CMPT 383
Midterm
October 25, 2005

1. (10%) Indicate whether the following statements are **True** or **False**
   a) The six attributes of a variable are name, alias, type, lifetime, scope, and address. **False**
   b) In aliasing, one name at a program point refers to two entities and only the context of its instantiation disambiguates it. **False**
   c) A pure interpreter produces a faster program execution than a compiler. **False**
   d) Top-down parsers are LL parsers where 1st L stands for left-to-right scan and 2nd L stands for a leftmost derivation. **True**
   e) The Shift process of a bottom-up parser is the most complicated process. **False**
   f) Bottom-up parsers are in the LR family, where L stands for left-to-right scan and R stands for rightmost derivation. **True**
   g) An enumeration type is a structured data type. **False**
   h) The size of a union data type is equal to the sum of the size of each variant. **False**
   i) The input of a lexical analyzer is the set of tokens and its output is a parse tree. **False**
   j) The variable `strBaby_boy` uses the Hungarian camel notation. **False**

2. (5%) Program# 1 is a legal Pascal program:
   a) What values do `true` and `false` have this program?
   Both (*true* and *false*) have the value false.

   b) What principles does this violate?
   Readability → syntax considerations → special words
   ***True* and *false* do no reflect their expected meaning (values).**

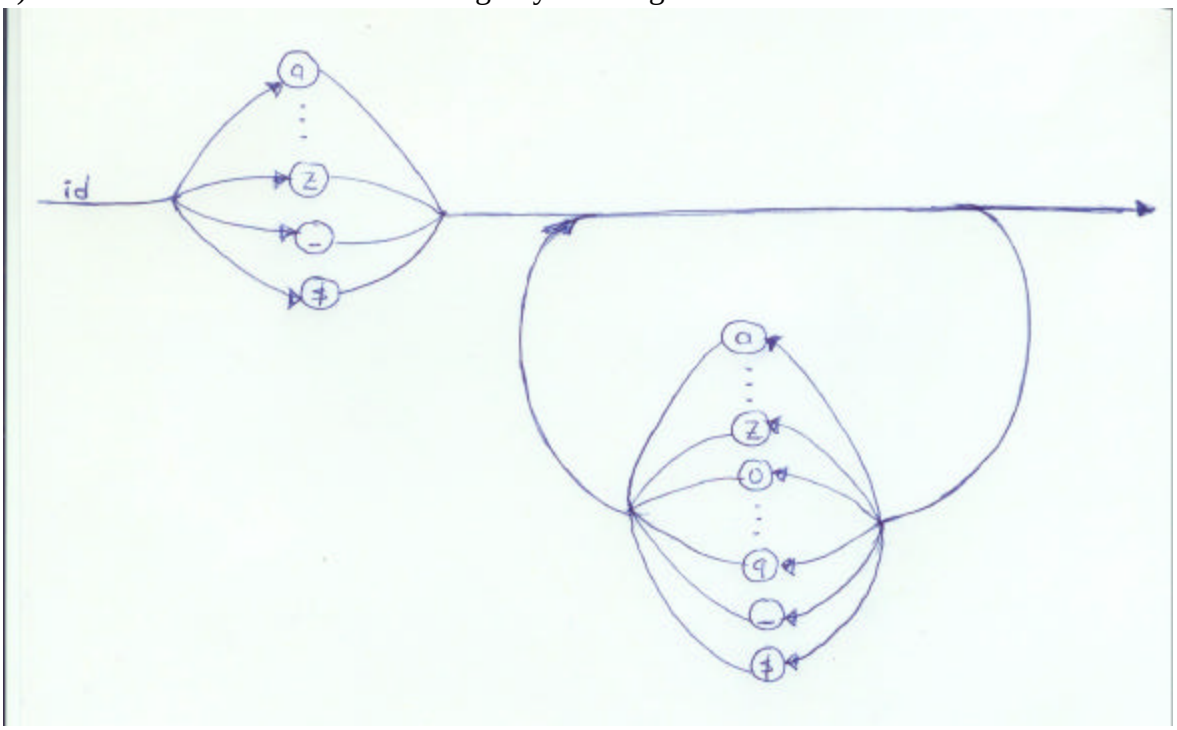
3. (20%) The following is the syntax definition for identifiers in Java:
   An identifier is a sequence of one or more characters. The first character must be a letter, underscore, or dollar sign. The other characters must be letters, numbers, underscores, or dollar signs
   a) Define Java identifier `<id>` in BNF.

```
<id> ::= <char> <char_numbers>
<char> ::=  a | b | … | z | A | B | … | Z | _ | $
<number> ::= 0 | 1 | … | 9
<char_number> ::= <char> | <number>
<char_numbers> ::= <char_number> | <char_number> <char_numbers>
```

   b) Define Java identifier `<id>` in EBNF with only one production rule.

<id> -> (a | b | … | z | A | B | … | Z | _ | $) {(a | b | … | z | A | B | … | Z | _ | $ | 0 | 1 | … | 9)}

c) Define Java identifier <id> using a syntax diagram.



d) Based on the BNF grammar, write the leftmost derivation for the following identifier: _state_01$a

<id> => <char> <char_numbers> => _ <char_numbers> => _ <char_number> <char_numbers> => _ <char> <char_numbers> => _ s <char_numbers> => _ s <char_number> <char_numbers> => _ s <char> <char_numbers> => _ s t <char_numbers> => _ s t <char_number> <char_numbers> => _ s t <char> <char_numbers> => _ s t a <char_numbers> => _ s t a <char_number> <char_numbers> => _ s t a<char> <char_numbers> => _ s t a t <char_numbers> => _ s t a t <char_number> <char_numbers> => _ s t a t <char> <char_numbers> => _ s t a t e <char_numbers> => _ s t a t e <char_number> <char_numbers> => _ s t a t e <char> <char_numbers> => _ s t a t e _<char_numbers> => _ s t a t e _ <char_number> <char_numbers> => _ s t a t e _ <number > <char_numbers> => _ s t a t e _ 0 <char_numbers> => _ s t a t e _ 0 <char_number> <char_numbers> => _ s t a t e 0 <number > <char_numbers> => _ s t a t e _ 0 1 <char_numbers> => _ s t a t e _ 0 1 <char_number> <char_numbers> => _ s t a t e _ 0 1 <char > <char_numbers> => _ s t a t e _ 0 1 $ <char_numbers> => _ s t a t e _ 0 1 $ <char_number> => _ s t a t e _ 0 1 $ <char > => _ s t a t e _ 0 1 $ a

4. (6%) Given Grammar#1. Show all pairwise disjoint tests for <A>

FIRST(<A_1>) = FIRST(<B>) = {f,g,c,d,b, Î }
FIRST(<A_2>) = FIRST(<C>) = {f,g,c, Î }
FIRST(<A_3>) = FIRST(a) = {a}

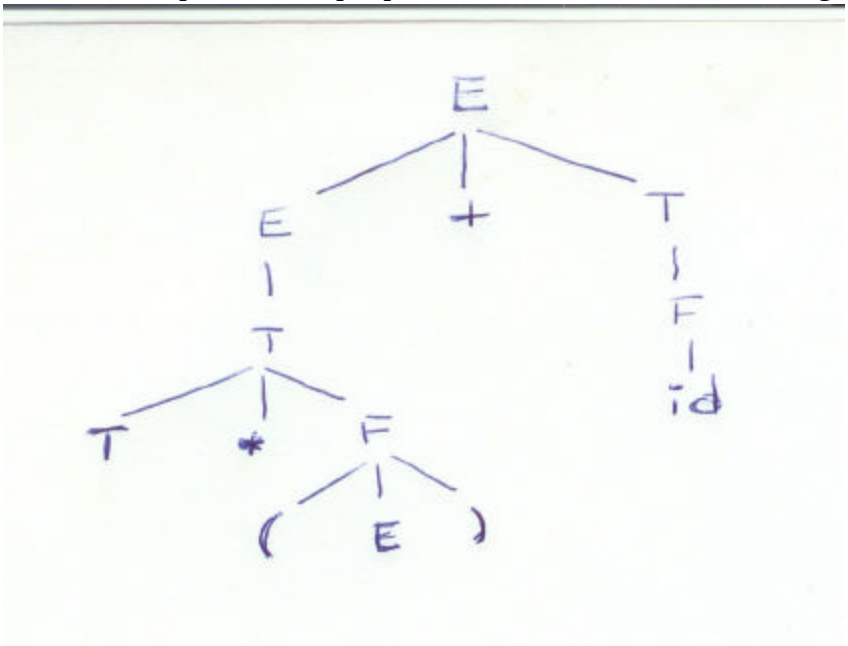
FIRST(<B>) = FIRST(<D>)ÈFIRST(b) = {f,g,c,d,b,Î}
FIRST(<D>) = FIRST(<C>)ÈFIRST(d)È{Î} = {f,g,c,d,Î}
FIRST(<C>) = {f}È{g}È{c}È{Î} = {f,g,c,Î}

FIRST(<A_1>) Ç FIRST(<A_3>) = Æ (pass)
FIRST(<A_2>) Ç FIRST(<A_3>) = Æ (pass)
FIRST(<A_1>) Ç FIRST(<A_2>) = {f,g,c,Î} (fail)

5. (10%) Given Grammar#2 and its corresponding LR parsing table
   a) Show `T*(E)+id` is a right sentential form of the grammar.

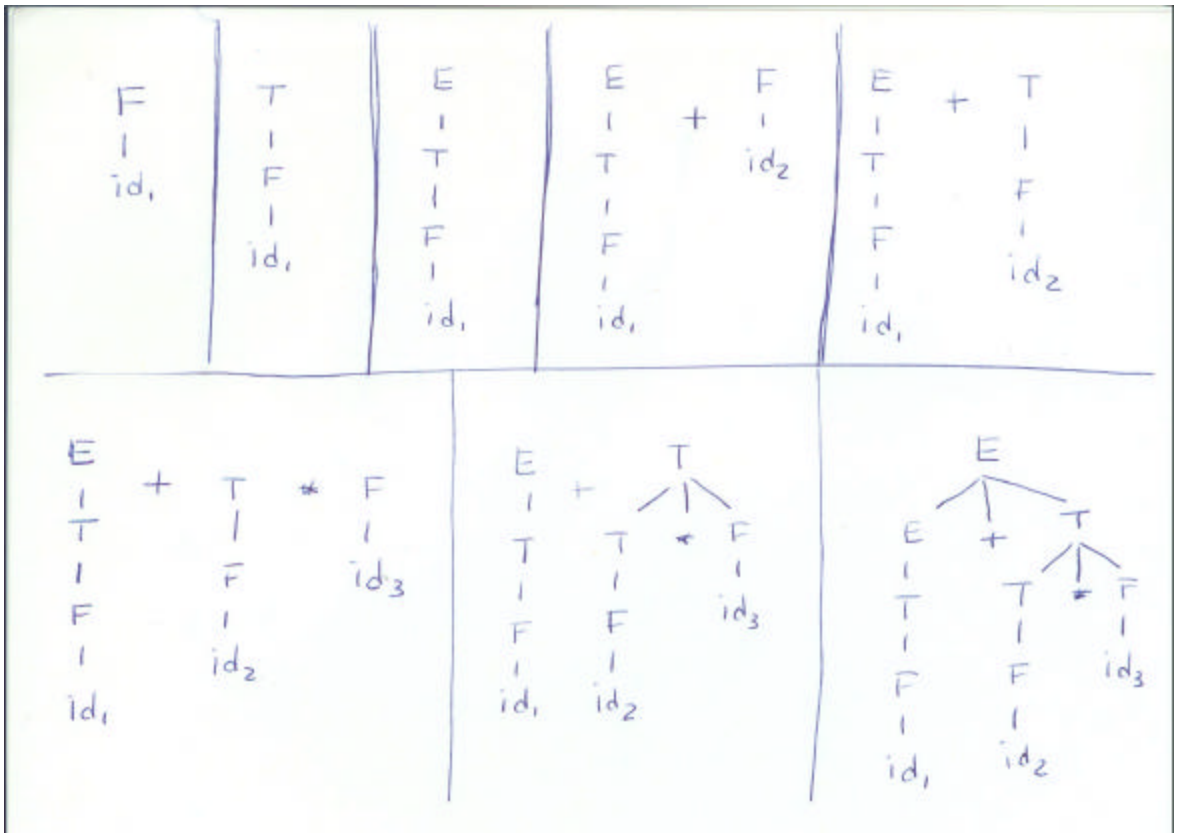
E => E + T => E + F => E + id => T + id => T * F + id
=> T * (E) + id

b) Show the phrases, simple phases, and handle of the above right sentential form.

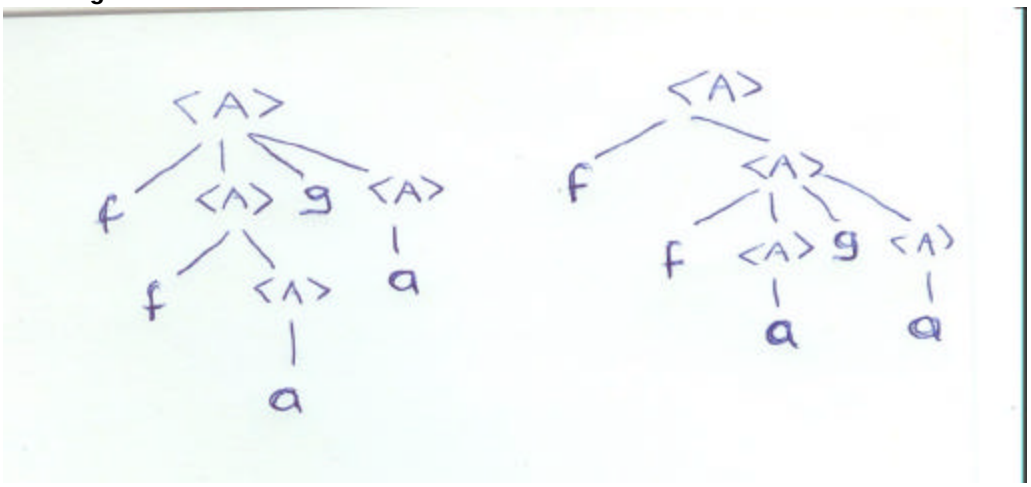


Phrases: T*(E)+id, T*(E), (E), id
Simple phrases: (E), id
Handle: (E)

c) For sentence `id+id*id`, for each **reduce** action in the LR parsing, show the partial parse tree built by this reduce.



6. (10%) Given Grammar#3. Is the grammar ambiguous? Is the grammar left-recursive?
   The grammar is ambiguous because there is more than one parse tree for some strings, such as `ffaga`



   The grammar is left-recursive because there is the following chain:
   `<A> => <B> w => <C> w <A> w => <A> g w <A> w`

7. (10%) Given Grammar#4. Show that the two conditions for predictive parsing are satisfied.
   First condition:
   `FIRST(<exp>₁) Ç FIRST((<exp>₂) = Æ`
   `FIRST(<exp>₁) = {(}`

FIRST(<exp>$_2$) = {a}

Second condition:
FIRST(<list>) Ç FOLLOW(<list>) = Æ
FIRST(<list>) = FIRST(<exp>) = { (, a }
FOLLOW(<list>) = FIRST()) = {) }

8. (12%) Consider the attribute grammar with nonterminals `A, B, C`, and terminals `x` and `y`. The start symbol is `A`. The attributes are assigned to these grammar symbols as indicated by parse tree #1. The grammar (Grammar #5) has 5 productions labeled *p, q, r, s,* and *t*. List the defined and used attribute occurrences.

	Defined	Used
p	A.attA, C.att2, C.att1	B.attB, C.att3
q	A[1].attA, C.att2, C.att1	B.attB, C.att3, A[2].attA
r	B.attB, x.attx	
s	C.att3, x.attx	C.att2, C.att1
t	C[1].att3, x.attx, C[2].att2, c[2].att1	C[1].att2, C[1].att1, C[2].att3

9. (5%) Given the partial program #2 (Explain your answer)
For variables `a, b, c, d`, which one is type compatible with which one, in terms of
   a) Name type compatibility?
   None of a, b, c, d are compatible in terms of name type compatibility.

   b) Structure type compatibility?
   a is compatible with c
   b is compatible with d

10. (12%) Use the partial program #3 (record definition) to answer the following questions. Assume that `char` and `bool` variables take 1 byte, `int` and `pointer` variables take 4 bytes and `double` variables take 8 bytes.
   a) How many bytes are needed to store a variable of the student type?
   20 + 4 + 4 + 8 = 36

   b) What are the offsets (starting position relative to the base address of the data objects in bytes) for each field in the record?

```
name        0
id          20
credits     24
gpa         28
```

   c) How many bytes would be needed to store an array of students created with the following declaration? `student class[20];`
   20 * 36 = 720

   d) What is the address of `student[10].credits` relative to the start of the array?
   10 * 36 + 24 = 360 + 24 = 384

Program #1 **(Question 2)**

```
program Homer;
var true, false : boolean;
begin
      (* := is assignment *)
      (* = is test for equality *)
      true := 1 = 0;
      false := true;
      (* here *)
end.
```

Program #2 **(Question 9)**

```
struct A {int x; float y;};
struct B {float a; int b;};
typedef A C;
typedef B D;
A a; B b; C c; D d;
```

Program #3

```
typedef struct student {
      char name[20];
      int id, credits;
      double gpa;
} student;
```

Grammar #1 **(Question 4)**

```
<A> ::= <B> | <C> | a
<B> ::= <D> | b <B>
<C> ::= f | g | c <C> | ∈
<D> ::= <C> d | ∈
```

Grammar #3 **(Question 6)**

```
<A> ::= f <A> g <A>| f <A>| <B> w | a
<B> ::= f <A> w | <C> w <A>
<C> ::= g <B> | <A> g | g <A> g
```

Grammar #2 **(Question 5)**

```
<E> ::= <E> + <T> | <T>
<T> ::= <T> * <F> | <F>
<F> ::= ( <E> ) | id
```

Grammar #4 **(Question 7)**

```
<exp> ::= ( <list> ) | a
<list> ::= <exp> [<list>]
```

LR Parsing Table for Grammar #2 **(Question 5)**

STACK	INPUT	ACTION
$	$id_1+id_2*id_3$	Shift
$id_1	$+id_2*id_3$	Reduce by F→id
$F	$+id_2*id_3$	Reduce by T→F
$T	$+id_2*id_3$	Reduce by E→T
$E	$+id_2*id_3$	Shift
$E+	id_2*id_3	Shift
$E+id_2	$*id_3$	Reduce by F→id
$E+F	$*id_3$	Reduce by T→F
$E+T	$*id_3$	Shift
$E+T*	id_3	Shift
$E+T* id_3	$	Reduce by F→id
$E+T*F	$	Reduce by T→T*F
$E+T	$	Reduce by E→E+T
$E	$	Accept

Grammar #5 **(Question 8)**

```
p: <A> ::= <B> <C>
q: <A> ::= <C><B><A>
r: <B> ::= y x
s: <C> ::= x
t: <C> ::= x <C>
```

Parse Tree #1 **(Question 8)**