NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING
EXAMINATION FOR
Mockkkkkk

CS4212 — Compiler Design

Mock Exam Paper Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. The examination paper contains **FOUR (4) questions** and comprises **ELEVEN (11) pages**.
2. The maximum attainable score is 50.
3. All questions must be attempted for the maximum score to be attained.
4. This is an OPEN BOOK exam.
5. Write all your answers in the space provided in this booklet.
6. **Please write your matriculation number below.**

MATRICULATION NUMBER: ____________________

(this portion is for the examiner's use only)

Question	Marks	Remark
Q1		
Q2		
Q3		
Q4		
Total		

Question 1 [8 marks]

Lexical Analysis

Consider the following three languages:

L_1 : strings over $\{a,b,c,d\}$ that contain the sequence *abc* a number of times n, such that $n = 3k+1$ for some $k > 0$.

L_2 : strings over $\{a,b,c,d\}$ that have either *ab* or *acb* as substrings. (Note: a substring of string s is obtained by deleting zero or more symbols from the beginning and end of s; for instance, *banana*, *nan*, and ε are substrings of *banana*.)

L_3 : strings over $\{a,b,c,d\}$ that have *ac* as a substring, but **do not** have *acb* as a substring.

A. Write a regular expression corresponding to L_1. [1 marks]

B. Write a regular expression corresponding to L_2. [1 marks]

C. Draw a (possibly non-deterministic) finite automaton that accepts L_1. [2 marks]

D. Write a regular expression corresponding to L_3. [2 marks]

E. Draw a (possibly non-deterministic) finite automaton that accepts L_3. [2 marks]

Question 2 [10 marks]

Syntactic Analysis

Consider the grammar G:

$$\begin{array}{lcl} S & \rightarrow & \texttt{'('}\ A\ \texttt{'|'}\ B\ \texttt{')'} \\ & | & \varepsilon \\ \\ A & \rightarrow & B\ \texttt{'a'} \\ & | & \varepsilon \\ \\ B & \rightarrow & A\ \texttt{'b'} \\ & | & \varepsilon \end{array}$$

A. Generate a string from the language of G with a length of at least 10 symbols. [2 marks]

B. Draw the derivation tree for your generated string. [2 marks]

C. Write a recursive descent parser for the grammar *G*. [6 marks]

Question 3 [10 marks] Consider the following C functions:

```
#include <stdio.h>

void f(void) {
   int x[1] ;
   *((int*)(x[1])+2) = 10 ;
}

void g(int a) {
   f() ;
   printf("Value of a = %d\n",a);
}

int main() {
   g(2) ;
   return 0 ;
}
```

A. Assuming that the program is compiled on a x86-32 architecture, by the `gcc` compiler, with no optimizations turned on, describe the effect of the assignment in function `f`. (Hint: think of the position of `x` in the activation record). [3 marks]

B. Describe, step by step, the execution of the program. Pay particular attention to the operations that are performed upon calling and returning from functions. What is the output of the program? [4 marks]

C. Comment on the advantages and disadvantages of this style of programming. Why do you think it is not possible to write such programs in other languages? [3 marks]

Question 4 [22 marks]

In this question, we introduce a program translation scheme by example. Then, we derive a ”low level” language based on this transformation, and you will be asked to consider implementing a compiler that translates the toy language of `comp_stmt.pro` into this new low-level language.

Consider the following toy language with two nested loops:

```
sum = 0 ; i = 0 ;
while i < 10 do {
    j = 0 ;
    while j < i do {
        sum = sum + i*j ;
        j = j + 1 ;
    }
    i = i + 1 ;
}
```

This program can be translated into the following equivalent program:

```
pc_exit = 8 ; pc = 0 ;
while pc != pc_exit do {
  switch pc of {
    0 :: { sum = 0 ;  pc = 1 }
    1 :: { i = 0 ; pc = 2 }
    2 :: { if i < 10 then { pc = 3 } else { pc = 8 } }
    3 :: { j = 0 ; pc = 4 }
    4 :: { if j < i then { pc = 5 } else { pc = 7 } }
    5 :: { sum = sum + i*j ; pc = 6 }
    6 :: { j = j + 1 ; pc = 4 }
    7 :: { i = i + 1 ; pc = 2 }
  }
}
```

The translation principle can be applied systematically to any other program. The result will be a program with two assignments, followed by a `while` loop that has a switch statement inside. The variable `pc` mimics the program counter, and in the `switch` statement, the labels play the roles of the program points of the original program.

Now, since some parts of these resulting programs are always the same, we don’t need to keep writing them. Thus, we invent a new *low-level* language that contains only the definition of the final state and the arms of the `switch` statement. We also discard the curly braces and the `pc=` part of the assignment, since they are always the same. For our program, the resulting low-level output is:

```
pc_exit = 8
0 :: sum = 0 ;  1
1 :: i = 0 ; 2
2 :: if i < 10 then 3 else 8
3 :: j = 0 ; 4
4 :: if j < i then 5 else 7
5 :: sum = sum + i*j ; 6
6 :: j = j + 1 ; 4
7 :: i = i + 1 ; 2
```

In essence, the instructions of the new language have the format:

```
   LabelNumber ::  Instruction
```

where `Instruction` may have one of the following two forms:

- `Var = Expr ; NextLabel`
- `if Condition then NextLabel1 else NextLabel2`

The first line declares the exit label. That label must not appear in the low-level program.

Consider now the following variant of our toy language:

- Statements: assignment, `if` (both one-armed and two-armed), `while`, `break` (single level), `goto`.
- All variables are integers; all usual arithmetic operators are available.
- No need for variable declarations.
- No scopes.
- No procedures.

Consider implementing a compiler that translates this toy language into the low-level language defined above. Answer the following questions.

A. Devise a grammar that generates the low-level language described above. [4 marks]

B. Taking as a base the compiler we developed in the file `comp_stmt.pro`, we would like to consider implementing the process of compiling the toy language of that compiler into the new low-level language. Before writing any code (do not worry, you will not have to implement the entire compiler), describe, in English, the translation scheme that you would use. Specifically, describe how you would implement `if` statements, `while` statements, `break` and `goto` statements. Be precise in describing the computed and generated attributes for each node in the AST. [10 marks]

C. Give the expression compiler rule that handles `break` statements. [4 marks]

D. Give the compiler rule that handles assignment. [4 marks]

— E N D   O F   P A P E R —