ECECS 511 Organization of Programming Languages

Masters Exam, June 1996

Name ______________________

1. The following pages contain six questions.

2. Questions 1 and 2 are worth 20 points each. Questions 3 through 6 are worth 10 points each.

3. You must answer questions worth 60 points.

4. Answer for each question must begin on a new sheet of paper.

1. (20 Points) Write Scheme code that implements a stream network specified as follows:

   (a) To create a producer stream: (define s1 (stream f)) where f is a recursive procedure that produces tokens. Each such f has a line (s-out token) which outputs a token to the stream. It is assumed that f will act as a coroutine and control will be returned to the (s-out) statement, with a value that you specify, when another token is demanded.

   (b) To close a stream: (close s). This means the created stream s can no longer supply tokens.

   (c) To consume a token: (get s). Returns the value of next token from stream s.

2. (20 points) Repeat question number 1 using C or C++ or Java.
3. (10 points) The function named \texttt{fac} is recursive.

\begin{verbatim}
(define fac (lambda (n) (cond ((zero? n) 1) (else (* n (fac (- n 1)))))))
\end{verbatim}

For each fixed point operator \( Y \), define a nonrecursive function \( F \) such that \( (Y F) \) is \( \text{fac} \). If for some reason, there is no such \( F \) that will work for a particular \( Y \) in Scheme, then write down the reason.

(a) \begin{verbatim}
(define Y (lambda (f) ((lambda (w)(w w))(lambda (x) (f (x x)))))
\end{verbatim}

(b) \begin{verbatim}
(define Y (lambda (f) ((lambda (w)(w w))(lambda (x) (f (lambda ()(x x)))))))
\end{verbatim}

(c) \begin{verbatim}
(define Y (lambda (f) ((lambda (w)(w w))(lambda (x) (f (lambda (u)((x x) u)))))))
\end{verbatim}

(d) \begin{verbatim}
(define Y (lambda (f) ((lambda (w)(w w))(lambda (x) (f (delay (x x)))))))
\end{verbatim}
4. (10 points) The following program uses continuation to handle exceptions.

```
(define divsum (lambda (a lat) (call-with-current-continuation (lambda (cc)
    (((lambda (f) ((lambda (w)(w w))(lambda (x)(f (lambda ()(x x))))))
      (lambda (g) (lambda (l) (cond ((null? l) 0) (else (+
        (cond ((zero? (car l))(cc 'division_by_zero))(else (/ a (car l))))
          ((g)(cdr l))))))))
    lat))))).
```

(a) What is the current continuation cc generated by call-with-current-continuation in the above program? Write it in a lambda form.

(b) call-with-current-continuation can be described as a function (A) of one argument, which is a function (B) of one argument, which is a function (C) of one argument. In which order are these functions executed?

(c) Which function uses the value of (cc 'division_by_zero) in the program? (Select one)

i. (lambda (cc) ...).

ii. (lambda (f) ...).

iii. (lambda (g) ...).

iv. +.

v. None of the above.

(d) Why can we not rewrite the program as follows?

```
(define ds ((lambda (f) ((lambda (w)(w w))(lambda (x)(f (lambda ()(x x))))))
    (lambda (g) (lambda (l) (cond ((null? l) 0) (else (+
        (cond ((zero? (car l))(cc 'division_by_zero))(else (/ a (car l))))
          ((g)(cdr l))))))))

(define divsum (lambda (a lat) (call-with-current-continuation
    (lambda (cc) (ds lat))))))
```
5. (10 points) The following is a definition of the `String` class with shared representation and reference counting.

```cpp
#class String {
    struct StringValue {
        int refCount;
        char *data;
        StringValue( const char *s) :
            refCount(1),
            data(new char[strlen(s)+1]))
        { strcpy(data,s); }
    ~StringValue() { delete [] data; }
    }; StringValue *value;
public:
    class CharProxy {
        String& theString;
        int charIndex;
    public:
        CharProxy(String& str, int i) :
            theString(str), charIndex(i) {} CharProxy& operator=(const CharProxy&);
        CharProxy& operator=(char );
        operator char() { return
            theString.value->data[charIndex]; }
    }; String(const char *s = "") :
        value( new StringValue(s)) {} String(const String& str) :
            value(str.value)
        { value->refCount++; }
    ~String();
    String& operator=( const String& );
    CharProxy operator[](int i) {
        return CharProxy(*this, i); }
    friend class CharProxy;
};
```

(a) Suppose a and b are String objects. Which statement requires the use of the `operator=(const CharProxy&)` member function of CharProxy?

   i. `cout << a[5];`
   ii. `a[5] = 'x';`
   iii. `a[5] = b[3];`
   iv. `a = b;`

(b) Which statement requires the use of the `operator=(char )` member function of CharProxy?

   i. `cout << a[5];`
   ii. `a[5] = 'x';`
iii. `a[5] = b[3];`
iv. `a = b;`

(c) Which statement requires the use of the operator `char()` member function of `CharProxy`?

i. `cout << a[5];`
ii. `a[5] = 'x';`
iii. `a[5] = b[3];`
iv. `a = b;`

(d) Define the destructor for `String` so that the shared representation will not be deleted until the reference count becomes zero.

(e) Define the assignment operator for `String`. 
6. (10 points) The following is a finite automaton implemented with exception handling.

```c
class Back {};  
char ch, buf[BFSZ];

void begin(){
    while(cin.get(ch)){
        if(ch==033) try{
            doEsc(); }
        catch(Back){
            continue; }
    }
}

void doEsc(){
    cin.get(ch);
    switch(ch){
        case '*': doStar();
        case '(': doParen();
        case '&': doAmp();
        default: throw(Back);
    }
}

void doStar(){
    cin.get(ch);
    switch(ch){
        case 'p': doSp();
        case 'c': doSc();
        default: throw(Back);
    }
}

void doSp(){
    for(i=0;i++){
        cin.get(buf[i]);
        if(isalpha(buf[i]))
            break;
    }
    switch(buf[i]){  
        case 'x': updatePos(0);
            doSp();
        case 'X': updatePos(0);
            doPlot();
        case 'y': updatePos(1);
            doSp();
        case 'Y': updatePos(1);
            doPlot();
```

default: throw(Back);
}
}

void doPlot(){
  for(;;){ cin.get(ch);
    if(ch==033) throw(Back);
    else plot(ch);
  }
}

(a) Suppose the input is the sequence (Esc has the ASCII 033)
    Esc*p1234x567Y....Esc

From which function will the exception Back be thrown?
  i. doEsc()
  ii. doStar()
  iii. doSp()
  iv. doPlot()

(b) After a throw of the exception Back is caught, which function is active?
  i. begin()
  ii. doEsc()
  iii. doStar()
  iv. doSp()

(c) In this implementation of a finite automaton, each C function is the embodiment of
  i. a state in the transition diagram.
  ii. an input symbol.
  iii. an action associated with a transition.
  iv. a separate finite automaton.
Organization of Programming Languages

Master's Exam, November 11, 1995

Name _____________________________

1. The following C++ and ML programs generate different results.

```c++
int x = 8;
int f( int y) { return x + y; }
main() { x = 3; cout << f(x); }
```

```ml
val x = 8;
fun f(y) = x + y;
val x = 3;
f(x);
```

(a) The output from the C++ program is ____________.
(b) The output from the ML program is ____________.
(c) Explain this difference.
2. Below is a C program that is syntactically correct to the GNU gcc compiler.

```c
int (*K( int x))(int) {
    int M( int y) { return x; }
    return &M;
}

main() {
    int (*f1)(int)=K(9);
    int (*f2)(int)=K(7);
    printf("%d\n",f2(f1(5)));
}
```

(a) Describe the type of K.

(b) K is intended to be an implementation of the K combinator in λ calculus, which is $K = \lambda x.(\lambda y.x)$. If this implementation is successful, then the output of the program is ____________.

(c) However, the output of the program is 5. Give an explanation to this outcome.

(d) How must the gcc compiler be modified to make the implementation of K possible?
(defun subst (x y z) (cond ((atom z) (cond ((eq y z) x) (t-z)))
  (t (cons (subst x y (car z))
           (subst x y (cdr z)))))))

The value of a list is the result of the value of its first element as a function applied to the values of other elements of the list as arguments. The first element can be a name, a lambda expression, or a list whose evaluation results in a lambda expression. This pure Lisp uses the lexical scope rule.

A pure lambda calculus term is one of the following three forms:

- x a variable.
- (M N) a list of two terms (function application).
- (L x M) a list of three items, the first one being a keyword (lambda).
  - the second a variable, and the third a term (abstraction).

A reduction expression (redex) is a subterm in the form of ((L x M) N). It can be reduced to N with every free occurrences of x in it replaced with M. (We assume that all bound variables have been systematically renamed.) A term is in normal form if it cannot be reduced. A pure lambda calculus term can be reduced to its normal form. If the innermost redex is reduced first, the reduction is called call-by-value.

Do only three of the following four problems.

1. What is the output of the following program if the parameter-passing method is 1) call-by-value; 2) call-by-reference; 3) call-by-name; 4) call-by-value-result?

   program main(output);
   var a: integer;
   procedure proc1(x: integer);
     begin x:=0; a:=1 end;
   begin a:=2; proc1(a); writeln(a) end.

2. Define a pure Lisp function that accepts a pure lambda calculus term as the argument and returns its normal form. The reduction should be call-by-value.

3. Rewrite the subst function above so that it is no longer a recursive definition. (hint: Incorporate a fixed-point combinator as a lambda expression in the definition.)

4. Show by giving an example that if the dynamic scope rule were used in the pure Lisp defined above, then the renaming principle must be violated. The renaming principle says that the computation set up by a program is unaffected if local variables and formal parameters in the source text are consistently renamed.

2
1. Consider the following program outline:

Module $M$;
procedure $P()$
  var $x$, $y$, $z$;
procedure $Q()$;
procedure $R()$;
  begin (* $R$ *)
    ... $z := P();$
  end $R$;
begin (* $Q$ *)
  ... $y := R();$
end $Q$;
begin (* $P$ *)
  ... $x := Q();$
end $P$;
begin (* $M$ *)
  ... $P();$
end $M$;

(a) (4) Show the activation stack for this module at the time when it has first grown to have seven entries. Show the name of each procedure on the corresponding stack entry.

(b) (10) Show the control (dynamic) and access (static) links on the above stack of activations.

2. (8) How does the semantics (and the role) of *If-Then-Else* construct in a procedural language (such as PASCAL) differ from its semantics and role in a functional language (such as SCHEME)? (Answer in less than 50 words.)

3. Consider the following ML type syntax:

   let $a=3$
   in
   let $p = fn (x) : (a + x)$;
   $a = 5$
   in
   ($a * p(2)$);

What value will be returned when

(a) (8) Dynamic scope rules are used.

(b) (8) Static scope rules are used.

4. Consider the following procedure and program segment.

procedure strange (var $x$ : integer; $y$ : integer);
var temp : integer;
begin
  print('PRE:', $x$, $y$, \newline);
  temp := $x$; $x := x + y$; $y := temp$;
  print('POST:', $x$, $y$, \newline);
end strange;
(18) Assume that all formal parameters preceded by keyword var are passed as call-by-NAME and all other formal parameters are passed as call-by-VALUE. In the above example x is passed by name and y by value. What changes will be made to the array A and what will be printed by the following sequence of program statements? Show the status of the array after each statement and assume that they are executed in the order shown below. Show all of your work and assumptions in arriving at your answer.

for i := 1 to 100 do A[i] := 100-i;
  i := 10;
  strange(i, A[i]);
  j := i-1;
  strange(A[i], A[j])

5. Consider the following LISP type syntax:

(define compose
 (lambda (f)
   (lambda (g)
     (lambda (x)
       (f (g x)))))

(define d2 (compose car))

(define d2r2 (d2 cdr))

(a) (8) cdr is a polymorphic operator whose type can be given as: (z list -> z list). What are the types of the functions d2 and d2r2?

(b) (8) What are the closures associated with d2 and d2r2?

6. Consider the following code in a functional language:

(set find (lambda (pred lis)
   (if (null? lis) '()
     (if (pred (car lis)) 'T
       (find pred (cadr lis)))))

(a) (4) What is wrong with the above code?

(b) (8) What is the difference in the ways SCHEME and ML handle the problem with the above code?

7. Consider the following program segment from an object oriented language.

(class C Object ()
  (define m1 () (m2 self))
  (define M2 () #C)
(class D C ()
  (define m2 () #D)
  (define m3 (m1 self)))

(a) (8) What will be the response after each of the following operations:

(set x (new D))
(m1 x)
(m3 x)

8. (8) In the context of Lambda Calculus what is a Fixed Point Combinator?
1. The two C++ functions that follow use call-by-pointer and call-by-reference respectively.

```cpp
void swap1(int *a, int *b) {
    int t=*a; *a=*b; *b=t; }
void swap2(int &a, int &b) {
    int t=a; a=b; b=t; }
```

(a) Which function uses call-by-reference?

(b) Is there any difference in the usage of these functions? If your answer is yes, explain.

(c) Is there any difference in the implementation of these functions? Explain.

(d) What problems will be there if these functions are called via a computer network (remote procedure call)? Is there any remedy?
2. The following is a Pascal definition of the factorial function.

```pascal
function factorial(n:integer):integer;
function identity(n:integer):integer;
begin identity:=n end;

function fac1(n:integer;function f(i:integer):integer);
function fac2(m:integer):integer;
begin fac2:=f(n+m) end;
begin if n<=1 then fac1:=f(1) else fac1:=fac1(n-1,fac2) end;

begin factorial:=fac1(n,identity) end;
```

(a) What programming style is used in this definition?

(b) Draw a diagram that represents the run time stack when `factorial(3)` is called. Indicate in each activation record (stack frame) the value of each local name and where the static (lexical) link points to.

(c) The function `fac1` has a function parameter. What must be stored in the activation record of `fac1` for the local name `f`?
3. In $\lambda$-calculus, a fixed-point combinator returns the fixed point of its argument. The following is the Y combinator.

$$\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

(a) Assume the following is the $\lambda$-expression for the factorial function.

$$\text{Fac} = \lambda x.\text{if}((x < 1), 1, x \times \text{Fac}(x - 1))$$

Write down a non-recursive function whose fixed point is the factorial function.

(b) What is a major obstacle that prevents the implementation of Y in a functional programming language like Scheme? Choose one of the following reasons and explain: typelessness, scope rule, parameter passing, functions that create functions.

4. The following C and ML programs generate different results.

```c
int x = 3;
int f( int y ) { return x + y; }
main() { x = 8; printf("%d\n", f(x)); }
```

```ml
val x = 3;
fun f(y) = x + y;
val x = 8;
f(x);
```

(a) What are the values of $f(x)$ generated by the two programs?

(b) Give an explanation why these values are different.
5. The K combinator in \( \lambda \)-calculus is

\[ K = \lambda x. (\lambda y. x) \]

which is implemented in C and compiled with the gcc compiler, which allows nested function definition as follows.

```c
int (*K( int x))(int) {
    int M( int y) { return x; }
    return &M;
}

main() {
    int (*f1)(int)=K(9);
    int (*f2)(int)=K(8);
    printf("%d\n", f1(f2(7)));
}
```

(a) If this C program correctly implements a typed version of the K combinator, then what is the output?

(b) The actual output is 7, which is not correct. Explain why.

(c) K can be correctly implemented in Scheme. What feature in Scheme which is lacking in the gcc compiler makes this possible?
The following description of a pure Lisp and the pure Lambda calculus are used in some of the problems.

A pure Lisp expression is either an atom or a list. The empty list (nil or ()) is both an atom and a list. The following built-in functions and constructs are available:

- t An atom that always evaluates to true (non-nil).
- (quote x) or 'x Quote, with value x.
- (eq x y) True if the values of variables x and y are the same atom, otherwise nil.
- (atom x) True if the value of x is an atom, otherwise nil.
- (cond L1 L2 ... Ln) Each Li is a list. The first element of the list Li is evaluated, if the result is not nil, then other element(s) in Li are evaluated, and the value of the last one is returned. Otherwise, the result is (cond L2 ... Ln). If there is only one element in a list, and its value is non-nil, then its value is returned. If there is no list following cond, the value is nil.
- (car x) Returns the first element of the value of x, which must be a non-empty list.
- (cdr x) Returns the tail of the value of x, which must be a non-empty list.
- (cons x y) Returns a list whose first element is the value of x, and the tail is the value of y.
- (lambda (parameters) expression) The formal specification of a function. It is called a lambda expression.
- (defun name (parameters) expression) Attaches the formal expression (lambda (parameters) expression) to the name name. Recursion is allowed.

Some predefined functions are also available:

- (car (cdr x)) is (cadr x), etc.
- (list x y z) returns a list of three elements, the values of x, y, and z. The number of arguments is not limited.
(defun subst (x y z) (cond ((atom z) (cond ((eq y z) x) (t z)))
  (t (cons (subst x y (car z))
         (subst x y (cdr z)))))))

The value of a list is the result of the value of its first element as a function
applied to the values of other elements of the list as arguments. The first
element can be a name, a lambda expression, or a list whose evaluation results
in a lambda expression. This pure Lisp uses the lexical scope rule.

A pure lambda calculus term is of one of the following three forms:

  x a variable.
  (M N) a list of two terms (function application).
  (L x M) a list of three items, the first one being a keyword (lambda).
    the second a variable, and the third a term (abstraction).

A reduction expression (redex) is a subterm in the form of ((L x M) N). It
can be reduced to N with every free occurrences of x in it replaced with M. (We
assume that all bound variables have been systematically renamed.) A term is
in normal form if it cannot be reduced. A pure lambda calculus term can be
reduced to its normal form. If the innermost redex is reduced first, the reduction
is called call-by-value.

Do only three of the following four problems.

1. What is the output of the following program if the parameter-passing
   method is 1) call-by-value; 2) call-by-reference; 3) call-by-name; 4) call-by-
   value-result?

   program main(output);
   var a: integer;
   procedure proc1(x: integer);
   begin x:=0; a:=1 end;
   begin a:=2; proc1(a); writeln(a) end.

2. Define a pure Lisp function that accepts a pure lambda calculus term as
   the argument and returns its normal form. The reduction should be call-by-
   value.

3. Rewrite the subst function above so that it is no longer a recursive
definition. (hint: Incorporate a fixed-point combinator as a lambda expression
   in the definition.)

4. Show by giving an example that if the dynamic scope rule were used
   in the pure Lisp defined above, then the renaming principle must be violated.
The renaming principle says that the computation set up by a program is unaf-
fected if local variables and formal parameters in the source text are consistently
renamed.
3. Below is a Pascal definition of the factorial function.

```pascal
function factorial(n:integer): integer;
  function identity(n:integer): integer;
  begin identity:=n end;

function fac1(n:integer;
  function f(i:integer):integer): integer;
function fac2(m:integer): integer;
  begin fac2:=f(n*m) end;
  begin if n<=1 then fac1:=f(1)
    else fac1:=fac1(n-1,fac2) end;

begin factorial:=fac1(n,identity) end.
```

(a) Identify the programming style used in this definition.

(b) The run time stack for the invocation of `factorial(3)` is sketched below. (AR: activation record)

<table>
<thead>
<tr>
<th>AR label</th>
<th>function name</th>
<th>static link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>factorial(3)</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>fac1(3,identity)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>fac1(2,fac2)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>fac1(1,fac2)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>fac2(1)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>fac2(2)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>identity(6)</td>
<td></td>
</tr>
</tbody>
</table>

Fill out the static(lexical) link column with AR labels indicating the AR that the static link of each activation record points to.

(c) Explain how the static links are computed at run time.
CS 511
Organization of Programming Languages

The following description of a pure Lisp and the pure Lambda calculus are used in some of the problems.

A pure Lisp expression is either an atom or a list. The empty list (nil or () is both an atom and a list. The following built-in functions and constructs are available:

- An atom that always evaluates to true (non-nil).
- (quote x) or 'x Quote, with value x.
- (eq x y) True if the values of variables x and y are the same atom, otherwise nil.
- (atom x) True if the value of x is an atom, otherwise nil.
- (cond L1 L2 ... Ln) Each Li is a list. The first element of the list L1 is evaluated, if the result is not nil, then other element(s) in L1 are evaluated, and the value of the last one is returned. Otherwise, the result is (cond L2 ... Ln). If there is only one element in a list, and its value is non-nil, then its value is returned. If there is no list following cond, the value is nil.
- (car x) Returns the first element of the value of x, which must be a non-empty list.
- (cdr x) Returns the tail of the value of x, which must be a non-empty list.
- (cons x y) Returns a list whose first element is the value of x, and the tail is the value of y.
- (lambda (parameters) expression) The formal specification of a function. It is called a lambda expression.
- (define name (parameters) expression) Attaches the formal expression (lambda (parameters) expression) to the name name. Recursion is allowed.

Some predefined functions are also available:

- (cadr x) is (car (cdr x)), etc.
- (list x y z) returns a list of three elements, the values of x, y, and z. The number of arguments is not limited.
6. (20 points) Explain the following items in a stack-based implementation of Pascal, or a Pascal-like language.

a. Return Address
b. Program Counter
c. Static link
d. Dynamic link
5. (20 points) Suppose procedures can be passed to other procedures as parameters. Nonlocals in the procedures can be found in their definition environment, their caller's environment, or their caller's environment. For example, in the following program, the nonlocal i can be found in the main program, in procedure p1, or procedure p2, depending on three different scope rules.

```pascal
program main(input, output);
var i: integer;
procedure p3;
begin writeln(i) end;
procedure p1(procedure f);
var i: integer;
begin i:=1; f end;
procedure p2;
var i: integer;
begin i:=2; p1(p3) end;
begin i:=3; p2 end.
```

What are the outputs of this program for these three different scope rules? What must be passed to the activation record for p1 when p2 passes p3 to p2 under each of these three scope rules?
3. (20 points) The following is a program written in a Pascal-like language, but the parameter passing mechanism is unknown. Suppose the output of the program is positive, which one(s) of the four combinations is(are) possible?

```pascal
program main(input, output);
var i: integer;
    procedure p1(i, j);
    begin j := i*2; if j > i then writeln('positive')
           else writeln('negative')
    end;
begin i := 6; p1(i, i) end.
```

a. Both arguments of p1 are passed as call-by-value.
b. Both arguments are passed as call-by-reference.
c. Both arguments are passed as call-by-name.
d. Both arguments are passed as call-by-value-result.

4. (20 points) ML functions curry and twice are defined as

```ml
fun curry f x y = f (x, y);
fun twice f x = f (f x);
```

Suppose mult is a function that returns the product of its two arguments. Specify the types of the following expressions.

```plaintext
curry mult
curry mult 3
twice mult
twice mult 3
```
Master's Exam
Organization of Programming Languages

NAME _______________________

5 May 1990

You have the option to do five out of the following six problems.
1. (20 points) Connect each feature with the language in the list that possesses the feature most prominently.

Features Languages
Independence from Algol 60 SmallTalk
Using call-by-value exclusively Lisp
A strongly-typed functional programming language C
Nested procedures are allowed ML
Object-oriented Ada

2. (20 points) The following is a Pascal program:

   program main(input,output);
   var i:integer;
   procedure p1;
   begin writeln(i) end;
   procedure p2;
   var i:integer;
   begin i:=7; p1 end;
   begin i:=3; p2 end.

What would the output of the program be, should Pascal follow the static scope rule? What would be the output should Pascal follow the dynamic scope rule? Which one does Pascal follow?
(c) In the second line of the expression, list the unary, binary, and keyword messages and state their precedence relations.

(d) In the third line of the expression, how is the meaning of the message // determined? Is this the division method defined in the integer class?
Smalltalk

(a) Define the term multi-inheritance in the context of object-oriented programming. Discuss the benefits and disadvantages of it.

(b) How many objects and messages are involved in the first line of the expression? List them and explain.
ML

(a) What is the type of the function `curry` defined as follows:

```haskell
fun curry 1 x y = f (x, y);
```

(b) Define the term `parametric polymorphism` in the context of ML. Contrast it to `overloading`. Discuss benefits and disadvantages of each.

(c) Give one benefit and one disbenefit of strong typing in programming language design.
(c) Define the function `delete` that deletes all occurrences of its first argument as an atom in its second argument, which must be a list, but otherwise leave the structure of the second argument unchanged. For example, `(delete 'a '((a)))` returns `()`, and `(delete 'a '((b a)))` returns `((b))`. You may use only `defun`, `cond`, `atom`, `null`, `car`, `cdr`, `cons`, `nil`, and `t` as built-in functions in your definition.
4 FUNCTIONAL & OBJECT-ORIENTED PROGRAMMING LANGUAGES

Select one and only one programming language among Lisp, ML, and SmallTalk, and answer all the questions about that language in this section.

Lisp

(a) Does the evaluation of the following Lisp expression terminate? If the answer is yes, what is the value of the expression?

\[ (\text{lambda}(x) (x x)) \quad \text{(lambda} (x) (\text{car} x)) \]

(b) Why early Lisps are dynamically scoped, even through \( \lambda \)-calculus is statically scoped? Describe one consequence of the dynamic scope rule upon Lisp programming.
(c) The constant \texttt{fix} and its corresponding \texttt{t-rule} can be realized by many pure lambda terms. Write down one of them.
4. Consider the following program segment:

```
procedure switch (var x : integer; var y : integer; var z : integer);
  var w : integer;
begin
  w := x; x := y; y := z; z := w;
end switch;
```

Assume that an array $A$ of 25 integers is so initialized that for $i=2\ldots25$ $A[i] = (A[i-1] + i + 1)$ and $A[1] = 1$. Consider the following sequence of a program's statements:

```
i := 2; j := 3;
print (i, j, A[i]);
switch (i, j, A[i]);
print (i, j, A[i]);
print (i, A[i], A[i+1]);
switch (i, A[i], A[i+1]);
print (i, A[i], A[i+1]);
```

What will be printed by the above program segment if the parameters are passed by:

(a) value-result
(b) reference

5. Answer the following in the context of a LISP environment:

(a) Show the cons-node data structures for the following lists

i. `((((A B C))))`
ii. `(A (A (A) A) A)`

(b) By drawing the appropriate data structures only, show the difference between a destructive append and a non-destructive append operation on two lists.

(c) What are the differences between a simple List and an Association-List?
Important Notes:

1. Do any three of the following four problems.
2. All questions carry equal marks.
3. Show all the work that you do for obtaining your answers.

1. Answer all the following questions.

(a) (5) What is the central and most crucial characteristic of the paradigm of functional programming (when compared to, say, procedural languages such as PASCAL)?

(b) Consider the following definitions in SCHEME-like syntax.

\[
\text{(set gcd* (lambda (l) (gcd*-aux l id)))}
\]

\[
\text{(set gcd*-aux (lambda (l f))}
\]
\[
\text{ (if (= (car l) 1)}
\]
\[
\text{ 1}
\]
\[
\text{  (if (null? (cdr l))}
\]
\[
\text{  (f (car l))}
\]
\[
\text{  (gcd*-aux (cdr l) (lambda (n) (f (gcd (car l) n))))))}
\]

\[
\text{(set gcd (lambda (m n))}
\]
\[
\text{  (if (= n 0)}
\]
\[
\text{  m}
\]
\[
\text{  (gcd n (mod m n))))}
\]

\[
\text{(set id (lambda (x) x))}
\]

i. (8) What is the type of the function \text{gcd}?

ii. (8) What is the type of the function \text{gcd*-aux}?

iii. (4) How many times would the function \text{gcd} be invoked while evaluating the expression (\text{gcd* '}(24 54 78))?

iv. (4) How many times would the function \text{gcd} be invoked while evaluating the expression (\text{gcd* '}(24 1 54 78))?

v. (4) How many times would the function \text{gcd} be invoked while evaluating the expression (\text{gcd* '}(24 54 78 1))?
2. Consider the following expressions of a LISP-like language.

\[ \text{-> (set a 3)} \]
\[ \text{-> (defun p (x)} \]
\[ \text{\quad (if (= x 1)} \]
\[ \text{\quad 1} \]
\[ \text{\quad (+ a (p (- x 1))))} \]
\[ \text{-> (defun q (a x) p(x))} \]

(a) What will be the output for the expression \((q 4 4)\) when
   i. (6) static scoping is used?
   ii. (6) dynamic scoping is used?

(b) (6) What, in your view, is the main difference between the languages SCHEME and LISP?

(c) (6) What would be one major adverse implication if dynamic scoping rules were used in PASCAL?

(d) (6) What is the difference between \textit{deep binding} and \textit{shallow binding} techniques?

(e) (3) What constitutes a \textit{closure} of a function? Give a small example.

3. Answer the following:

(a) (9) Call-by-reference allows us to write some procedures that can not be written using call-by-value. Give at least two examples of such procedures.

(b) (12) How would you implement passing of an array (of integers) parameter when it should be passed: 1. \textit{by value} and 2. \textit{by reference}? Does your implementation work for the case when each element of the array is an also an array? If not, suggest modifications.

(c) (12) How is a \textit{call-by-value-result} implemented? What is the main advantage and the main disadvantage of this parameter passing technique?
4. Answer the following questions:

(a) (8) Show the cons-node data structures for the following S-expressions (assume LISP environment):
   i. (a ((b (c d r)) x) g)
   ii. ((a b) ((c d)) (((e f)))

(b) (5) What is the difference between a **destructive** and a **non-destructive** LISP operation? Illustrate your answer with an example and any relevant data structures.

(c) (6) What is the most important characteristic of the programming paradigm known as **Object Oriented Programming**?

(d) (7) Is it correct to state that object-oriented-programming allows an abstract data type to have multiple implementations that co-exist in a single program? Give reasons for your answer.

(e) (7) What is the role of the **super** variable in an object-oriented language, such as SMALLTALK?
Name:........................................

Computer Science Department
Master's Exam: Organization of Programming Languages
Autumn 1993
Date: Nov. 13th, 1993

Read these before starting:

1. Do any three of the following four problems.
2. All questions carry equal points.
3. Show all the work that you do for obtaining your answers.

1. Consider the following definitions in SCHEME-like syntax.

(build ccombine (lambda (f1 sum zero)
   (lambda (f2 n L)
      (if (null? L) zero
         (if (f2 n (car L))
            (sum (f1 (car L)) ((ccombine f1 sum zero) f2 n (cdr L)))
            ((ccombine f1 sum zero) f2 n (cdr L))))))

(set pos (lambda (n x) (if (< x n) 0 1)))
(set sq (lambda (x) (* x x)))
(set **5 (ccombine sq + 0))

(a) (6) What will be the output for the expression (**5 pos 5 '(7 9 -2 3 4 -1 8))? Give explanation for your answer.
(b) (12) Write the required expressions for generating a function named */+ (from ccombine)
   which can multiply all the positive elements of a list when appropriate arguments are
   provided.
(c) (12) Determine the type for the function **5. Show all the steps used in deriving your answer.

1993

Prof. Langs.
2. Consider the following expressions of a LISP-like language.

\[
\begin{align*}
\text{-> } & (\text{set } f 5) \\
\text{-> } & (\text{set } b 8) \\
\text{-> } & (\text{set } d 3) \\
\text{-> } & (\text{define } a (b c) (+ (* (\text{succ } b) (+ c d)))) \\
\text{-> } & (\text{define } e (c d) (+ (a f c) (a b d))) \\
\text{-> } & (\text{define } h (c b d) (+ (a b f) (e b c)))
\end{align*}
\]

(a) What will be the output for the expression (h 1 6 7) when
   i. (10) static scoping is used?
   ii. (10) dynamic scoping is used?

(b) (10) What is polymorphism? Give an example of polymorphism and state the language of the example.

3. Answer the following:

(a) (12) Call-by-reference allows us to write some procedures that can not be written using call-by-value. Give an example of one such procedure.

(b) (9) Consider the following program segment in SCHEME-like syntax and consider that it uses call-by-reference for parameter passing.

\[
\begin{align*}
\text{let } b & = 3; \\
p & = \text{proc } (x,y) \\
& \text{begin} \\
& \hspace{1cm} x := 4; \\
& \hspace{1cm} \text{print } y \\
& \hspace{1cm} \text{end} \\
& \text{in } p(b,b); \\
\end{align*}
\]

What would be printed by this program segment?

(c) (9) Consider the following program segment in SCHEME-like syntax and consider that it uses call-by-name convention for parameter passing.

\[
\begin{align*}
\text{define } p & = \text{proc } (x) \\
& \text{begin} \\
& \hspace{1cm} i := 1; \ x := 2 \\
& \hspace{1cm} \text{end}; \\
\text{define } i & := 0; \\
\text{letarray } a[2] \\
\text{in } \text{begin} \\
& \hspace{1cm} a[0] := 1; \\
& \hspace{1cm} p(a[i]); \\
& \hspace{1cm} \text{writeln}(a[0], a[1]) \\
& \hspace{1cm} \text{end}; \\
\end{align*}
\]

What would be printed by this program segment?
4. Answer the following in the context of a LISP environment:

(a) (10) Show the cons-node data structures for the following S-expressions:
   i. (a (b c (d e (f)))))
   ii. (((a)) ((b c d)))

(b) (10) Describe the structure and a possible use for the *property list* in LISP.

(c) (10) Consider the following session with a LISP interpreter:

-> (define f (x) (begin (set a (+ a x)) a))
-> (set a 0)
   WHAT IS THE RESPONSE HERE?
-> (f 6)
   WHAT IS THE RESPONSE HERE?
-> (f 6)
   WHAT IS THE RESPONSE HERE?

What do you find peculiar about the above responses? What is the name given to this peculiarity?
Name: ..............................................

Computer Science Department
Master's Exam: Organization of Programming Languages
Spring 1993
Date: Apr. 3rd, 1993

Read these before starting:

1. Do any four of the following five problems.

2. All questions carry equal points.

3. Show all the work that you do for obtaining your answers.

1. Consider the following definitions in ML syntax. For the function defined in each part, what type would be inferred for the function by the ML type-checker. If the type-checker fails to find a type for any of these functions, describe why a consistent type description could not be found.

   (a) val add = fn x : int ⇒ (fn y ⇒ x + y);

   (b) fun find pred [] = false |
       find pred (a::d) = if pred a then true else find pred d;

   (c) fun find pred [] = false |
       find pred (a::d) = if pred a then true else find pred (hd d);

   (d) fun lt ((i1 : int, j1 : int), (i2, j2)) = i1 < i2 orelse
       i1 = i2 andalso j1 < j2;
2. Consider the following expressions of a LISP-like language.

-> (set s 3)
-> (set y 7)
-> (set x 1)
-> (set w 4)
-> (define h (y s) (+ (* s w) (+ x y)))
-> (define g (x s) (+ s (h (- x w) y)))
-> (define f (x y) (+ (h x s) (g y w)))

What will be the output for the expression (f 7 2) when

(a) static scoping is used?
(b) dynamic scoping is used?

3. In SCHEME language, the composition of two functions f and g may be defined as :

(define compose
  (lambda (f g)
    (lambda (x)
      (f (g x))))))

Also, the arity of a procedure is the number of arguments that it takes. Most functions are defined for a fixed arity. The arity of the function “compose” above is two because it accepts two inputs f and g. The effect of a variable arity in a function is achieved in the following procedure for addition by defining the argument to be x and then binding it to a list containing a variable number of elements :

(define plus (lambda (x)
  (if (null? (cdr x))
    (lambda (y) (+ (car x) y))
    (apply + x))))

Answer the following questions :

(a) What will be returned in response to the expression :
   ((compose car cdr) '(a b c d))

(b) What will be returned in response to the expressions :
   (plus (1 2))
   (plus 1 2)

(c) Define a version of compose that takes either two or three procedures of one argument and composes them. The composition of two function is done as in the above stated compose function and the composition for three functions is done as follows :
   (compose f g h) \(\Rightarrow\) (compose f (compose g h))
Read these before starting:

1. Do any four of the following six problems.
2. All questions carry equal points.
3. Show all the work that you do for obtaining your answers.

1. Answer the following parts:

(a) In Pascal two variables are considered to have the same type in some cases if the variables are declared using the same type name as described in the type statements (it is called name-equivalence); in other cases they have the same type if the declared types have the same structure (structural equivalence). Assume that Name equivalence is used for determining if formal and actual parameters in subprogram calls have the same type and structural equivalence is used in all other situations.

Write a Pascal program segment containing a function named sumarr. This function has an array of ten integers as its formal parameter and returns the sum of these ten integers. Include in the main program a segment which reads ten integers into the array and then invokes the function sumarr to compute their sum.

(b) Name the language in which the type-checking, in your view, is the most lenient.
(c) Name the language in which the type-checking, in your view, is the strictest.
(d) What is Polymorphism? (less than 75 words)
(e) Is polymorphism at all possible in PASCAL? If yes, how?
(f) Write a small function in either LISP or SCHEME to show polymorphism capability of the language? Explain in less than 75 words why it is polymorphic.
2. Consider the following expressions of a LISP-like language.

\[
\begin{align*}
\text{-> (set s 2)} \\
\text{-> (set y 5)} \\
\text{-> (set x 1)} \\
\text{-> (define f (x y) (+ (h x s) (g y s)))} \\
\text{-> (define h (y s) (+ s (+ x y)))} \\
\text{-> (define g (x s) (+ s (h (- x 1) y)))}
\end{align*}
\]

What will be the output for the expression (f 3 5) if

(a) dynamic scoping is used?

(b) static scoping is used?

3. Consider the following SCHEME expressions.

\[
\begin{align*}
\text{(set mymapcar (lambda (f) (lambda (c) (lambda (l) \\
\text{\hspace{1cm} (if (null? l) '() \\
\text{\hspace{2cm} (if (c (car l)) (cons f (car l)) \((\text{mymapcar f} c) (\text{cdr l})\)) \\
\text{\hspace{2cm} (\text{mymapcar f} c) (\text{cdr l})\))))))}
\end{align*}
\]

(a) Write the needed SCHEME expressions and function definitions that use mymapcar to construct a function m2 such that when a list of integers is passed to m2 it returns a list containing the squares of all those integers of the original list that are greater than 0. For example,

\[
\begin{align*}
\text{-> (m2 '(0 2 -6 8 7 -2 4))} \\
\text{(4 64 49 16)}
\end{align*}
\]

(b) We have a situation in which different f-functions need to be applied to the same pairs of l-lists and c-functions. We would like to generate a function in which l and c have been fixed but f can be accepted as a parameter. Modify the above definition of mymapcar to suit this need and using it generate the function m3 in which l = (1 2 5 3 8) and (c x) returns true only when x is greater than 2 in value.
4. In SMALLTALK-80, a part of the class hierarchy looks like this:

```
Object
  Magnitude
    Date Time Number
      Integer Float Fraction
  SmallInteger
```

(For defining classes and functions within them, use syntax structure of any object-oriented language familiar to you.)

(a) Define class Magnitude by writing methods for performing the >, >=, <=, and between (determine if the receiver is between the first and the second arguments) operations. Messages < and +1 should be defined as being subclass responsibilities.

(b) Define Date as a subclass of Magnitude. A Date is given by a month, day, and year. You need to define only the method +1 in this class. This message should properly account for the different lengths of the months and for leap years every fourth year.

(c) Show how a message may be sent to an object of type date to advance it by one day.

5. Consider the following program segment:

```
procedure swap (var x : integer; var y : integer);
  var z : integer;
  begin
    z := x; x := y; y := z;
  end swap;
```

(a) Assume that all parameters are passed using call-by-reference mechanism. Also assume that an array A of 100 integers is so initialized that for all i, A[i] = i + 5. What changes will be made in the array A by the following sequence of program statements?

```
i := 5;
swap(i, A[i]);
swap(i, A[i]);
```

(b) Assume that integers are always passed-by-value and array elements are always passed-by-reference. Now repeat the previous question with this assumption.

(c) Assume that all parameters are passed-by-value-result. Repeat part (a) with this assumption.
6. In the syntactic representation of a list in the usual list notation, the terminating CDR pointer to NIL is implicit, e.g.,

\[ \text{NIL} \]

\[ \text{NIL} \]

is written as \((A B C)\). Occasionally it is desirable to allow the last element of a list to have a CDR pointer to an atom other than NIL. In this case an alternative notation called the DOT-notation is used. For example, \((A)\) is written as \((A . \text{NIL})\), \((A B)\) is written \((A . (B . \text{NIL}))\), and, \(((A B) C)\) is written as \(((A . (B . \text{NIL})) . (C . \text{NIL}))\).

(a) Write the following in DOT-notation:

i. \((A (B C))\)

ii. \(((A)) (B (C D))\)

(b) Show the cons-node structure for the list \(((A . (B . C)) . ((E . (F . \text{NIL})) . \text{NIL}))\).
Read This before Starting:

Do four out of the following six problems.

1. Parameter Passing (25%)
   1.1. What is the output of the following Pascal program?

   ```pascal
   program m;
   var i: integer;

   procedure p(var j: integer);
   begin i:=20; j:=i+j; i:=i+j end;

   begin i:=10; p(i); WriteLn(i) end.
   Answer: ________________________________
   ```

   1.2. What would the output from the program in 1.1 be, if the parameter were passed by value?

   Answer: ________________________________

   1.3. What would the output from the program in 1.1 be, if the parameter were passed by value-result?

   Answer: ________________________________
2. Scope Rules (25%)

2.1. What is the output from the following Pascal program?

```pascal
program m;
  procedure a(x: integer);
    procedure b(procedure p);
      procedure c(x: integer);
        begin p end;
        begin c(1) end;
      end;
    end;
  end;
  begin b(d) end;
  begin a(2) end.
end.
```

Answer: ________________________________

2.2. What would the output from the program in 2.1 be, if the scope rule were dynamic?

Answer: ________________________________

2.3. The renaming principle says that the computation set up by a program is unaffected if local variables in the source text are consistently renamed. Formal parameters are considered as local variables. Using the program in 2.1, demonstrate that the renaming principle is violated under one of the two scope rules (static and dynamic).

2.4. Explain why in Smalltalk, the static scope rule (using Kamin's terminology) cannot be implemented for messages.
3. Scheme (25%)

3.1. The following is a Scheme program segment. Recall that in Scheme, a program is a sequence of expressions, and every expression has a value, which may be a closure.

\[
\text{(set grow (lambda (n) (lambda () (set x (+ 1 n)))))}
\]
\[
\text{(set count (grow 0))}
\]

What is the value of count after this segment is executed?

3.2. Suppose the next two expressions after the program segment in 3.1 are

\[
\text{(count)}
\]
\[
\text{(count)}
\]

What are the evaluation results of these expressions? What is the value of count after these expressions are evaluated?

3.3. Suppose the next two expressions after the program segment in 3.1 are

\[
\text{(set re-count (lambda (n) (count)))}
\]
\[
\text{(re-count 7)}
\]

What is the evaluation of the second of these expressions, (re-count 7)?

3.4. What would the value of (re-count 7) in 3.3 be, if the scope rule were dynamic? Why?
4. Implementation (25%)  

4.1. A closure is a function (procedure) plus an environment. Describe how the environment can be implemented when the static scope rule is used. Describe how a closure can be implemented.

4.2. Suppose a stack of activation records is used to implement Pascal. When a function (procedure) \( F \) is passed as a parameter to another function (procedure) \( G \), how the closure of \( F \) (its definition and its definition environment) is passed to the activation record of \( G \)? How this information is used later when \( F \) is called and an activation record is created for it?

4.3. Explain why Scheme's activation records cannot be implemented as a stack.
5. \textit{\textindex{\LaTeX}{\Lambda-Calculus} (25\%)}

5.1. Write down the \textit{Y-combinator}.

5.2. The \textit{Y-combinator} generates the fixed point of its argument. Which of the following equalities illustrates this property of the \textit{Y-combinator}? Assume the \textit{Y-combinator} is denoted with \textit{Y}.

\begin{itemize}
  \item [a.] \( \epsilon = (\textit{Y} \varepsilon) \) for all \( \epsilon \).
  \item [b.] \( \textit{Y} = (\varepsilon \textit{Y}) \) for all \( \epsilon \).
  \item [c.] \( (\textit{Y} \varepsilon) = (\varepsilon (\textit{Y} \varepsilon)) \) for all \( \epsilon \).
  \item [d.] \( (\varepsilon \textit{Y}) = (\textit{Y} (\varepsilon \textit{Y})) \) for all \( \epsilon \).
\end{itemize}

5.3. Describe the relationships between iteration, recursion, and the fixed-point generator (the \textit{Y-combinator} is an example). Give an example showing how the \textit{Y-combinator} can be used to eliminate recursion.
6. List and Functional Programming (25%)  

The following is the description of a limited version of Scheme (a kind of pure lisp with static scope). The value space includes names, lists, lambda expressions (closures). The operations are limited to if, null?, car, cdr, cons, mapcar, append, and eq. No while, no begin, and set is only allowed on the top level to give names to functions. Using this language, define the function permutation that takes a list as its argument and returns a list of all permutations of the argument. For example, the value of (permutation '(a b c)) is ((a b c) (a c b) (b a c) (b c a) (c a b) (c b a)). The order of the permutations in the list is insignificant.
Programming Languages Exam

NAME ____________________________

January 6, 1990

Do 3 of the following 4 problems.

Note also that, if you choose to do problem 4, you will need to choose further — among the languages LISP, ML, and Smalltalk.

Note: The test is long; we expect you will not be able to finish all 3 problems.
1. PARAMETER PASSING

Parts (a)-(b) deal with the following program (written in pseudo-Pascal):

```pascal
program Awful(input, output);
var x: integer;
procedure Twist(x,y,z:integer);
begin
  x := x + z;
  y := x + y + z
  writeln(x, ',', y, ', ', z);
end;
begin
  x := 3;
  Twist(x,x,x+x);
  writeln(x)
end.
```

(a) What is the output if the expression \(x+x\) is passed by value and the other parameters are passed by reference (i.e., by address)?

(b) What is the output if the parameters are passed by value/result (i.e., copy/restore linkage is used)?
(c) Suppose, in the subprogram below (written in pseudo-Pascal), parameters are passed by name.

```
function S(cntr, start, finish: integer; E: real): real;
  var T: real;
begin
  T := 0;
  for cntr := start to finish do
    T := T + E;
  S := T
end;
```

What does the call `x := S(i,1,n,W[i])` set `x` to? Explain your answer.
2. SUBROUTINE ACTIVATION

(a) Describe what needs to be stored in the stack of activation records for a Pascal procedure call. (Remember that Pascal is a *statically scoped* language that allows recursion. Ignore function and procedure parameters for now.)

(b) Why are the activation records stored in a stack?
(c) What would need to be added if functionals were added to the language - functions which return other functions. (Hint: the extra difficulty is to maintain static scoping on the function which is returned - to keep from losing its environment of definition.)
3. \( \lambda \) CALCULUS

(a) Does the reduction of the following pure lambda term terminate? If so, what is the resultant normal form?

\[
(\lambda x. (\lambda y. x(xy))) (\lambda x. (\lambda y. x(xy)))
\]

(b) The following are the \( \delta \)-rules for an applied lambda calculus:

\[
\begin{align*}
\text{mix red blue} & \quad \xrightarrow{\delta} \quad \text{yellow} \\
\text{mix blue yellow} & \quad \xrightarrow{\delta} \quad \text{red} \\
\text{mix yellow red} & \quad \xrightarrow{\delta} \quad \text{blue} \\
\text{mix } N \ N & \quad \xrightarrow{\delta} \quad N \\
\text{mix } M \ N & \quad \xrightarrow{\delta} \quad \text{mix } N \ M \\
\text{if } \text{red } M \ N \ P & \quad \xrightarrow{\delta} \quad M \\
\text{if } \text{blue } M \ N \ P & \quad \xrightarrow{\delta} \quad N \\
\text{if } \text{yellow } M \ N \ P & \quad \xrightarrow{\delta} \quad P \\
\text{fix } M & \quad \xrightarrow{\delta} \quad M(\text{fix } M)
\end{align*}
\]

What is the reduction result of the following term?

\[
(\text{fix } \lambda f. \lambda x y. \text{if } x y (f(\text{mix } x \text{ red})(\text{mix } y \text{ blue}))) \text{ yellow red}
\]