Polymorphism - A name is bound to more than one entity.
Alias - Many names are bound to one entity.

Reference Type	Pointer
`int A;` `int& rA = A;`	`int A;` `int* pA = &A;`
rA ⇒ A	*pA ⇒ A
N/A	pA++
cannot reseated	pA = &B
cannot be null	pA = null
cannot be uninitialized	int* pA

Pointers refers to an address
References refers to object or value

Tombstone: extra heap cell that is a pointer to the heap-dynamic variable
The actual pointer variable points only at tombstones
When heap-dynamic variable de-allocated, tombstone remains but set to nil
Costly in time and space

Locks-and-keys: Pointer values are represented as (key, address) pairs
Heap-dynamic variables are represented as variable plus cell for integer lock value
When heap-dynamic variable allocated, lock value is created and placed in lock cell and key cell of pointer

Dangling pointers (dangerous)
A pointer points to a heap-dynamic variable that has been de-allocated

Question 1

Given the following infix expression: **(a + b + c) * d - e * f * g** and the precedence and associativity of the operators in the expression are like those in C. Transverse it into the Polish prefix format?

- ○ - + + a b c * d * * e f g
- ○ - * + + a b c d * e * f g
- ◉ - * + + a b c d * * e f g
- ○ - * + a + b c d * f * g h

Yes, you are correct

Question 3

Assume that the precedence and associativity of operators in the following infix expression are like those in C: a * b * (c - e - f) * g. Rewrite the expression in other notations while keeping the appearance order of operands. No space is allowed.

The Polish prefix notation of the above expression is ***ab--cefg

The Cambridge Polish prefix notation of the above expression is (*ab(-cef)g)

The Polish postfix of the above expression is ab*ce-f-*g*

Restart | Show Answers

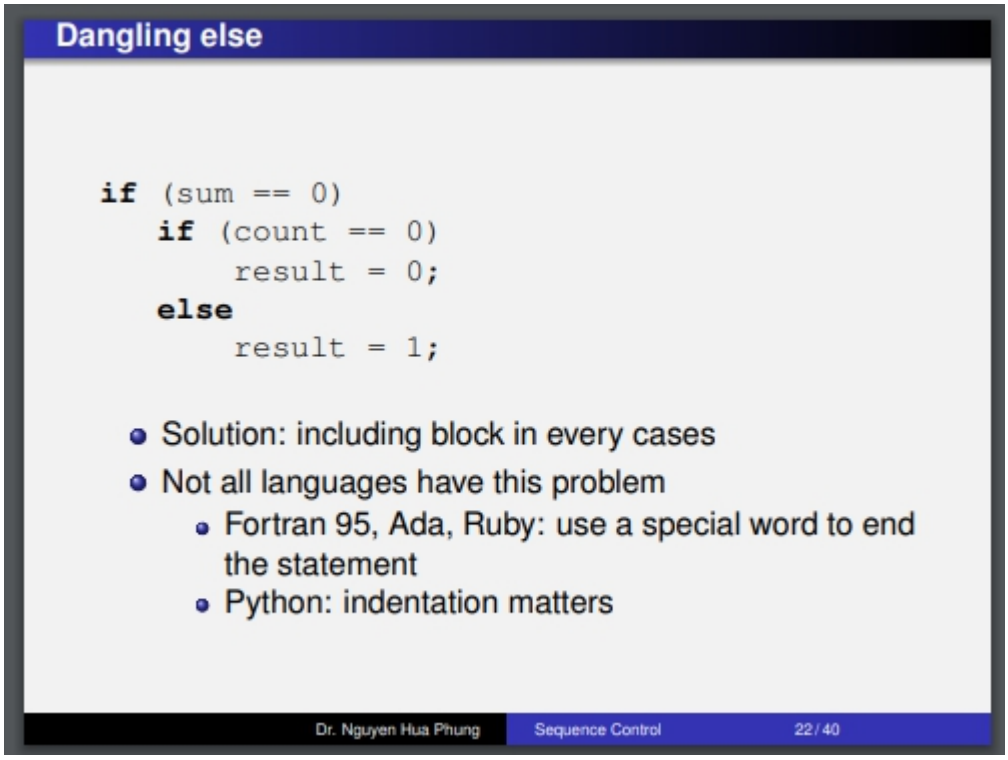
Your score is 3/3.

Given the following expression in Polish prefix notation: * * a + b c - d * e f

With the minimum of ( ), the same appearance order of operands and no space, the equivalent expression in infix notation is a*(b+c)*(d-e*f)

Check | Show Feedback

Your score is 1/1.



Simple Call-Return:

No recursion # recursion
Explicit Call Site (tới cái lệnh gọi này thì gọi chương trình con) #Exception
Single Entry Point (gọi chương trình con thì tới đúng 1 điểm đó) # Coroutines
Immediate Control Pasing (khi a gọi b thì chương trình a ngưng để chuyển sang b) # Schedule
Single Execution (giống trên) # Tasks

Recursion:
+ Khác nhau :

Có 2 dạng:
Direct Recursive call (trực tiếp) - gọi lại chính nó
Inderect Recursive call (gián tiếp) - ví dụ a gọi b gọi c xong c mới gọi lại a

Exception Processing Handler (biến cố và xử lý biến cố):
+ Khác nhau:

Không có Explicit Call Site
Used in:
Event-Driven Programming (lập trình hướng đến sự kiện)
Error Handling(xử lý biến cố)

A language must specified:
Which exceptions can be handle and How can they be defined
How an exception can be raised
How an exception can be handled

Coroutines:
+Khác nhau:

Many Entry Point

Tasks:
+Khác nhau:

Able to execute concurrently with other tasks (đồng thời chạy nhiều tasks)
Run on multi processor machine
Run on single processor machine using time sharing

Scheduled:
+Khác nhau:
The execution not started when it is invoked (Ko chạy liền khi được gọi)
Scheduled by time (theo thời gian)
Scheduled by priority (theo độ ưu tiên)

Input-Output Parameter:
Pass by value result (xong rồi mới trả lại kết quả)
Pass by reference (vd: int &a) như allias
Pass by name

Input Parameter:
Pass by value
Pass by constant reference (vd: const int &a,...)

Output Parameter:
Pass by result
As a result of a function

A function is high-order when it accept functions:
As its input parameters (fairly common)
As its out parameters (required in functional programming)

Non-local environment:
Deep binding (vùng sâu)
Shallow binding (vùng cạn)

Static Scope

Example, Static scope: z = 6

```
int x = 1;
int f(int y){ return x+y; }

int g (int h(int b)){
    int x = 2;
    return h(3) + x; //shallow binding
}
...
{int x = 4;
 int z = g(f); //deep binding
}
```

Dynamic Scope + Deep Binding

Example, Dynamic scope + Deep binding: z =

```
int x = 1;
int f(int y){ return x+y; }

int g (int h(int b)){
    int x = 2;
    return h(3) + x; //shallow binding
}
...
{int x = 4;
 int z = g(f); //deep binding
}
```

Skip Scala Example

Dynamic Scope + Shallow binding

What is non-local environment?

- Deep binding
- Shallow binding

Example, Dynamic scope + Shallow binding: z = 7

```
int x = 1;
int f(int y){ return x+y; }

int g (int h(int b)){
    int x = 2;
    return h(3) + x; //shallow binding
}
...
{int x = 4;
 int z = g(f); //deep binding
}
```

Skip Scala Example