Midterm Exam
CS 314, Fall '17
October 27
SAMPLE SOLUTION

DO NOT OPEN THE EXAM UNTIL YOU ARE TOLD TO DO SO

Name: ______________________________

Rutgers ID number: ______________________________

Section: _____

WRITE YOUR NAME ON EACH PAGE IN THE UPPER RIGHT CORNER

Instructions

We have tried to provide enough information to allow you to answer each of the questions. If you need additional information, make a *reasonable* assumption, write down the assumption with your answer, and answer the question. There are **5** problems, and the exam has **8** pages. Make sure that you have all pages. The exam is worth **250** points. You have **80 minutes** to answer the questions. Good luck!

This table is for grading purposes only

1	/ 80
2	/ 60
3	/ 30
4	/ 20
5	/ 60
total	/ 250

NAME: ______________

Problem 1 – Regular Expressions, DFA and Context Free Grammars (80 pts)

The context-free grammar `G` is specified in Backus-Naur-Form as follows, with `A` as the start symbol:

```
1: A ::= a A |
2:       b B
3: B ::= b B |
4:       C
5: C ::= c
```

1. Give a rightmost derivation ($\Rightarrow_R$) for the string `a b c` given the grammar above. (15 pts)

   A $\Rightarrow_R$ aA $\Rightarrow_R$ abB $\Rightarrow_R$ abC $\Rightarrow_R$ abc

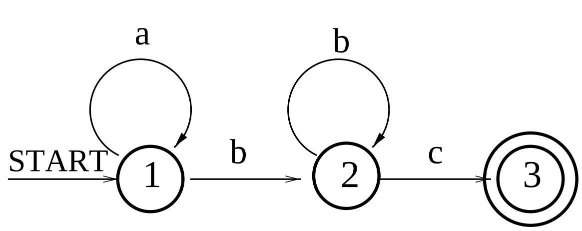
2. Give the LL(1) parse table for the grammar `G`. **Insert the rule number or leave an entry empty**.(35 pts)

	a	b	c	eof
A	1	2		
B		3	4	
C			5	

3. Give a regular expression for the language generated by the grammar `G`. (15 pts)

   a^*b^+c

4. Specify a DFA by extending the state transition diagram below. The start state is **state 1**, and the final (accepting) state is **state 3**. You are only allowed to add edges with their appropriate labels, i.e., valid labels are a, b, and c. Note that an edge may have more than one label. You **must not** add any states. (15 pts)



NAME: ________________

Problem 2 – Context Free Grammars (60 pts)

A *context-free language* is a language that can be specified using a context-free grammar. A *regular* language is a language that can be specified using a regular expression.

For the three languages given below, if the language is context-free, give a compact context-free grammar in Backus-Naur-Form (BNF). If the language is regular, give a compact regular expression using the regular expression syntax introduced in class. If a language is context-free and regular, give both specifications, a BNF and a regular expression. You do not have to justify why you believe a language is not context-free or not regular.

1. { $a^{3n}b^n$ | n $\geq$ 0 }, with alphabet Σ = {a, b}

   S ::= aaaSb | ϵ

2. { $a^n b^{2m} c^n$ | n $\geq$ 0, m >0 }, with alphabet Σ = {a, b, c}

   S ::= aSc | B

   B::= bbB | bb

3. { w | w has at least 2 symbols, but no more than 4}, with alphabet Σ={a, b}

   S ::= A A B B

   A ::= a | b

   B ::= a | b | ϵ

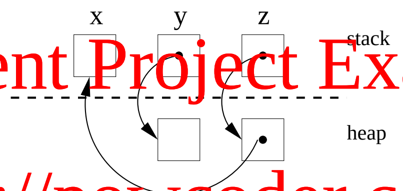
   Regular expression: (a | b) (a | b) (a | b | ϵ) (a | b | ϵ)

NAME: ________________

Problem 3 – Pointers and Memory Allocation in C (30 pts)

```
int main() {
  int   x;
  int  *y;
  int **z;

  z = (int **) malloc (sizeof(int *));
  y = (int *) malloc (sizeof(int));
  x = 1;
  *z = &x;
  *y = x;
  x = x + 1;
  **z = *y + 3;
  printf("x=%d, *y=%d, **z=%d\n", x, *y, **z);
  return 0;
}
```



1. What is the output of the above program? (3 pts each)

   x= <u>4</u>, *y= <u>1</u>, **z= <u>4</u>

2. Specify, whether the following program objects are allocated on the **stack** (includes global variables), on the **heap**, or **not defined** (2 pts each).

   x is allocated on the <u>stack</u>   y is allocated on the <u>stack</u>

   z is allocated on the <u>stack</u>   *x is allocated on the <u>not defined</u>

   *y is allocated on the <u>heap</u>   *z is allocated on the <u>heap</u>

   **y is allocated on the <u>not defined</u>   **z is allocated on the <u>stack</u>

3. Assume the following code segment:

```
int *x;
*x = 5;
printf("%d\n", *x);
```

   Is there a problem with this code? Assume that when you ran the code a couple of times, it printed “5”. If you believe there is a problem, give a possible “fix” for the problem? (5 pts)

The content of variable `x` is not initialized. However, its content is used as an address of a memory location, and that memory location is assigned the value `5`.

To fix the problem, the pointer `x` should be initialized to `NULL` in its declaration, and `x` must point to an object on the heap before it is dereferenced. This object is allocated as follows:

```
x = (int *) malloc(sizeof(int *))
```

This statement should be placed before statement `*x = 5`, i.e., before the dereference operation on `x`.

NAME: ________________

Problem 4 – Compilers vs. Interpreters (20 pts)

To answer this question, please use the following definitions.

Definition A **compiler** maps an input program to a semantically equivalent output program. Note that the input and output language may be the same. □

Definition An **interpreter** maps an input program to the answers it computes; In other words, it executes the program. □

As part of the C project, we used and/or wrote the following programs/commands:

- gcc – *usage*: gcc <program>
- compile – *usage*: compile <program>
- constfolding – *usage*: constfolding < <program>
- sim – *usage*: sim <program>

Under Unix/Linux, commands can be entered on a single command line if they are separated by a semicolon. For instance, saying cd foo; ls will change the current directory to subdirectory foo, and will list its files.

In the project, we used several languages, namely C, tinyL, RISC machine code and ilab machines object code (executables). **Classify the following commands or the entire sequence of commands** as either compiler or interpreter. Note that you should treat a sequence of commands as a single unit, i.e., as a single composed command. If the single or composed command is a compiler, give its input and output language (e.g.: input language: C, output language: tinyL). For an interpreter, just give its input language.

1. compile test1

   Answer (mark one): compiler: [X] or interpreter [ ]

   input language: tinyL, output language: ILOC RISC machine code

2. constfolding < tinyL.out

   Answer (mark one): compiler: [X] or interpreter [ ]

   input language: ILOC RISC machine code, output language: ILOC RISC machine code

3. compile test1; sim tinyL.out

   Answer (mark one): compiler: [ ] or interpreter [X]

   input language: tinyL, output language: ____________________

4. gcc Compiler.c

   Answer (mark one): compiler: [X] or interpreter [ ]

   input language: C, output language: ilab machine code (executable)

NAME: _______________

Problem 5 – Syntax-Directed Translation (60 pts)

Assume the following partial expression grammar:

<expr> ::= + <expr> <expr> |
　　　　　$*$ <expr> <expr> |
　　　　　<const>
<const> ::= 1 | 2

instr. format	description	semantics
memory instructions		
`loadI <const>` $\Rightarrow_R$ r_x	load constant value <const> into register r_x	$r_x \leftarrow$ <const>
arithmetic instructions		
`add` r_x , r_y $\Rightarrow_R$ r_z	add contents of registers r_x and r_y , and store result into register r_z	$r_z \leftarrow r_x + r_y$
`mult` r_x , r_y $\Rightarrow_R$ r_z	multiply contents of registers r_x and r_y , and store result into register r_z	$r_z \leftarrow r_x * r_y$

Here is a recursive descent parser that implements a compiler for the above grammar. Here is the important part of the code:


```
int expr() {
        int reg, left_reg, right_reg;
        switch (token) {
        case '+': next_token();
                  left_reg = expr(); right_reg = expr(); reg = next_register();
                  CodeGen(ADD, left_reg, right_reg, reg);
                  return reg;
        case '*': next_token();
                  left_reg = expr(); right_reg = expr(); reg = next_register();
                  CodeGen(MULT, left_reg, right_reg, reg);
                  return reg;
        case '1':
        case '2': return const();
        }
}

int const() {
        int reg;
        switch (token) {
        case '1':  next_token(); reg = next_register();
                   CodeGen(LOADI, 1, reg);
                   return reg;
        case '2': next_token(); reg = next_register();
                   CodeGen(LOADI, 2, reg);
                   return reg;
        }
}
```

NAME: ______________

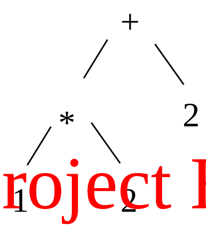
Make the following assumptions:

- The value of variable "token" has been initialized correctly.
- The function `CodeGen` is the one from our first project.
- **The first call to function `next_register()` the shown parser returns integer value "1".** In other words, the first register that the generated code will be using is register r_1.
- Your parser "starts" by calling function `expr()` on the entire input.

1. Show the code that the recursive descent parser generates for input

`+ * 1 2 2`

will produce:



loadI 1 $\Rightarrow$ r1
loadI 2 $\Rightarrow$ r2
mult r1, r2 $\Rightarrow$ r3
loadI 2 $\Rightarrow$ r4
add r3, r4 $\Rightarrow$ r5

NAME: ______________

2. Change the basic recursive-descent parser to implement an interpreter for our example language. You may insert pseudo code in the spaces marked by ________. No error handling is necessary.

```
  int  expr() {

        int a, b;
        switch (token) {
        case '+': next_token();
                       a = expr(); b = expr();
                  return (a + b);

        case '*': next_token();
                       a = expr(); b = expr();
                  return (a * b);

        case '1':
        case '2': return const();
        }
}

  int  const() {

        switch (token) {
        case '1':  next_token();
                   return 1 ;

        case '2':  next_token();
                   return 2 ;
        }
}
```