# 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.

   b) In aliasing, one name at a program point refers to two entities and only the context of its instantiation disambiguates it.

   c) A pure interpreter produces a faster program execution than a compiler.

   d) Top-down parsers are LL parsers where 1st L stands for left-to-right scan and 2nd L stands for a leftmost derivation.

   e) The Shift process of a bottom-up parser is the most complicated process.

   f) Bottom-up parsers are in the LR family, where L stands for left-to-right scan and R stands for rightmost derivation.

   g) An enumeration type is a structured data type.

   h) The size of a union data type is equal to the sum of the size of each variant.

   i) The input of a lexical analyzer is the set of tokens and its output is a parse tree.

   j) The variable `strBaby_boy` uses the Hungarian camel notation.

2. (5%) Program# 1 is a legal Pascal program:
    a) What values do `true` and `false` have this program?
    b) What principles does this violate?

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

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

5. (10%) Given Grammar#2 and its corresponding LR parsing table
    a) Show `T*(E)+id` is a right sentential form of the grammar.
    b) Show the phrases, simple phases, and handle of the above right sentential form.
    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?

7. (10%) Given Grammar#4. Show that the two conditions for predictive parsing are satisfied.

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.

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?
    b) Structure type compatibility?

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?
    b) What are the offsets (starting position relative to the base address of the data objects in bytes) for each field in the record?
    c) How many bytes would be needed to store an array of students created with the following declaration? `student class[20];`
    d) What is the address of `student[10].credits` relative to the start of the array?

**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)**