1 Lexing

a. What kinds of languages are regular expressions unable to recognize? Give a general principle and a specific example.

b. Write regular expressions for each of the following languages, or indicate that no such regular expression exists.
   
i) Strings over the alphabet \{x, y, z\} in which no x’s appear after the first y (if one exists).

   ii) Binary numbers that are multiples of 4.

   iii) Identifiers that start with a lowercase letter and then have 0 or more alphanumeric characters, ending in a number.

   iv) Strings over the alphabet \{a, b\} that have exactly as many a’s as b’s.

2 Parsing

a. Write a context-free grammar for each of the following languages.
   
i) Strings over the alphabet \{a, b\} that have exactly as many a’s as b’s.

   ii) Arithmetic expressions involving addition and subtraction, with no particular associativity or precedence.
iii) The same language, but where addition and subtraction are left-associative, and addition has higher precedence than subtraction.

iv) L-values, where an l-value is either the terminal symbol ID, l.ID where l is an l-value, or l[n] where l is an l-value and n is a number (you should define numbers as another nonterminal in your grammar).

b. Consider the following excerpt of a yacc grammar:

```
exp:
  ID
| NUM
| exp PLUS exp
| exp TIMES exp
| REF exp      // memory access

stm:
  ID EQ exp     // assignment
| REF exp EQ exp // store
| stm SEMI stm  // sequence
| IF exp LBRACE stm RBRACE ELSE LBRACE stm RBRACE
| IF exp LBRACE stm RBRACE

```

For each production, write an action that produces an appropriate AST node for that production. Assume the existence of appropriate node constructors for each type of node.
3 Semantic Analysis

Consider the following program:

```c
1: int x = 0;
2: int *y = malloc(sizeof(int));
3: while(x < 10){
4: int z = 0;
5: int y = x;
6: while(y < 10){
7: int z = z + y;
8: y = y + 1;
9: }
10: }
11: *y = x;
```

a. Write an environment/symbol table showing the type of each variable that is in scope at line 4.

b. Write an environment/symbol table showing the type of each variable that is in scope at line 6.

c. Write an environment/symbol table showing the type of each variable that is in scope at line 11.

4 Intermediate Representations

a. Name one difference between abstract syntax trees and intermediate representation trees.

b. Translate each of the following Tiger programs into IR trees. Assume that all variables are temporaries unless otherwise specified.

   i) a.field3[2] := 1, where a’s type is

   { field1 : int, field2 : string, field3 : array of int }
ii) if \( b = 0 \) then \( f(0, "a") \) else \( f(1, "b") \), where \( b \) is the second variable in the current stack frame.

iii) \( c := \text{myrec} \{ \text{field1} = 3, \text{field2} = "Hi", \text{field3} = \text{myarr} [ n ] \text{ of } 0 \} \) where \( \text{myrec} \) is the type from i) and \( \text{myarr} \) is defined as \( \text{type myarr = array of int} \)

c. Suppose we wanted to add a repeat-until loop to Tiger, so that \( \text{repeat e until c} \) executes the expression \( e \), checks whether the condition \( c \) is true, and if not repeats the process.

i) Implement the function \( T_{\text{repeat}} \) that translates an AST of the form \( \text{repeat e until c} \) into an IR tree, without calling any other translation functions. Use \( T_{\text{newlabel}} \) to create new labels, and \( T_{\langle \text{kind} \rangle} \) to create a new node of kind \( \text{kind} \).

\[
T_{\text{exp}} \ T_{\text{repeat}}(T_{\text{exp}} \ e, T_{\text{exp}} \ c)\}
\]

ii) Give a different implementation of \( T_{\text{repeat}} \) that makes an appropriate call to \( T_{\text{while}} \) instead of generating new labels and jumps. You may need to add statements onto the beginning or end of the return value of \( T_{\text{while}} \).
5 Instruction Selection

Suppose we have a target language with the following tiles, where \( d \) indicates a destination register, \( s \) indicates a source register, and \( c \) indicates a constant:

<table>
<thead>
<tr>
<th>instruction</th>
<th>tiles</th>
</tr>
</thead>
</table>
| addi \( d, s, c \) | Plus Plus Const Temp
|             | Const Const |
| add \( d, s_1, s_2 \) | Plus |
| lw \( d, c(s) \) | Mem Mem Mem Mem
|             | Mem Mem Mem |
|             | Plus Plus Const |

a. Which of the following are correct translations of the AST to the target language?

Assume that the register \( r_0 \) always contains the value 0.

i) \[
\text{addi } r_1, r_0, 2 \quad \text{add } r_2, b, r_1
\]

ii) \[
\text{lw } r_1, 2(b) \quad \text{addi } r_2, a, 1
\]

iii) \[
\text{addi } r_1, b, 2 \quad \text{add } r_3, r_1, r_2
\]

iv) \[
\text{addi } r_1, b, 0 \quad \text{lw } r_2, 2(r_1)
\]

b. Of the correct translations above, which represent optimal tilings?

6 Dataflow Analysis

The rules for reaching definition analysis are as follows:

If a node \( n \) assigns to a variable \( a \), then \( \text{gen}[n] = \{n\} \) and \( \text{kill}[n] \) is the set of all other nodes that define \( a \). Otherwise, \( \text{gen}[n] \) and \( \text{kill}[n] \) are both empty.

\[
\text{in}[n] = \bigcup_{n' \in \text{pred}(n)} \text{out}[n']
\]

\[
\text{out}[n] = \text{gen}[n] \cup (\text{in}[n] - \text{kill}[n])
\]

a. Is this a forward or backward dataflow analysis?
b. Consider the following CFG:

```
1: x = a
2: y = b
3: if (x = 0)
   4: y = y + z
   5: y = y - z
   6: z = y
```

Write the in and out sets for each node after the first iteration of the analysis.

c. Finish performing the analysis, and write the final in and out sets.
7 Register Allocation

a. Consider the following program, annotated with live variable information:

```plaintext
// live: {v, x}
u = v + 1
// live: {u, v, x}
w = u - v
// live: {u, w, x}
x = x + w
// live: {u, w, x}
y = u - w
// live: {x, y}
z = x + y
// live {z}
```

i) Draw the interference graph for the program.

ii) What is the smallest number of colors that can be used to color the graph without spilling? Explain why no smaller number of colors will be enough.

iii) Suppose you have machine registers r1, r2, and r3. Write an allocation of variables to registers such that no two adjacent variables have the same register, spilling if necessary.
8 Loop Analysis and Optimization

a. Consider the following control flow graph:

![Control Flow Graph]

i) What is the immediate dominator of node 8?

ii) Draw the dominator tree for the graph.

iii) List each of the natural loops in the graph, by identifying the header node and all the other nodes in the loop.

b. Consider the following program:

```plaintext
x = 0
test:
if (x > 8) goto done
z = 2 * x
v = a + 4
store(0, v + z)
x = x + 1
y = b + x
done:
```

i) Can the instruction \( v = a + 4 \) be hoisted outside of the loop without changing the behavior of the program? Why or why not?
ii) What is the basic induction variable of the loop? What are the derived induction variables in its family?

iii) Strength reduction on a derived induction variable $j$ has the following steps:
   
   i. Pick a new variable $j'$ and add an instruction $j' = j' + c$ at the end of the loop, where $c$ is the amount by which $j$ increases in each iteration.
   
   ii. Add an instruction $j' = c_0$ before the beginning of the loop, where $c_0$ is the initial value of $j$.
   
   iii. Replace the right-hand side of the assignment to $j$ with $j'$.

Perform strength reduction on one of the derived induction variables in the program, and show the resulting program.