CS 324 – Language Processors

END-SEMESTER EXAMINATION

DATE: 30th April 2003 | **Weightage: 50%**
Maximum Marks: 100 | **Open Notes Exam**

1. Consider the Pascal program and its compiled code shown below:

```
type  Lptr = ^List;
      List = record
               val  : integer;
               next : Lptr
             end;
var   olist: Lptr

procedure driver;
  var ilist : Lptr;
procedure revlist(ilist : Lptr;
                  var olist: Lptr);
  var temp : Lptr;
  begin  (* revlist *)
        if (ilist = nil) then
        else
          begin
            temp := ilist;
            ilist := ilist^.next;
            temp^.next := olist;
            olist := temp;
            revlist(ilist, olist);
          end
    end; (* revlist *)
begin  (* driver *)
  (* initialise ilist*)
  olist := nil;
  revlist(ilist, olist);
end; (* driver *)

begin (* main *)
  driver
end. (* main *)
```

```
_REVLIST:
   pushl   %ebp
   movl    %esp,%ebp
   subl    $4,%esp
   movl    12(%ebp),%eax
   cmpl    $0,%eax
   je      .L7
   jmp     .L8
.L8:
   movl    12(%ebp),%edi
   movl    %edi,-4(%ebp)
   movl    12(%ebp),%eax
   movl    2(%eax),%edi
   movl    %edi,12(%ebp)
   movl    -4(%ebp),%eax
   movl    16(%ebp),%edx
   movl    (%edx),%edi
   movl    %edi,2(%eax)
   movl    16(%ebp),%eax
   movl    -4(%ebp),%edi
   movl    %edi,(%eax)
   movl    16(%ebp),%eax
   pushl    %eax
   pushl    12(%ebp)
   pushl    8(%ebp)
   call    _REVLIST
.L7:
   leave
   ret     $8

_DRIVER:
   pushl   %ebp
   movl    %esp,%ebp
   subl    $8,%esp
   movl    $0,_OLIST
   pushl $_OLIST
   pushl    -4(%ebp)
   pushl    %ebp
   call     _REVLIST
.L23:
   leave
   ret

main:
   pushl    %ebp
   movl     %esp,%ebp
   call     _DRIVER
   leave
   ret
.bss
.lcomm   _OLIST,4
```

Now answer the following questions:

(a) i. Write a grammar to represent all the syntactic features in the body of the procedure `revlist` in their full generality. Examples of such syntactic features are compound statement and if statement.
To keep things simple, you do not have to expand anything under an assignment statement or an expression. Let them be represented by the terminal symbols `ass` and `exp`.

$body \rightarrow compoundstatement$
$\ldots$ complete this part
$assignment \rightarrow$ `ass`
$expression \rightarrow$ `exp`

ii. Draw the parse tree for the body of `revlist` according to your grammar.

iii. Ignoring ambiguities, is your grammar LR(0)? Clearly explain your answer.

(b) Draw the symboltable for the entire program. Build on top of the following structure.

Information about procedure →	*name*	*size of local data*	*nesting level*	*argtype*	*rettype*		
information about variables in procedure →	*name*	*var/ proc*	*type*	*param/ local*	*width*	*var/value*	*address*
information about procedures in procedure →	*name*	*var/ proc*	*pointer to symboltable*				

The symboltable information should be consistent with the generated code.

(c) Show the contents of the activation record for the procedures `driver` and `revlist`. Showing the contents means specifying for each item of interest, its displacement with respect to the base pointer.

(d) Describe how the statement `temp^.next := olist` will be type checked. Enumerate the steps.

(e) Show the fragment of code generated for the same statement.

(f) If a (meaningless) statement `olist^.next := ilist` had been inserted in `revlist`, what code would have been generated for it? $(9 + 4 + 8 + 4 + 4 + 6 = 35\ marks)$

2. In this question assume that as code is being generated, it is being dumped into a global data structure. Use the familiar functions gen (or $emit$), $backpatch$, $nextstat$ etc. You can also use other functions, but explain what they do.

(a) Give *S-attributed definitions* to generate 3-address code for the C-style expression grammar shown below:

$$E \longrightarrow id \mid ++id \mid id{++} \mid E + E \mid E \mathbin{?} E : E$$

(b) Give L-attributed definitions to generate code for `case` statements given by the following grammar.

$$
\begin{array}{lcl}
\mathit{casestat} & \longrightarrow & \texttt{case}\ \mathit{exp}\ \texttt{of begin}\ \mathit{sclauses}\ \texttt{end} \\
\mathit{sclauses} & \longrightarrow & \mathit{sclause}\ \mathit{sclauses} \mid \mathit{sclause} \\
\mathit{sclause} & \longrightarrow & \mathit{const} :\ \mathit{statement}
\end{array}
$$

The nature of the code that we would like to generate is illustrated below:

```
case  E  of                  1. code to evaluate E in t
  begin                      2. go to 7
    V1: S1                   3. code for S1
    V2: S2                   4. goto 9
  end                        5. code for S2
                             6. goto 9
                             7. if t = V1 goto 3
                             8. if t = V2 goto 5
                             9. ...
```

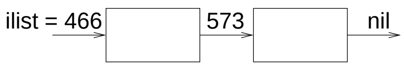
Assume that code has already been generated for expressions and other statements, and that statements *are not required to* produce code containing unfilled gotos.

$(10 + 10 = 20\ marks)$

3. (a) Use the program outline shown below to illustrate the difficulty of using displays for accessing the environment, when procedures are passed as parameters

```
procedure P(Q)
  begin
    Q
  end
procedure S
  procedure T
    begin (* T *)
    end   (* T *)
  begin (* S *)
    P(T)
  end   (* S *)
begin (* main *)
  S
end (* main *)
```

(b) Assuming `ilist` in the first question is initialised to the structure shown below:



draw the complete stack of activation records just after entering `revlist` with the value of `ilist` as `nil`. $(6 + 9 = 15\ marks)$

4. (a) What is the minimum number of registers required by Aho-Johnson algorithm to produce code (i.e. not get blocked)?

(b) Consider a machine model with 3 general purpose registers and instructions shown below with their costs.

$R_i \leftarrow R_j\ op\ R_k$, i, j and k are distinct	cost – 1
$R \leftarrow c$	cost – 1
$R \leftarrow m$	cost – 1
$R \leftarrow ind(R)$	cost – 1
$R \leftarrow ind(R + m)$	cost – 3
$m \leftarrow R$	cost – 1

Now consider the expression $((a * b) * (ind(c + 1))) * (ind(1 + 2))$.

i. Calculate the array of costs for each node in the tree for the expression above.

ii. Encircle the least cost element at the root, and the element at each node which gave rise to the least cost. $(3 + 12 = 15\ marks)$

5. (a) This question is related to Sethi-Ullman algorithm. If the operator in a certain node in the expression tree is commutative, we may want to swap its left and right subtrees to bring down the register requirement of the tree.

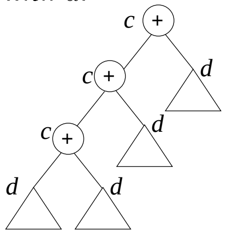
Characterise the nodes where doing such a thing may lead to a reduction in the register requirement.

(b) Next, we want to take advantage of associative properties of operators also. Define a *cluster* as a maximal set of nodes such that:

i. Each node in the set is labeled by the same associative operator.

ii. The branches between the nodes in the set form a tree.

Also define the *descendants of a cluster* as the descendants of the leaves of the cluster. In the figure shown below, the nodes forming a cluster are marked with c whereas the descendant nodes are marked with d.



Let the descendants of a cluster be ordered by their labels $l_1 \dots l_n$, so that $l_i \geq l_{i+1}$.

i. How will you reshape (by changing association) the cluster so that the label at the root of the reshaped cluster is minimized? Draw a diagram to explain your answer.

ii. In terms of $l_1 \dots l_n$, what is the minimum label that can be achieved by reshaping? $(5 + 10 = 15\ marks)$