CS302-2023-24: End-Sem Exam

Duration : 3 Hours Max Marks : 130

- No explanations will be provided. In case of a doubt, make suitable assumptions and justify.
- You are allowed to use two A4 sheet of hand-written (not xeroxed) notes.
- Please write only the final answers in your answer sheet. Rough work may be done at the back. Extra material (or multiple answers of the same questions) will receive negative marks.

(1) Consider the following lex script and an unrelated transition table. **(5+12=17)**

(a) What is the output for the input abdc? How many times is a transition on letter b made? Why?

(b) Encode the given transition table using the most compact four arrays representation. Assume that the codes for letters p to t are from 0 to 4, respectively. If s_j is the default state for s_k, the entries for s_j must precede those of s_k.

```
%%
abc { printf ("ABC\n"); }
a   { printf ("A\n"); }
b   { printf ("B\n"); }
c   { printf ("C\n"); }
.   { ; }
```

	p	q	r	s	t
0	1	2	2	2	3
1	2	2	2	2	1
2	2	2	2	2	
3	2	2	1	2	3

Answer:

(a) The output of the scanner is

```
A
B
C
```

A transition on b takes place twice. The first time when ab is read expecting to see "abc", the suffix bc is not consumed because only a matches a token. Then b is read again for the next token.

Evaluation rubrics: 3 marks for completely correct output and 2 marks for number of transitions on b. Any mistake in any part fetches 0 marks for it.

(b) The four arrays representation is as shown below

	Base	Default
0	4	2
1	2	2
2	0	
3	3	2

	Next	Check
0	2	2
1	2	2
2	2	2
3	2	2
4	1	0
5	1	3
6	1	1
7	3	3
7	3	0

Any other sequence would require more entries.

Evaluation rubrics: 12 marks for a completely correct minimal encoding. 0 marks for correct but non-minimal encoding or incorrect encoding.

Accepted variation: Some students have used equivalence and have combined the columns of q and s and have assigned the same code to q and s. With this change, if their four-arrays representation is consistent with other requirements, we have given them full marks.

Some students have explicitly made a blank entry for transition on t in state 2. This is spurious and violates mainimality.

---

(2) Consider the following yacc script and the associated parsing table obtained after resolving conflicts by describing precedences and associativities which are missing from the yacc script. **(14+6+4=24)**

(a) Reconstruct the DFA of the LR(0) sets of items from the parsing table.

(b) Explain all conflicts by using the relevant items from the states in which conflicts exist.

(c) Complete the script by writing the minimum number of precedences and associativities in yacc syntax. Choose left or right associativity for a symbol only if the symbol is associative as indicated by relevant items in some state. Do not explain your choices.

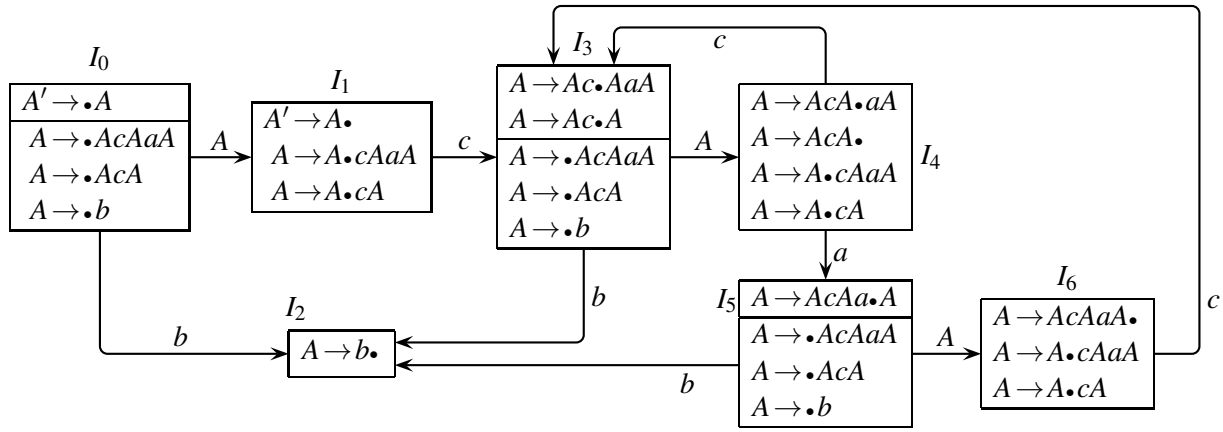
```
%token a b c
%%
A : A c A a A
  | A c A
  | b
```

1 $A \rightarrow A\ c\ A\ a\ A$
2 $A \rightarrow A\ c\ A$
3 $A \rightarrow b$

	a	b	c	\$	A
0		$s2$			$c1$
1			$s3$	Accept	
2	$r3$		$r3$	$r3$	
3		$s2$			$c4$
4	$s5$		$s3$	$r2$	
5		$s2$			$c6$
6	$r1$		$s3$	$r1$	

Answer:

(a) The DFA of LR(0) items is as shown below.



Thus, all possible actions and the rejected actions are as shown below.

	a	b	c	\$	A
0		$s2$			$c1$
1			$s3$	Accept	
2	$r3$		$r3$	$r3$	
3		$s2$			$c4$
4	$s5/\not r2$		$s3/\not r2$	$r2$	
5		$s2$			$c6$
6	$r1$		$s3/\not r1$	$r1$	

Evaluation rubrics: There are 7 states and 2 marks for completely correct state. A correct state involves all in and out transitions are correct, and items are correct, no item is missed. If there is any error in any state, 0 marks.

(b) The conflicts can be explained by considering $FOLLOW(A) = \{a, c, \$\}$.

- State 4 has two conflicts.
  - Consider items $\boxed{A \rightarrow AcA\bullet aA}$ and $\boxed{A \rightarrow AcA\bullet}$. If we get a in the input, we can shift and go to state 5 or reduce by $\boxed{A \rightarrow AcA\bullet}$.
  - Consider items $\boxed{A \rightarrow A\bullet cA}$ and $\boxed{A \rightarrow AcA\bullet}$. If we get c in the input, we can shift and go to state 3 or reduce by $\boxed{A \rightarrow AcA\bullet}$.
- State 6 has a conflict on c. If the next character is c we can shift it and go to state 5 or reduce by $\boxed{A \rightarrow AcAaA\bullet}$.

Evaluation rubrics: There are three conflicts. A correct description of each conflict (with clear explanation), fetches 2 marks. If there is no description or if the description is incomplete or unclear,0 marks.

(c) The precedences and associativities are used by yacc using the following rules.

β t_1 α	$t_2 \ldots \$$	Here the parsing stack contains terminal t_1 possibly covered by a some grammar symbols consisting solely of non-terminals (i.e., there is an RHS of a production with the suffix $t_1\beta$). The next token in the input is t_2.

If the parsing table indicates the possibility of both shifting t_2 on the stack as well as reducing by $X \to \beta' t_1 \beta$, then the conflict is resolved using the precedences and associativities of t_1 and t_2 as follows:

- If $predecence(t_1) < predecence(t_2)$, then t_2 is shifted.
- If $predecence(t_1) > predecence(t_2)$, then reduction by $X \to \beta' t_1 \beta$ is performed.
- If $predecence(t_1) = predecence(t_2)$, then
  - If they are left-associative, then reduction is performed.
  - If they are right-associative, then t_2 is shifted.
  - If they are non-associative, it is an error.

  Note that yacc syntax does not allow specification of different associativities for terminals with the same precedence (because they must appear on the same line).

It is clear from the parsing table shown above that each conflict is resolved by choosing a shift action and avoiding reduce action.

- State 4 contains the following complete item $\boxed{A \to AcA\bullet}$.
  - For input symbol a, since we want a shift, a must have a higher precedence than c.
  - For input symbol c, since we want a shift, c must be right associative.
- State 6 contains the following complete item $\boxed{A \to AcAaA\bullet}$ so we need to compare the precedences of a and c. We want a shift, which is enabled by giving higher precedence to c.

In state 6, there is no transition on a implying that a is not associative (i.e. we never have aAa appearing in any right sentential form).

The completed yacc script is:

```
%token a b c
%right c
%nonassoc a
%%
A : A c A a A
  | A c A
  | b
```

Evaluation rubrics: Only the completed yacc script is expected in the answer; the blue text is only an explanation of the expected answer.

2 marks per correct specification; 0 for any mistake. Deduct one mark if precedence of b is provided. In case a is specified to be left or right associative, 1 mark will be given instead of 2.

Accepted variation. The specification of associativities introduces a token so if "`% token`" does not introduce a and c, it is fine.

Some people have complicated the answer using "`%prec`" directive by introducing additional tokens. This is technically correct but is a contrived and complicated answer for something that should have been much simpler. This is an abuse of the "`%prec`" directive—it is meant to be used for the situation when the same symbol needs to be given two different precedences in different rules (such as unary and binary minus). We did not find any correct usage of "`%prec`".

Rejected. Some students have claimed the answer to be wrong pointing out the given state number does not have a conflict in the yacc generated parser. I appreciate the enthusiasm of checking with yacc but the conclusion drawn is wrong. This is because the bison based yacc has a different numbering scheme of states. In particular, bison based yacc also shows an explicit shift action on "$". The classical yacc does not do so. The numbering

given in the question is consistent with the classical yacc, with the DFA of LR(0) items given in the answer, and with what we have studied in the class. In any case, the numbering does not matter because a parsing table was given and your sets of items are required to be consistent with it.

---

(3) Consider the SDD for generating IR for array accesses using the usual attributes. **(7+4+10=21)**

$S \rightarrow id = E;$	$S.code = E.code \mid\mid gen(id.place, =, E.place)$
$E \rightarrow id$	$E.code = NULL; E.place = id.name$
$E \rightarrow num$	$E.code = NULL; E.place = num.val$
$E_1 \rightarrow E_2\ op\ E_3$	$E_1.place = getNewTemp()$ $E_1.code = E_2.code \mid\mid E_3.code \mid\mid gen(E_1.place, =, E_2.place, op, E_3.place)$
$E \rightarrow A$	$E.place = getNewTemp(); t_1 = getNewTemp()$ $E.code = A.code \mid\mid gen(t_1, =, A.offset \times width(A.name)) \mid\mid gen(E.place, =, A.name, [, t_1,])$
$A \rightarrow id\,[\,E\,]$	$A.name = id.name; A.ndim = 1$ $A.offset = E.place; A.code = E.code$
$A_1 \rightarrow A_2\,[\,E\,]$	$t_1 = getNewTemp(); A_1.offset = getNewTemp()$ $A_1.name = A_2.name; A_1.ndim = A_2.ndim + 1$ $c_1 = gen(t_1, =, A_2.offset \times dimlimit(A_1.name, A_1.ndim))$ $A_1.code = A_2.code \mid\mid E.code \mid\mid c_1 \mid\mid gen(A_1.offset, =, t_1, +, E.place)$

Consider the code fragment on the right. The size of integer is 4 and that of float is 8. The temporaries are numbered $t_0, t_1, t_2, \ldots, t_k$ and are generated in the order of visits to nodes in bottom-up parsing.

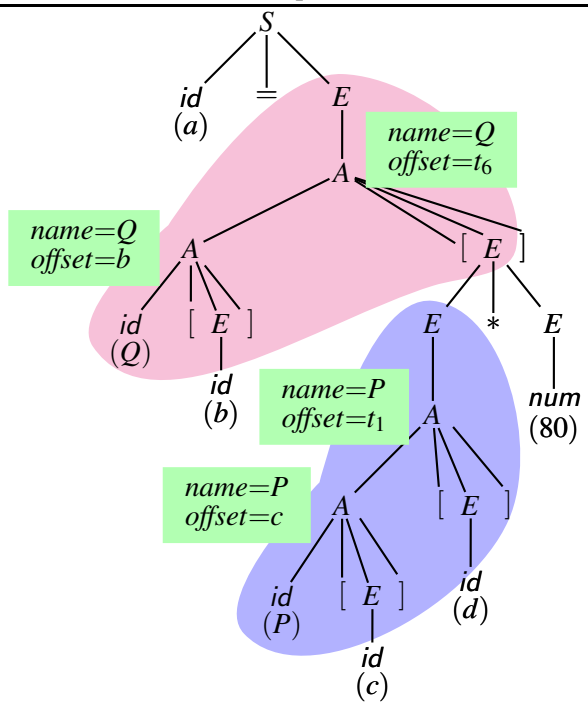
(a) Construct the parse tree for the assignment statement and show only the following attributes for the nodes representing A: $A.offset$ and $A.name$.

(b) Write the final generated code (in the order of bottom-up parsing).

```
int b, c, d, P[20][30];
float a, Q[40][10];
a = Q[b][P[c][d]*80];
```

Answer:

Parse tree with the required attributes of A



Code

$t_0 = c * 30$
$t_1 = t_0 + d$
$t_2 = t_1 * 4$
$t_3 = P[t_2]$
$t_4 = t_3 * 80$
$t_5 = b * 10$
$t_6 = t_5 + t_4$
$t_7 = t_6 * 8$
$t_8 = Q[t_7]$
$a = t_8$

Evaluation rubrics: 7 marks for the parse tree: 3 for the pink region and 4 for the blue region. Both the regions must be completely correct. Deduct 1 mark per error in any region. 4 marks for the attributes (shown in green background). 10 marks for the code. 1 mark per line provided each line is correct and in the correct position.

Accepted variations.

- The order of temporaries t_2 and t_3 should be swapped. Similarly the order of temporaries t_7 and t_8 should be swapped. This is because the order of actions in the rule $E \rightarrow A$ is such that the temporary for $E.place$ is

generated before the temporary in which the expression is evaluated; however it is used later in generating the code. This is an unfortunate side-effect of editing the question to remove a line ensure that the question paper fits in two pages!

- Some students have stated the temporary numbers from 1 instead of 0. If everything else is consistent with this choice, it will be treated correct.
- Some students have pointed out that the semicolon after the statement is missing from the answer and they have included it. This is fine. The rubrics reserve the marks for other important parts of the parse tree and none for semicolon, anyway.

If *id* or other attributes were missing in the coloured regions, 1 mark was deducted. Surprisingly, a large number of students did not draw the parse tree correctly which is shocking. We have given zero where the internal nodes are wrong, and deducted 1 mark per missing edge.

Rejected. The required attributes are not expressions or the statements containing expressions. They go in the generated code.

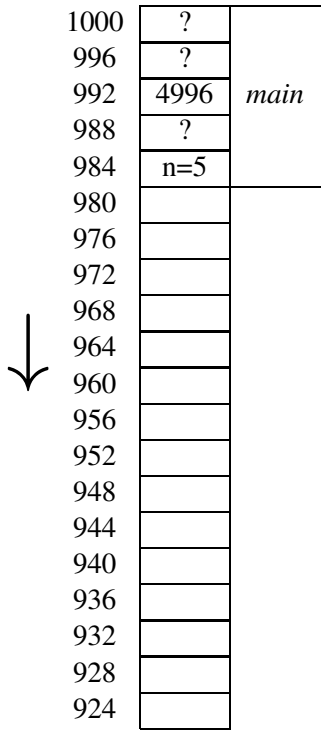
---

(4) Consider the following program and the given activation record structure. **(22)**

```
void main()
{
    int n;
    n=5;
    n=sum(n);
    cout << n;
}
```

```
int sum(int n)
{
    int r;
    if (n<=0) r=0;
    else r=n+sum(n-1);
    return r;
}
```

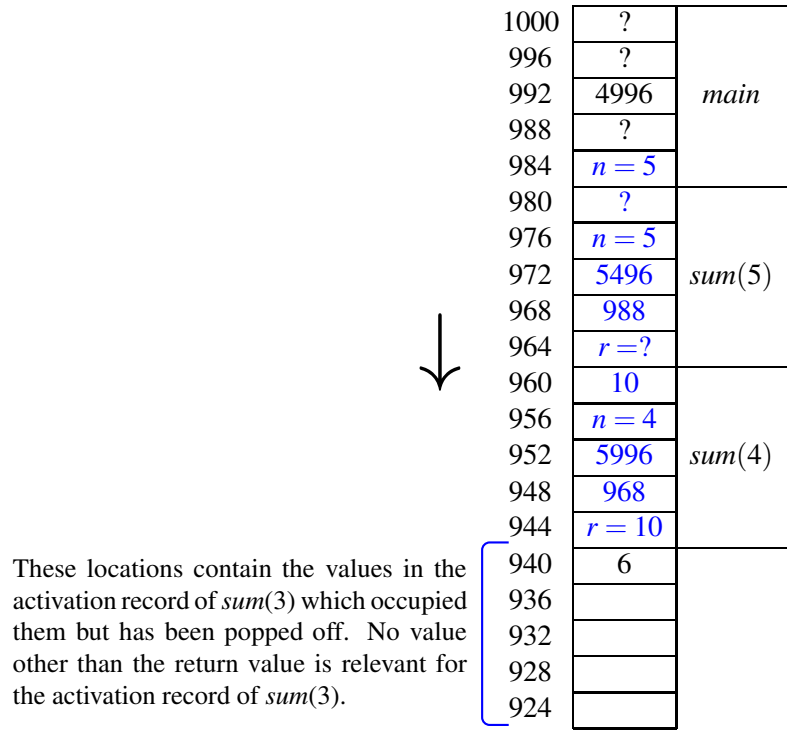
return value
param *n*
...
param 1
return address
control/dynamic link
local 1
...
local *m*



Fill up the contents of the locations in the given control stack at the time when the call to *sum*(4) is just about to return. The activation record of *main* has been shown. If the contents of a location are not available, fill up "?". Assume the following:

- Each location is a 4 byte word. Integers and stack pointers fit in a single word.
- The number to the left of a location is its address. If the address is 1000, then the content in the location is stored in four bytes from 1000 to 1003.
- Both the code memory and the stack grow downwards. Thus if a location has address 1000, the address of the "next" location is 996.
- The address of the control link field is the base of the activation record.
- The call to `main` is made at the address 5000, the call to `sum` in `main` is made at address 5500, whereas the recursive call to `sum` is made at address 6000.

Answer: The stack looks as shown below.



These locations contain the values in the activation record of *sum*(3) which occupied them but has been popped off. No value other than the return value is relevant for the activation record of *sum*(3).

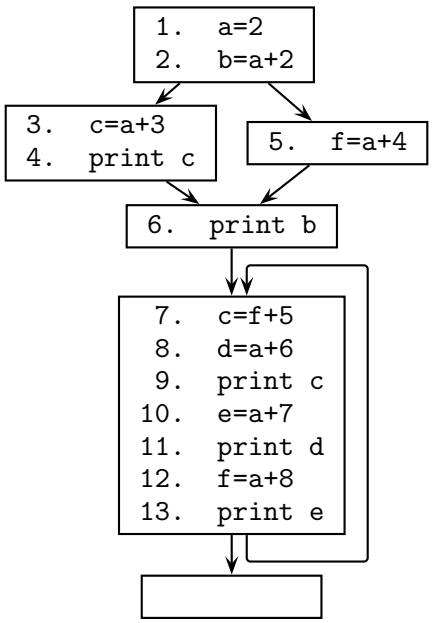
Evaluation rubrics: There are 10 entries to be filled in by the students (shown in blue). Each correct entry (i.e. correct value at the correct position) carries 2 marks. Labeling the activation records carries 1 mark each. It is fine if the label does not distinguish between the two activations of *sum*.

Accepted correction. The return addresses should be 4996 (instead of 5504), 5496 (instead of 5004), 5996 (instead of 6004).

Rejected. The argument about base being at the bottom of the activation record. Base is the address relative to which offsets are calculated. We have studied in the class that parameters have positive offsets and local variables have negative offsets. All of you have implemented this in sclp and are supposed to know this very well.

(5) Consider the given program for register allocation. **(6+6+3+6=21)**

Note that for a statement $x = y + 1$, although the statement belongs to the live ranges of both x and y, the live ranges do not interfere if there is no further use of y before being defined.



(a) Identify the live ranges in the program in terms of statement numbers. Use the following format:

Variable	Set of statements in the live range
a	$\{\ldots\}$
...	

(b) Draw the interference graph.

(c) Does Chaitin's method require any spilling with three colours? Justify. Do not show the colouring process nor the coloured interference graph.

(d) Colour the graph using Chaitin-Briggs' method using three colours: R_0, R_1, and R_2. Do not show the colouring process. Just show the coloured graph by annotating each node with a colour. If a node needs to be spilled, explain why it needs to be spilled.

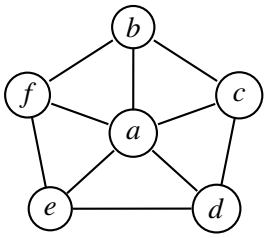
Answer:

(a) The live ranges are as follows.

Variable	Statements in the live range
a	$\{1,2,3,4,5,6,7,8,9,10,11,12,13\}$
b	$\{2,3,4,5,6\}$
c	$\{3,4,7,8,9\}$
d	$\{8,9,10,11\}$
e	$\{10,11,12,13\}$
f	$\{5,6,7,12,13\}$

Evaluation rubrics: Every completely correct live range gets 1 mark; any mistake (missing a statement, a spurious statement) leads to 0 marks for the live range.

(b) The interference graph is as follows

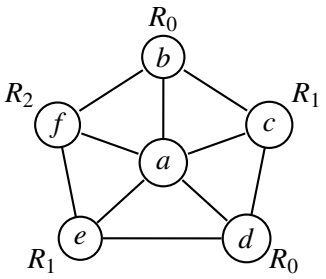


Evaluation rubrics: There are 6 nodes. Every completely correct node (with correct edges) gets 1 mark. This automatically means that missing a single edge (or adding a single spurious edge) causes loss of 2 marks.

(c) Since the degree of every node is at least 3 and we are given 3 colours, Chaitin's method cannot colour the graph without spilling a live range.

Evaluation rubrics: 3 marks if clearly stated with reason. 0 otherwise.

(d) Live range a needs to be spilled. The remaining ranges can be coloured with three colours in any order. For example,



Note that f cannot be coloured with R_0 because it interferes with b. The outer cycle has an odd number of nodes. If it had an even number of nodes, two registers would have sufficed for the outer cycle and we would not have to spill a.

Evaluation rubrics: Completely correct answer with justification gets 6 marks. 0 otherwise. The explanation in blue text is not expected in the answer.

Accepted variation. Any live range can be spilled.

Rejected. The live range of f begins at 1. Variable f is live at the start but has not been defined. A live range includes the statements from a definition of a variable to its last use. Here, there is no definition of f in node 1. The rationale behind this choice is that the value of a variable is loaded into a register at the start of the live range. In this case, since there is no definition of f, there is no value to be loaded in the register. Note that this is not an incorrect program because the condition in the node 2 could be such that the control may not be transferred to statements 3 and 4. In most cases, a compiler can do nothing about it, although in some cases, such unreachable code can be removed.

Exceptions that may be considered.

- If someone has stated clearly that f is considered to be a parameter, then this contention is tenable because the formal parameters are supposed to be defined.
- If someone has created two separate live ranges for variable c, it may be considered for partial marks subject to everything else being correct. As per the convention used in the class, by default, we begin with a single live range per variable and then splitting decides if multiple live ranges should be constructed.

If interference graph is wrong, parts (c) and (d) have been awarded zero because then you are not giving consistent colouring for the variables in the program.

---

(6) Consider expression $((a+b)*(c+d))/((e*f)+(g*h))$ for code generation using Sethi-Ullman algorithm using instructions given below on the right. Assume two registers R_0 and R_1. **(6+7+12=25)**

(a) Draw the expression tree.

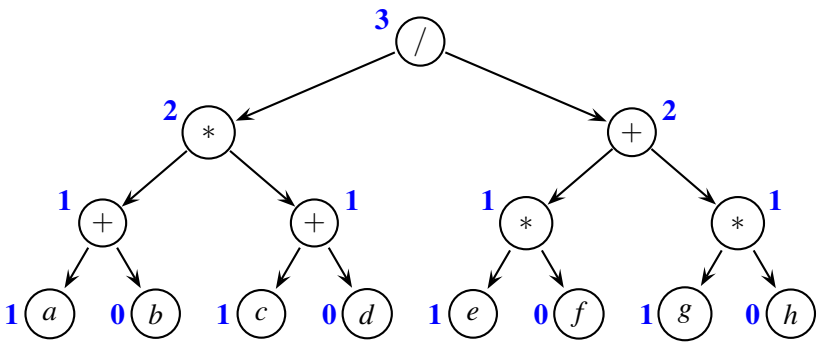
(b) Label the tree for its register requirements.

(c) Generate the code assuming rstack and tstack to be $[R_0, R_1]$ and $[T_0, T_1]$ where R_0 and T_0 are on the top.

$M \leftarrow R_i$
$R_i \leftarrow M$
$R_i \leftarrow R_i \ op \ R_j$
$R_i \leftarrow R_i \ op \ M$

Answer:

(a) [and (b)] The expression tree with its labels as as follows.



Evaluation rubrics: No partial marks. The expression tree is a very simple and basic concept and must be completely correct. If it is correct 6 marks, 0 otherwise. For labels, 1 mark for correct label of each internal node (if the error in one node percolates to parent node, the student looses marks for parent node too.)

(c) The generated code is as shown below.

$$
\begin{aligned}
R_0 &\leftarrow e \\
R_0 &\leftarrow R_0 * f \\
R_1 &\leftarrow g \\
R_1 &\leftarrow R_1 * h \\
R_0 &\leftarrow R_0 + R_1 \\
T_0 &\leftarrow R_0 \\
R_0 &\leftarrow a \\
R_0 &\leftarrow R_0 + b \\
R_1 &\leftarrow c \\
R_1 &\leftarrow R_1 + d \\
R_0 &\leftarrow R_0 * R_1 \\
R_0 &\leftarrow R_0 / T_0
\end{aligned}
$$

Evaluation rubrics: There are 13 statements. Every correct statement (in correct position, not moved up or down) gets 1 mark.

Some students drew a parse tree instead of expression tree. 2 marks were deducted. Some students drew parentheses in the expression tree. This is wrong because syntactic features do not appear in an AST unless they carry a semantics that is not represented by the structure and other nodes in the AST. Worse still, students assigned labels to parenthesis. In such cases 3 marks have been deducted.

Best Luck!