FIFTH SEMESTER B.E(CSE)
Makeup exam Scheme
Principles Of Programming Languages
January-2009

TIME : 3 HOUR MAX. MARKS : 50

1a. Each answer carries 1 M 1X3=3M

(1) The value of a variable is dynamic since it can change during execution.

(2) The data type of a variable cannot change during execution; it is fixed during translation by its declaration.

(3) The name of a variable is also fixed by its declaration and cannot change during execution, except in

the case of a dynamically allocated memory location without a name of its own, which can be accessed using different pointer variable names at different times, as for example in

```
int* p;
int* q;
p = (int*) malloc(sizeof(int));
*p = 2;
q = p;
*q = 1;
```

After the assignment **q = p**, the same (dynamically allocated) variable can be accessed using the names ***p** and ***q**. (An alternative view would say that the variable *accessed* using the *operations* ***p** and ***q** has in fact no name of its own, so we cannot say that its name has changed.)

1b

The question really is one of whether a goto-statement exists and can be used to override the structuring of the if-statement, as in

```
main()
{ goto a;
if (2 < 1)
a: printf("ack!\n");
else printf("ok.\n");
return 0;
}
```

1M

In Java no goto is available, so this is impossible. In Standard Pascal and Ada, it is also impossible, since a goto cannot jump inside a structured construct. In C and FORTRAN (permissive languages both), this is permitted. 1M

1c.

In Java :This is a non orthogonality, since it is an interaction between assignment and the data types of the subjects of the assignment. It definitely cannot be viewed as a non generality, since it makes no sense for assignment to be so general as to apply to all cases (assignment should only apply when data types are comparable in some sense). It also can't be labeled a non uniformity, since we are not comparing two different constructs. 1M

In C : This is a security issue, since assignment of a real (double or float) to an integer results in automatic truncation, which could result in incorrect execution. 1M

1d. Full explaination with example carries 3 Marks

C has the same problems with semicolons as C++ — indeed, C++ inherited them from C. Thus, in C, we must always write a semicolon after a **struct** declaration:

struct X { int a; double b; } ; /* semicolon required here */

but never after a function declaration:

int f(int x) { return x + 1; } /* no semicolon */

The reason is C's original definition allowed variables to be declared in the same declaration as types :

struct X { int a; double b; } x;
/* x is a variable of type struct X */

In addition to this nonuniformity of semicolon usage, C (and C++) have at least one additional such nonuniformity: semicolons are used as *separators* inside a for-loop specifier, rather than as terminators:

for (i = 0; i < n; i++ /* no semicolon here! */)

2a. Regular Expression 1X 4=4M

- **name**: [a-zA-Z]*
- **surname**: [a-zA-Z']+
- **email**: [a-zA-Z0-9_\.]+@[a-zA-Z0-9-]+\.[a-zA-Z]{0,4}
- **phone number**: [0-9]+\-[0-9]+

2b 1X6= 6M

```
<postal-address> ::= <name-part> <street-address> <zip-part>

      <name-part> ::= <personal-part> <last-name> <opt-jr-part> <EOL>
                    | <personal-part> <name-part>

  <personal-part> ::= <first-name> | <initial> "."

 <street-address> ::= <opt-apt-num> <house-num> <street-name> <EOL>

       <zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>

    <opt-jr-part> ::= "Sr." | "Jr." | <roman-numeral> | ""
```

Each carries 1 Mark . Any five can be given.
Parse Tree :1M

3a. 6 X 0.5=3M

(i) Not an l-value: + computes the integer sum of (the rvalue of) **x** and 2.
(ii) Not an l-value: **&** returns a pointer (r-value) indicating the address of **x**.
(iii) An l-value equivalent to **x** itself.
(iv) Since **&x** is a pointer, the + computes the pointer (r-value) that points two locations past the location of **x** (scaled by the size of **x**).
(v) An l-value: the dereference unary operator * in C, when applied to a pointer, always produces an l-value.

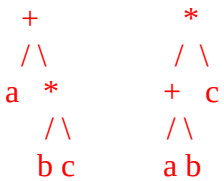
(vi) Not an l-value, but the (pointer) r-value contained in **y**. (Thus, ***&x** is equivalent to **x**, but **&*y** is *not* equivalent to **y**.)

3b. Automatic initialization to int and truncation of fractional part. 2M

3c. Full derivation 3M

S → ID = E
a = E
a = E * E
a = ID * E
a = b * E
a = b * (E)
a = b * (E + E)
a = b * (ID + E)
a = b * (c + E)
a = b * (c + ID)
a = b * (c + a)

Yes the it is ambiguous. An expression such as: a + b * c can generate two different parse trees:



2M

4a. 1 X 3= 3M

i) scope :The textual region of a program where a binding is active
ii)static scoping – scope defined in terms of physical (lexical) structure of the program.
iii)dynamic scoping – bindings depend on flow of execution at runtime.

4b. Definition 1M Example :1 M
An alias is an extra name for something that already has a name in the current scope. “Two names for the same object.”

4c. The basic ways through which a software component can be modified for reuse.
i)Extension ii) restriction iii)redefinition iv)abstraction

Listing each name :0.25 M X 4 =1 M
Explanation :1M 1 X4=4M

5a.

```
c => ( 6 14 27 )            1M
(car (cdr c)) => 14         1M
```

5b. 3M

```
(nth '(4 5 6) 1) => 4
(define (nth alist n)
(if (= 1 n) (car alist)
(nth (cdr alist) (- n 1) ) ) )
```

c)Write the syntactic specification or signature for abstract data type bstree(binary search tree) 5M

create:bstree
make:bstree x element x bstree →bstree
empty:bstree→boolean
left:bstree→bstree
right:bstree→bstree
data:bstree→element
isin:bstree x element→boolean
insert:bstree x element →bstree

At least 5 each 1M X 5 =5M

6. a) What is the difference between normal-order and applicative order evaluation? Give example for each
1M Definition for each 1 X 2 =2M
Example 0.5 for each 0.5 x 2=1M
Normal Order – evaluate arguments only when needed (sometimes this means you never have to evaluate them)
Applicative Order – evaluate arguments before passing them to function

6b)

1 X 5=5M

- i)It is raining **and** I am indoors. *P* & *Q*

- ii) **If** it is raining **then** I am indoors. $P \longrightarrow Q$

- iii) It is raining **if** I am indoors. $Q \longrightarrow P$

- iv) It is raining **if and only if** I am indoors. $P \longleftrightarrow Q$

- v) It is **not** raining. $\neg P$

6c) What does it mean for something to be a first class variable in a programming language? 2M

A first variable citizen can be passed as a parameter, returned from a subroutine, or assigned into a variable. Integers are first class in most programming languages. Functions are first class in functional languages