CS3423 Compilers-II Aug 2016 Lab Exam Handout

November 27, 2016, Time: 3Hrs

Instructions:

- You are allowed to access flex/bison manuals ONLY through man pages which are available locally.
- Internet is allowed ONLY to download this handout at the start of exam and to submit your work at the end of exam. 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.
- Rename your folder with *<roll_No>_EXAM* and then compress it.

---

1 PolyCodegen: Simplified Polyhedral Code Generation (40 points)

Polyhedral compilation consists of applying automatic loop transformations onto a set of nested loops. Typically it involves extracting the model from an input program, performing transformations over it, and then generating the code from the transformed model.

For example consider following loop nest:

Polyhedral representation of the *domain* of the above loop nest is shown below:

```
// 3 dimensional loop nest
for(i=0; i<M;i++)
  for(j=0; j<i; j+=2)
    for(k=0; k<=N; k+=3)
```

Listing 1: Sample loop nest

```
{i,j,k | 0 <= i < M:1; 0 <= j < i:2 ; 0 <= k <= N:3}
```

This problem is about generation of a C program from model specification provided through an input file. The input file would be provided in the following format: (Like C++ syntax, comments have '//' as prefix.)

```
//Program Parameters
Let N := 1000
Let M := 100000
//Statements
Let S1(i,j,k) := (A(i,j,k) = 10)
Let S2(i,j) := (B(i,j) = 1)
// Polyhedral representation
S1(i,j,k)-> {i,j,k | 0 <= i < M: 1; 0 <= j < i: 2 ; 0 <= k <= N: 3}
S2(i,j)--> { i, j| 0 <= i< 10: 2; -10 <= j <= i: 10}
```

Listing 2: PolyCodegen: Input format

The expected output for above input is shown below:

```
#include <stdio.h>
#define N 1000
#define M 100000
#define S1(i,j,k) (A(i,j,k) = 10)
#define S2(i,j) (B(i,j) = 1)

int main(){
  for(i=0; i<M; i++)
    for(j=0; j<i; j+=2)
      for(k=0; k<=N; k+=3)
        S1(i,j,k);

  for(i=0; i<10;i+=2)
    for(j=-10; j<=i; j+=10)
      S2(i,j);
}
```

Listing 3: PolyCodegen: Expected output

Some hints:

- Using of attribute grammar is recommended.
- You may use three attributes per loop dimension corresponding to:
  - Lower bound
  - Upper bound
  - Loop increment
- You may assume that the ordering of constraints (like 0<=i<M) is same in which you need to generate loop nest.
- You may assume that statement macros consist of array initializations only. This will simplify your attribute grammar.

2 HTMLtagGen: Automatic generation of closing HTML tags (30 points)

In this question, you have to write a parser for a (restricted) HTML script, and use the same to automatically generate the closing tags. To simplify your design, following are the restrictions on input:

The input file consists of only the following tags: <HTML>, <HEAD>, <TITLE>, <BODY>, <P>, <U>, <I>, <B>, <DIV>. These tags can be in uppercase, lowercase or mixture of both. The HTML script contains text as content. This rules out the possibility of images, hyperlinks etc.

Input document does not contain closing tags at all. Your program has to generate them at the right location. There is no need to perform any error-handling. Your program should exit gracefully without any segmentation fault etc.

Just to remind you: HTML document starts with <HTML> tag. The <HEAD> tag specifies the header contents and this included the title of the page through <TITLE> tag. The <BODY> tag specifies the actual web-page contents. Typically <BODY> comprises of several <DIV> tags. A paragraph is defined by using <P> tag. The formatting tags namely <U>, <I>, <B> format the next word immediately following them.

Some hints:

- The input script consists of a single <Body> and <title> in the entire document.
- A paragraph ends before the beginning of the next paragraph.
- Indentation of the output will be bonus.

Sample Input:

```
<HTML>
  <HEAD>
    <TITLE>
       Heterogeneous parallel architectures
  <BODY>
    <P> GPUs are motivated from hardware arrays of processors.
    <P> FPGA programming is <B>difficult.
    <P> A Polyhedron must be bounded by <U>affine inequalities.
    <P> Neural networks can be trained efficiently on <I>FPGAs.
```

Listing 4: HTMLtaggen: Sample Input HTML script

Expected Output:

```
<HTML>
  <HEAD>
    <TITLE>
      Heterogeneous parallel architecture
    </TITLE>
  </HEAD>
  <BODY>
    <P>GPUs are motivated from hardware arrays of processors. </P>
    <P>FPGA programming is <B>difficult</B></P>
    <P>A Polyhedron must be bounded by <U>affine</U> inequalities</P>
    <P>Neural networks can be trained efficiently on <I>FPGAs.</I> </P>
  </BODY>
</HTML>
```

Listing 5: HTMLtagGen: Expected output

3 PyDSPrettyPrint: Python array/tuple/dictionary pretty printer (30 points)

In this problem, you have to write a pretty printer for Python format data.
Here are the rules:

- One indentation level is 4 spaces: use of tabs is not allowed.
- Strings consist of alphanumeric characters ONLY, and are bound by single or double quotes. No other characters will be in the strings, and multi-line strings need-not/should-not be considered.
- Numbers may be integers or floating point, and may have exponents.

- While printing strings in the output, they should be delimited only by single quotes.
- Lists, tuples, and dictionaries will be strictly non-empty.

Here is the grammar in EBNF style:

⟨*input*⟩ ::= ⟨*value*⟩

⟨*Value*⟩ ::= ⟨*String*⟩
| ⟨*Number*⟩
| ⟨*List*⟩
| ⟨*Tuple*⟩
| ⟨*Dict*⟩

⟨*List*⟩ : '`[`' ⟨*ValueList*⟩ '`]`'

⟨*Tuple*⟩ : '`(`' ⟨*ValueList*⟩ '`)`'

⟨*Dict*⟩ : '`{`' ⟨*PairList*⟩ '`}`'

⟨*ValueList*⟩ : ⟨*Value*⟩ [['`,`' ⟨*Value*⟩]]*

⟨*PairList*⟩ : ⟨*Pair*⟩ [['`,`' ⟨*Pair*⟩]]*

⟨*Pair*⟩ : ⟨*Value*⟩ '`:`' ⟨*Value*⟩

Sample Input:

```
{'abc':[1,-2e-5,3], ("123",7):[4,0,.6]}
```

Listing 6: PyDSPrettyPrint: Input

Expected Output:

```
{
    'abc': [
            1,
            -2e-5,
            3
    ],
    (
            '123', <Notice single quotes used here
            7
    ): [
            4,
            0,
            .6
    ]
}
```

Listing 7: PyDSPrettyPrint: Expected Output

—— ALL THE BEST ——

IITH-Compilers team
