Final by _____
COP 4020 15 Dec 89

Each problem is worth 10 points. Budget your time carefully.

1 Terminology: Match the letter of the phrase (below) that best applies to the following terms:

___ inheritance	___ aliasing	___ name equivalence
___ generic	___ static	___ dynamic
___ overloading	___ method	___ structural equivalence
___ encapsulation		

A. Things which can be determined at compile time.
B. Things which are determined at run time.
C. Allows factoring out common code so that a given piece of code need appear only once.
D. A template for a procedure or package abstraction which can be used for more than one type.
E. A procedure or function which associated to a particular object or class.
F. A single name denoting more than one thing within the same scope.
G. A single data object with two or more names.
H. Two objects are of the same type if they are declared as coming from the same domain of values.
I. Two objects are of the same type if they are declared together or with the same type identifer.
J. The practice of hiding information that the user doesn't need to know about the implementation of the abstraction.

2. Lisp 1: Draw binary tree representations of the following S-expressions:
(a . (b c)) (a (b . c)) (a (b c)) (()()) (nil . (nil . nil))

3. Fill in the blanks:

A. A _____ is an implicit type conversion and a _____ is an explicit type conversion.

B. A _____ word has special meaning in certain syntactic contexts and a _____ word can't be used for a programmer-declared object.

C. The _____ of an operator is its priority in the absence of parentheses. The _____ of an operator is the way it groups with itself.

D. Algol used two forms of parameter passing, call by value and call by _____. In C, by the use of macros (#define's with parameters) one can get the same effect as parameters passed by _____.

E. In C, Pascal, Ada and Algol, local variables are bound to absolute addresses at _____ time, while global variables are bound to absolute addresses at _____ time.

4. For the C code in the middle:

A. Assumming static scoping draw a contour diagram (show parameters and procedures names too).

```
int n = 35;
main()
{
  int x = 5;
  P(7);
}
P(int y)
{
  int z = 2;
  Q(11);
}
Q(int w)
{
/*show*/
}
```

B. Assumming dynamic scoping, draw a contour diagram (show parameters and procedures names too) at the line marked /*show*/.

```
Program Final; Var i, j:integer; a: array[1..2] of integer;

      Procedure One; begin i := 2; end;

      Procedure Two; Var i:integer; begin One; end;

      Procedure Swap ( x, y:integer ); Var i, t: integer;
      begin t := x; x := y; y := t; end.

begin Two; a[1]:= 3; a[2]:= 1; j:= 2; Swap(j, A[j]);end.
```

5. For the Pascal-like code above:
A. If Dynamic scoping is used when procedure two calls procedure one, which "i" is assigned the value two?

B. If Static scoping is used when procedure two calls procedure one, which "i" is assigned the value two?

C. If the call to Swap is call by reference, what are the values of j, a[1] and a[2] after Swap returns?

D. If the call to Swap is call by value, what are the values of j, a[1] and a[2] after Swap returns?

E. If the call to Swap is call by name, what are the values of j, a[1] and a[2] after Swap returns

6. Project (Lisp): evaluate (and simplify where possible):
A. (pairlis '(w x y z) '(a 7 (c) nil) nil)

B. (assoc 'z '((t.7)(u lambda (x) (y))(w.z)(z.5)(t.z)(z.a)(good.doctor)))

C. (mapcar '(lambda (x) (times x (plus x 1))) '(2 5 7 11))

D. (label fn (lambda (x)
      (cond ((eq x 0) 2)(t (plus x (fn (difference x 1))))))) 5)

7. Replace the recursive routine mapPlus2 with an equivalent non-recursive routine.

```
typedef struct node { int value; struct node * next;} Node;
Node * newNode { return (Node *) malloc ( sizeof ( Node ));}

Node * mapPlus2 ( Node * list )
{
      Node * temp;

      if ( list == NULL )
            return NULL;
      temp = newNode();
      temp->value = list->value + 2;
      temp->next =
            mapPlus2 ( list->next );
      return temp;
}
```

8. Project: Use C to write a recursive-descent recognizer for the grammer below. Assume token is the next CHARACTER in the input stream. Assume the function advance(); advances the token to the next character. Assume a main() which has already called advance() once. (i. e. main() could be {advance();printf("%s\n", have_W()? "True" : "False");} Assume the input stream has no white space or newline characters. Write the boolean functions have_W (and respectively, have_X) which return true or false depending on if the input is a string in W (respectively, in X). (Sort of like get_s and get_t but they return true or false instead of anything useful.) W is the start symbol, and X is the only other nonterminal.

W ::= X | %%

X ::= W | @X | #

9. Write "pure" (don't use set or setq) recursive Lisp functions for:
A. lat - a boolean function with one parameter x. Assume x is a list. The function lat returns true if x is nil or if x is a list of atoms.
B. reverse - a function with one parameter x. Assume x is a list. The function reverse returns the list x in reverse order. For example (reverse '(1 2 3)) is (3 2 1).
C. similar - a function with two (s-expression) parameters x and y. The function similar returns true if both x and y have the same binary tree representation except for the names of the atoms. For example (1 (2) 3) and (a (b) c) are similar but (a) and ((a)) are not.

10. Consider two implementations of the C switch statement: "`switch (i) {case 1: S1; break; case 2: S2; break; case 5: S5; break; default: S6;}`". (The statement "`goto L0+i`" is a computed goto; for example, if i=3 this goes to the third line after L0, where it says "`goto L6`".)

(A)

```
      if i<1 then goto L6;
      if i>5 then goto L6;
L0:   goto L0+i;
      goto L1;
      goto L2;
      goto L6;
      goto L6;
      goto L5;
L1:   S1;
      goto L7;
L2:   S2;
      goto L7;
L5:   S5;
      goto L7;
L6:   S6;
L7:   (continue)
```

(B)

```
      if i ≠ 1 then goto L2;
      S1;
      goto L7;
L2:   if i ≠ 2 then goto L5;
      S2;
      goto L7;
L5:   if i ≠ 5 then goto L6;
      S5;
      goto L7;
L6:   S6;
L7:   (continue)
```

Suppose we generalize (A) and (B) above for a switch statement whose lowest and highest case labels are L and H respectively, and which has N nontrivial cases excluding the "default". (Above L=1, H=5, N=3.) Suppose the "goto L0+i" takes two instructions and each other "goto" takes one instruction, and each "if ... then ..." takes three instructions.

Give algebraic formulae (in terms of L, H and N) for the (worst case) execution time and code space (both in number of instructions) for the switch code alone (i.e. excluding the instructions for S1-S6).

(A) Execution Time: _____ Code Size: _____
(B) Execution Time: _____ Code Size: _____

Final by KEY
COP 4020 15 Dec 89

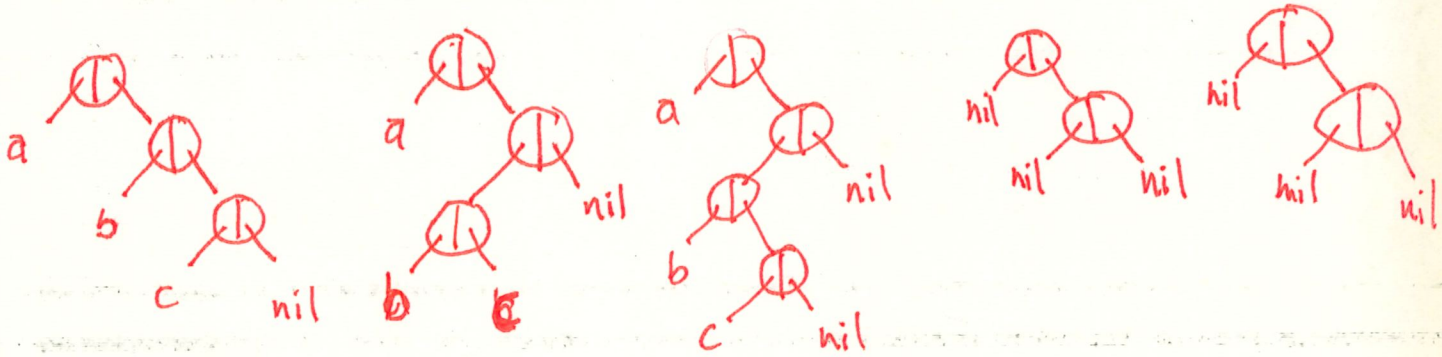
Each problem is worth 10 points. Budget your time carefully.

1 Terminology: Match the letter of the phrase (below) that best applies to the following terms:

C inheritance	G aliasing	I name equivalence
D generic	A static	B dynamic
F overloading	E method	H structural equivalence
J encapsulation		

A. Things which can be determined at compile time.
B. Things which are determined at run time.
C. Allows factoring out common code so that a given piece of code need appear only once.
D. A template for a procedure or package abstraction which can be used for more than one type.
E. A procedure or function which associated to a particular object or class.
F. A single name denoting more than one thing within the same scope.
G. A single data object with two or more names.
H. Two objects are of the same type if they are declared as coming from the same domain of values.
I. Two objects are of the same type if they are declared together or with the same type identifer.
J. The practice of hiding information that the user doesn't need to know about the implementation of the abstraction.

2. Lisp 1: Draw binary tree representations of the following S-expressions:
(a . (b c)) (a (b . c)) (a (b c)) (()()) (nil . (nil . nil))



coercion

3. Fill in the blanks:

A. A coercion is an implicit type conversion and a cast is an explicit type conversion.

B. A key word has special meaning in certain syntactic contexts and a reserve word can't be used for a programmer-declared object.

C. The precedence of an operator is its priority in the absence of parentheses. The associativity of an operator is the way it groups with itself.

D. Algol used two forms of parameter passing, call by value and call by name. In C, by the use of macros (#define's with parameters) one can get the same effect as parameters passed by text.

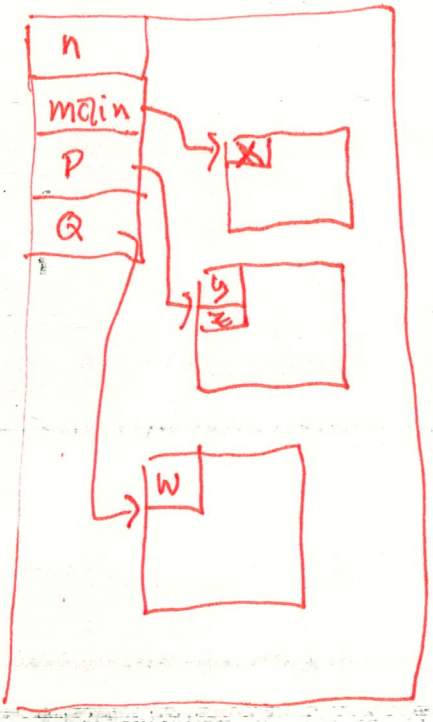
E. In C, Pascal, Ada and Algol, local variables are bound to absolute addresses at run time, while global variables are bound to absolute addresses at link/load time.

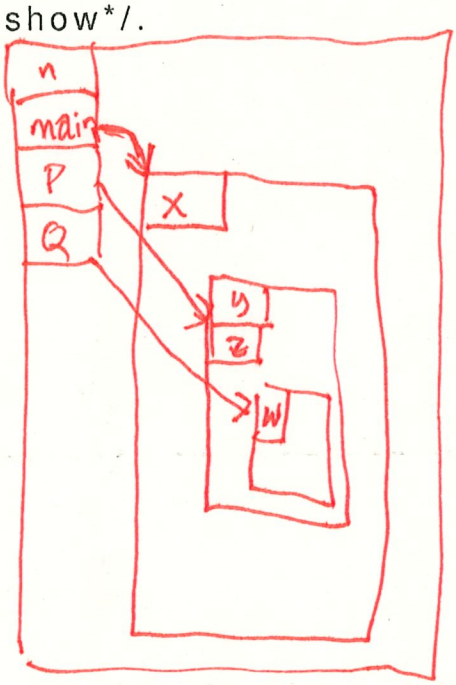
4. For the C code in the middle:

A. Assumming static scoping draw a contour diagram (show parameters and procedures names too).

```
int n = 35;
main()
{
  int x = 5;
  P(7);
}
P(int y)
{
  int z = 2;
  Q(11);
}
Q(int w)
{
/*show*/
}
```

B. Assumming dynamic scoping, draw a contour diagram (show parameters and procedures names too) at the line marked /*show*/.





```
Program Final; Var i, j:integer; a: array[1..2] of integer;

      Procedure One; begin i := 2; end;

      Procedure Two; Var i:integer; begin One; end;

      Procedure Swap ( x, y:integer ); Var i, t: integer;
      begin t := x; x := y; y := t; end.

begin Two; a[1]:= 3; a[2]:= 1; j:= 2; Swap(j, A[j]);end.
```

5. For the Pascal-like code above:

A. If Dynamic scoping is used when procedure two calls procedure one, which "i" is assigned the value two? two.i ← the one in two

B. If Static scoping is used when procedure two calls procedure one, which "i" is assigned the value two? Final.i ← the global

C. If the call to Swap is call by reference, what are the values of j, a[1] and a[2] after Swap returns?

J = 1 A[1] = 3 A[2] = 2

D. If the call to Swap is call by value, what are the values of j, a[1] and a[2] after Swap returns?

J = 2 A[1] = 3 A[2] = 1

E. If the call to Swap is call by name, what are the values of j, a[1] and a[2] after Swap returns

J = 1 A[1] = 2 A[2] = 1

6. Project (Lisp): evaluate (and simplify where possible):

A. (pairlis '(w x y z) '(a 7 (c) nil) nil)

((w.a) (x.7) (y c) (z))

B. (assoc 'z '((t.7)(u lambda (x) (y))(w.z)(z.5)(t.z)(z.a)(good.doctor)))

(z.5)

C. (mapcar '(lambda (x) (times x (plus x 1))) '(2 5 7 11))

(6 30 56 132)

D. (label fn (lambda (x)
(cond ((eq x 0) 2)(t (plus x (fn (difference x 1))))) 5)

5+4+3+2+1+2

17

7. Replace the recursive routine mapPlus2 with an equivalent non-recursive routine.

```
typedef struct node { int value; struct node * next;} Node;
Node * newNode { return (Node *) malloc ( sizeof ( Node ));}

Node * mapPlus2 ( Node * list )
{
      Node * temp;

      if ( list == NULL )
            return NULL;
      temp = newNode();
      temp->value = list->value + 2;
      temp->next =
            mapPlus2 ( list->next );
      return temp;
}
```

```
Node * mapPlus2 (Node * list)
{
  Node * head, *temp;
  if (list==NULL) return NULL;
  for (head=temp=new Node();
       list; list=list->next){
  {
      temp->value = list->value+2
      temp->next = new Node()
      temp = temp->next
  }
  temp = NULL; return head;
}
```

8. Project: Use C to write a recursive-descent recognizer for the grammer below. Assume token is the next CHARACTER in the input stream. Assume the function advance(); advances the token to the next character. Assume a main() which has already called advance() once. (i. e. main() could be {advance();printf("%s\n", have_W()? "True" : "False");} Assume the input stream has no white space or newline characters. Write the boolean functions have_W (and respectively, have_X) which return true or false depending on if the input is a string in W (respectively, in X). (Sort of like get_s and get_t but they return true or false instead of anything useful.) W is the start symbol, and X is the only other nonterminal.

W ::= X | %%

X ::= W | @X | #

```
boolean get_W()
{
  if (token == '%')
  {
    advance();
    if (token == '%')
    {
      advance();
      return true;
    }
    else
    {
      advance()
      return false;
    }
  }
  else
    return getX()
}
```

```
boolean get_X()
{
  if (token == '#')
  {
    advance();
    return true;
  }
  if (token == '@')
  {
    advance();
    return get_X()
  }
  if (token == '$')
  {
    advance();
    if (!get_W()) return false
    if (token == '$')
    { advance(); return true;
    }else
    { advance(); return false;
    }
  }
  return false;
}
```

9. Write "pure" (don't use set or setq) recursive Lisp functions for:
A. lat - a boolean function with one parameter x. Assume x is a list. The function lat returns true if x is nil or if x is a list of atoms.
B. reverse - a function with one parameter x. Assume x is a list. The function reverse returns the list x in reverse order. For example (reverse '(1 2 3)) is (3 2 1).
C. similar - a function with two (s-expression) parameters x and y. The function similar returns true if both x and y have the same binary tree representation except for the names of the atoms. For example (1 (2) 3) and (a (b) c) are similar but (a) and ((a)) are not.

```
(def similar (lambda (x y)
  (cond ((atom x) (atom y))
        ((atom y) nil)
        (t (and (similar (car x) (car y))
                (similar (cdr x) (cdr y)))))))
```

```
(def lat (lambda (x)
  (cond ((null x) t)
        (t (and (atom (car x)) (lat (cdr x)))))))
```

```
(def reverse (lambda (x)
  (cond ((null x) nil)
        (t (append (reverse (cdr x))
                   (list (car x)))))))
```

10. Consider two implementations of the C switch statement: "`switch (i) {case 1: S1; break; case 2: S2; break; case 5: S5; break; default: S6;}`". (The statement "`goto L0+i`" is a computed goto; for example, if i=3 this goes to the third line after L0, where it says "`goto L6`".)

(A)

```
      if i<1 then goto L6;
      if i>5 then goto L6;
L0:   goto L0+i;
      goto L1;
      goto L2;
      goto L6;
      goto L6;
      goto L5;
L1:   S1;
      goto L7;
L2:   S2;
      goto L7;
L5:   S5;
      goto L7;
L6:   S6;
L7:   (continue)
```

(B)

```
      if i ≠ 1 then goto L2;
      S1;
      goto L7;
L2:   if i ≠ 2 then goto L5;
      S2;
      goto L7;
L5:   if i ≠ 5 then goto L6;
      S5;
      goto L7;
L6:   S6;
L7:   (continue)
```

Suppose we generalize (A) and (B) above for a switch statement whose lowest and highest case labels are L and H respectively, and which has N nontrivial cases excluding the "default". (Above L=1, H=5, N=3.) Suppose the "goto L0+i" takes two instructions and each other "goto" takes one instruction, and each "if ... then ..." takes three instructions.

Give algebraic formulae (in terms of L, H and N) for the (worst case) execution time and code space (both in number of instructions) for the switch code alone (i.e. excluding the instructions for S1-S6).

(A) Execution Time: 10 Code Size: 8+(H-L+1)+N+1

(B) Execution Time: 3N+1 Code Size: 4N