	0.9	0.8	0.6	0.2	0.4	0.8
0.9	1.8	1.6	1.2	0.4	0.8	1.6
0	1	1	1	0	0	1

Phần tô xanh là phần lặp lại.

=> 44.45 = 101100.01110011001100… x 2^0

= 1.01100011 10011001100… x 2^5

- số 16 bit, 7 bit mũ => base = 16-7-1 = 8 bit, phần tô xanh ở trên chính là giá trị fraction, lấy sau dấu chấm đếm đủ 8 bit số.

➢ Kết quả :

- Giá trị bit dấu : 0 (vì số dương)
- Giá trị bit mũ = bias + 5 = 68 = 1000100
- Giá trị bit fraction = 01100011

0	1	0	0	0	1	0	0	0	1	1	0	0	0	1	1
S	bit mũ							bit fraction							

14.

➢ 1 + 1p + 2 + 1 + 3p +4+ 4p +8 +4 =32

➢ 8+4+4+2+1+1 => 24 min