Introduction to Programming Languages,Anthony A. Aaby

Chapter 1: introduction

Marjan Albouye 9131060

2. Construct a trace of the execution of the following program (i.e. complete the following proof).

1. parentOf(john, mary)	fact
2. parentOf(kay, john)	fact
3. parentOf(bill, kay)	fact
4. ancestorOf(X,Y) if parentOf(X,Y)	Rule
5. ancestorOf(X,Z) if parentOf(X,Y) and ancestorOf(Y,Z)	Rule
6. not ancestorOf(bill,mary)	Assumption
7.not(parentOf(bill,Kay) and ancestorOf(Kay,mary))	from 5 and modus tollens
8.not parentOf(bill,Kay) or not ancestorOf(Kay,mary)	
9.not ancestorOf(kay,mary)	
10.not(parentOf(Kay,john) and ancestorOf(john,mary))	
11.not parentOf(Kay,john) or not ancestorOf(john,mary)	
12.not ancestorOf(john,mary)	
13.not parentOf(john,mary)	from 4 and modus tollens
14.contradiction	

****

3. Construct a trace of the execution of fac(4)given the function definition

```
fac(N) = if N = 0 then 1
         else N*fac(N-1)
```

```
fac(4)= 4*fac (4-1)

      = 4*fac(3)

      = 4*3*fac(2)

      = 4*3*2*fac(1)

      = 4*3*2*1=24
```

****

5. Using the following definition of a list,

list([ ]) -- the empty list

list([X|L]) if list(L) -- first element is X the rest of the list is L

[X0,...Xn] is an abbreviation for [X0|[...[Xn|[ ]]...]

complete the following computation (proof) and determine the result of concatenating the two lists.

1. concat([],L,L)	fact
2. concat([x|L0],L1,[x|L2]) if concat(L0,L1,L2)	fact
3. ~concat([0,1],[a,b],L)	Assumption
4. L0 = [0,1]	Unification
5. L1 = [a,b]	Unification
6. concat([],[a,b] ,[a,b])	from 1,5
7. concat([0],[a,b],[a,b,0])	x=0 and modus pones
8. concat([0,1],[a,b],[a,b,0,1])	x=0 and modus pones
9. ~concat([0,1],[a,b],L)	
10. contradiction	8,9 L=[a,b,0,1]

****

6. Classify the following languages in terms of a computational model: Ada, APL, BASIC, C, COBOL, FORTRAN, Haskell, Icon, LISP, Pascal, Prolog, SNOBOL.

Imperative: Ada, BASIC, C, COBOL, FORTRAN, Pascal, Icon

Logic : prolog,SNOBOL(partial logic)

Functional: Haskell, LISP, APL, SNOBOL (partial functional)

****

8. Compare the syntactical form of the if-command/expression as found in Ada, APL, BASIC, C, COBOL, FORTRAN, Haskell, Icon, LISP, Pascal, Prolog, SNOBOL.

Ada:

```
if <condition> then
  ……
else
  ……
 end if;
```

APL:

```
:If <boolean expression>
...
[:ElseIf <boolean expression>]
...
[:Else]
...
:EndIf
```

BASIC:

```
If condition [ Then ]
   [ statements ]
[ ElseIf elseifcondition [ Then ]
   [ elseifstatements ] ]
[ Else
   [ elsestatements ] ]
End If
```

C:

```
if (condition true ) {
   [statement]
}
else {
   [statement]
}
```

COBOL:

```
IF <condition>
     statement
ELSE
      Statement
END-IF
```

Fortran:

```
IF (logical-expression-1) THEN
  statements-1
ELSE IF (logical-expression-2) THEN
  statements-2
ELSE IF (logical-expression-3) THEN
  statement-3
ELSE IF (.....) THEN
  ...........
ELSE
  statements-ELSE
```

```
END IF
```

Haskell:

Example: (using guards)

```
describeLetter :: Char -> String
describeLetter c
   | c >= 'a' && c <= 'z' = "Lower case"
   | c >= 'A' && c <= 'Z' = "Upper case"
   | otherwise            = "Not an ASCII letter"
```

Icon:

```
if a := read() then write(a)
```

in Icon control structures are based on the "success" or "failure" of expressions, rather than on boolean logic.

LISP:

```
(if (< 2 3)
  (... true ...)
  (... false ...)) → works like else
```

Pascal:

```
if condition then S1 else S2;
```

prolog:

```
( condition -> then_clause ; else_clause )
```

****

11. What languages provide for binding of type to a variable at run-time?

A variable binding represents either a field of a class or interface, or a local variable declaration (including formal parameters, local variables, and exception variables).

the time & manner in which variables are bound to attributes is key to the behavior/implementation of the language

static binding: prior to run-time, remains fixed usually more efficient

dynamic binding: occurs or can change during run-time usually more flexible.

It means in dynamic binding there is no declaration before run time. A variable is bound to a type at run time and take the type of the value in assignment statement.

Languages providing type binding at run-time:
JavaScript, APL, SNOBOL4

****

13. Compare two programming languages from the same computational paradigm with respect to the programming language design principles.

For example : C++ and Java are both Imperative

-Principles of computing model:

both are universal.They are definitely Object Oriented.

-Principles of Implementation:

C++ doesn’t have Parallel Process Implementation, The parallel processing facility shall be designed to minimize execution time and space. Processes shall have consistent semantics whether implemented on multicomputers, multiprocessors, or with interleaved execution on a single processor.Java leave some semantics open to permit efficient implementation on different operating systems.but c++ doesn’t.Java is better in time and space.

-Principles of Simplicity:

C++ has C operations and its own operations (new/delete vs. malloc/free, cout vs. printf).

java is better in this feature but C++ mostly has this too.

-Principles of Extensibility:

C/C++ preprocessor can be used to create some (obscure) syntactic forms. A preprocessor (such as cpp or m4) can be used with any language, including Ada and Java, but neither include a preprocessor in their definition.C++ is not extensible but java is.

-Principles of Regularity:

Java is better than c++ in this feature.

****

14. Construct a program in your favorite language to do one of the following:
a. Perform numerical integration where the function is passed as a parameter.
b. Perform sorting where the the less-than function is passed as a parameter.

sort(Xs) :=Fold(print(len(less-than(Xj))),Xj)