CS 571
Homework 1

Due: Sep 28 **100 points**

Important Reminder: As per the course Academic Honesty Statement, cheating of any kind will minimally result in receiving an F letter grade for the entire course.

To be submitted on paper in class.

1. The Ruby Programming Language book describes a ruby integer literal as follows:

   > You write integers using an optional leading sign, an optional base indicator (0 for octal, 0x for hex, or 0b for binary), followed by a string of digits in the appropriate base. Underscore characters are ignored in the digit string.

   Note that the above description is somewhat informal; the letters mentioned in the above description may be either upper or lower case and the underscore characters can only occur **strictly within** the digit string.

   Provide a regular expression which describes ruby integers. *5-points*

2. The Java Language Specification describes floating point numbers as follows:

   > A floating-point literal has the following parts: a whole-number part, a decimal or hexadecimal point (represented

> by an ASCII period character), a fractional part, an exponent, and a type suffix. A floating point number may be written either as a decimal value or as a hexadecimal value. For decimal literals, the exponent, if present, is indicated by the ASCII letter e or E followed by an optionally signed integer. For hexadecimal literals, the exponent is always required and is indicated by the ASCII letter p or P followed by an optionally signed integer.
>
> For decimal floating-point literals, at least one digit, in either the whole number or the fraction part, and either a decimal point, an exponent, or a float type suffix are required. All other parts are optional. For hexadecimal floating-point literals, at least one digit is required in either the whole number or fraction part, the exponent is mandatory, and the float type suffix is optional.
>
> A floating-point literal is of type float if it is suffixed with an ASCII letter F or f; otherwise its type is double and it can optionally be suffixed with an ASCII letter D or d.

Note also that a hexadecimal floating point number must start with either `0x` or `0X` and its significand (i.e. the whole number and fraction parts) may contain hexadecimal digits while its exponent is restricted to contain only decimal digits.

Using standard regex syntax, give a regex for Java floating-point literals. Instead of writing a single regex, you may use a *regex definition* using a CFG-style syntax with a sequence of named regex definitions with later regex definitions referring to earlier regex definitions by name (enclosed within angle-brackets). So notation like:

```
<digit>
 : [0-9]
 ;
<integer>
 : <digit>+
 ;
```

is acceptable.

Your regex need not make any attempt to restrict the range of the represented numbers.

15-points

3. As mentioned in the **Rationale for the Requirements** for Project 1, the requirements do not allow any kind of whitespace characters within constructed regex's.

   (a) How would you modify the requirements to permit whitespace to be specified?

   (b) How would you change the implementation to permit whitespace within the constructed regex's. *10-points*

4. As mentioned in the **Rationale for the Requirements** for Project 1, the translated regex's may have redundant parentheses. How would you modify your parser to ensure that the translation contains only non-redundant parentheses.

   For example, the project requires the input `((chars(a)) . chars(b))+chars(c)` be translated to `((((a])[b]))|[c])` where all the parentheses in the output are redundant; it could simply be translated to `[a][b]|[c]`. OTOH, `chars(a) . chars(b) + chars(c)` translates to `([a]([b]|[c]))`, where not all the parentheses are redundant; without redundant parentheses, it would be `[a]([b]|[c])`.

   Hint: consider providing an additional parameter to each parsing function which gives it some idea of the context within which it is being called. *20-points*

5. A language $\mathcal{L}$ consists of strings containing n `a`'s followed by $n + 4$ `b`'s for $n >= 0$.

   (a) Give a CFG for the language $\mathcal{L}$.

   (b) Give an inductive proof that your CFG describes precisely the language $\mathcal{L}$. *15-points*

6. Why is a recursive-descent parser for a grammar amenable to recursive-descent parsing guaranteed to terminate? *5-points*

7. The following C function in program t.c:

```
static void
f(int a, double b, int c)
{
  int t1 = a + b;
  int t3 = t1 + c;
  double t2 = b + c;
  int *p1 = &a;
  printf("address of a is %p\n", &a);
  printf("address of b is %p\n", &b);
  printf("address of c is %p\n", &c);
  printf("address of p1 is %p\n", &p1);
  printf("address of t1 is %p\n", &t1);
  printf("address of t2 is %p\n", &t2);
  printf("address of t3 is %p\n", &t3);
}
```

prints out the addresses of its arguments and local variables. On one run, the output was:

```
address of a is 0x7ffcd3fe829c
address of b is 0x7ffcd3fe8290
address of c is 0x7ffcd3fe8298
address of p1 is 0x7ffcd3fe82b0
address of t1 is 0x7ffcd3fe82a0
address of t2 is 0x7ffcd3fe82a8
address of t3 is 0x7ffcd3fe82a4
```

(a) Show the layout for the stack frame for the above function invocation. Specifically, show the stack frame with the specific addresses occupied by the different variables.

(b) Explain the above layout in terms of what has been mentioned in class about stack-frame layout.

(c) Guess at possible addresses where the return address for the above function invocation may be stored. *15-points*

8. Discuss the validity of the following statements:

(a) Garbage collection was a new technique first used by the Java programming language.

(b) In a typical programming language, if the name referring to an entity goes "out of scope" and is no longer accessible, then the entity becomes inaccessible and can be destroyed.

(c) Global variables cannot be used in multi-threaded programs.

(d) A scanner must always discard whitespace and comments.

(e) Stack allocation can be used for entities having arbitrary lifetimes.
15-points