CS3423: Compilers-II: Lab Exam

Aug17 Semester

September 26th, 2017, Time: 1.5Hrs

Instructions

- You are allowed to access flex/bison manuals ONLY through man pages which are available locally (use **info (man) flex/bison** command).
- Internet is allowed ONLY to submit your work at the end of exam. You should inform TAs while submission. Usage of internet during the course of exam is **strictly** prohibited.
- You are not allowed to carry Mobile phones, Tablets, USB sticks, etc.
- Any methods to access solutions through unfair means would result immediate granting of an FR grade in the entire course (CS3423).
- You can specify any additional assumptions your implementation makes and/or status (if incomplete) in the README file bundled along with the solution.
  - Do note that if you are submitting only a partial solution, your chances of getting partial marks increases if you document your code and give a good README.
- Rename your folder with <roll_No>_EXAM and then compress it.
- You can assume input is error free. In particular no need to do error handling.

Q1) Pretty Printer for Integer Sets and Relations (30 pts)

An **integer set** is a set of integers in n dimensional vector space, which can be used to efficiently represent a loop nest.

Example:

```
for (int i = 1; i < M; i++) {
  for (int j = 1; j < N; j++) {
     // Body of loop
  }
}
```

The loop nest above can be represented as

`D = [M, N] -> {i, j | 0 < i < M, 0 < j < N}`

M, N - The parameters used in the set.
i, j - The variables used in the set.

A **Relation** is a function which maps (integer) elements from one set to another set. A relation can be represented as

`R =[M,N] -> { (i,j) -> (p,q) | 0 < i < M and 0 < j < N and p=j and q=i}`

Your program needs to recognize this representation and pretty print it as shown below.

<u>**Input:**</u> File containing integer sets and relations bounded by affine constraints
<u>**Output:**</u> Pretty printed Integer sets and relations as they appear in input file.

Format for pretty printing
<Type> Set or Relation
Dims: <variable list_lower_case>
Params: <params_list_in_upper_case>
Constraints:
    <Each constraint on new line> of form: …. < ….

Sample input file

```
D= [M] -> {i,j| 0<i<100, 5<j<M}
R= [M,N] -> {i,j -> p,q | 0<i<M and 0<j<5 and 0<p<5 and 0<q<N
and p=i+j+M and q=i-j+N}
```

Expected output file:

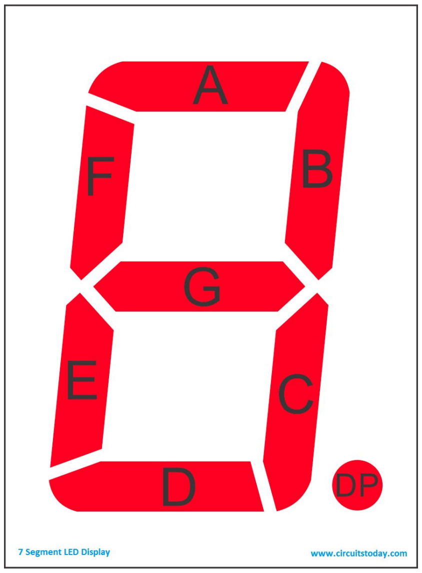
```
Set
Dims: i, j
Params: M
Constraints:
     0<i
     i<100
     5<j
     j<M

Relation
Dims: i, j, p, q
Params: M, N
Constraints:
     0<i
     i<M
     0<j
     j<5
     0<p
     p<5
     0<p
     p<5
     0<q
     q<N
     p = i+j+M
     q = i-j-N
```

Bonus: Handle the sets and relations having >= and <=.

Q2) Seven segment display code generator (20 pts)

A seven segment display consists of 8 LEDs which can be turned on so as to display the desired number. A typical seven segment looks as shown below. Notice the labelling convention for various LEDs.



For displaying 1 we will use following encoding:

```
#    a    b    c    d    e    f    g    dp
     0    1    1    0    0    0    0    0
```

For displaying 1.1 we will use following encoding:

```
#    a    b    c    d    e    f    g    dp
     0    1    1    0    0    0    0    1          →    Seg#1
#    a    b    c    d    e    f    g    dp
     0    1    1    0    0    0    0    0          →    Seg#2
```

This question asks you to write a code generator whose input will be a specification file which describes which LEDs to turn on and output should be C file which we can compile using a sophisticated compiler.

Your program (written in LEX or YACC/BISON or both) should take input file and generate C code. Since it is largely a code-generation problem, we will evaluate your solution based on effective usage of regex or grammar or combination. We will also analyze your code for style, modularity. For sample input we expect following .c file to get generated.

Input file(1)

```
# This specification uses three 7-segment displays connected in
series
# Display value of PI=3.14
INIT
Select: 1
#    a    b    c    d    e    f    g    dp
#Display 3 followed by decimal point
     1    1    1    1    0    0    1    1
Delay:     40
Select: 2
#Display 1
     0    1    1    0    0    0    0    0
Delay: 40
Select: 3
#Display 4
     0    1    1    0    0    1    1    0
Delay: 40
```

Expected Output file:

```
#include<stdio.h>
#include<stdlib.h>
#include<seven_segment.h>
int main()
{
     init();
     while(1)
     {
          select(1);
          write(strtol("11110011"));
          delay(40);
          select(2);
          write(strtol("01100000"));
          delay(40);
          select(3);
          write(strtol("01100110"));
          delay(40);
     }
}
```

Input file(2)

```
# This specification uses one 7-segment display.
# Depending upon input it prints zero or one.
INIT
input: int
input = readInt
Select: 1
if input == 0:
#    a    b    c    d    e    f    g    dp
#Display 0
     1    1    1    1    1    1    0    0
Else:
#Display 1
     0    1    1    0    0    0    0    0
Delay: 40
```

Expected Output file:

```
#include<stdio.h>
#include<stdlib.h>
#include<seven_segment.h>
int main()
{
     init();
     int input;
     while(1)
     {
          input =  readInt();
          select(1);
          if(input==0)
          {
               write(strtol("11111100"));
          else
          {
               write(strtol("01100000"));
          }
          delay(40);
```

```
        }
}
```

You can specify any additional assumptions and/or status(if incomplete) in the README file bundled along with the solution.

ALL THE BEST
IITH-Compilers-Admin team
