CORE EXAMINATION
Department of Computer Science
New York University
January 24, 2003

This is the common examination for the M.S. program in CS. It covers core computer science topics: Languages and Compilers, Operating Systems, and Algorithms. The exam has two parts. The first part lasts three hours and covers the first two topics. The second part, given this afternoon, lasts one and one-half hours, and covers algorithms.

Attempt all of the questions. Use the proper booklet for each question. Each booklet is marked with the Area and Question number, in the form PLC1, PLC2, PLC3, OS1, OS2, ALGS1, ALGS2, ALGS3. Use the appropriate booklet for each question. DO NOT put your name on the exam booklet. Instead, your exam number must be on every booklet.

You will be graded according to your exam number, shown on the envelope containing the booklets. Remember your exam number: when grades are given out, they will be published according to this number, not by name.

Make sure your name and signature are on the envelope. This is the only place where your name appears. Please include all the booklets inside the envelope. You can keep the exam. Good luck!

Programming Languages and Compilers

Question 1

Consider the ML function:

```
fun repeat f n x =
  if n > 0 then repeat f (n - 1) (f x)
  else x
```

Informally, **repeat** applies f n times to x.

1. What is the *most general type* of **repeat**?
2. In order to reproduce this functionality in C++ or Ada, we must use a template or generic. In either of these two languages, show how to write such an equivalent function.
3. If we restrict f to be a function on integers, the same effect can be obtained in C, using a function pointer. Write this simpler C solution.
4. Java has no notion of pointers. How can you obtain the effect of a method that is a parameter to another method?

Answer 1

```
a) val repeat = fn : ('a-> 'a) -> int -> 'a -> a'
```

The type of x is polymorphic: 'a. The argument f is a function from 'a to 'a.

b) In either language, we need to parametrize a construct (function, class) with a type parameter, and a with function that operates on the type. In Ada this can be written as follows:

```
generic
   type A is private;
   with function f (X : A) return A;
function repeat (N : Integer; X : A) return A;

function repeat (N: Integer; X : A) return A is
begin
   if N > 0 then
       return repeat (N-1, F (X));
   else return X;
   end if;
end repeat;
```

In C++, it is simpler to write a parameterized class that contains the desired function **repeat**. **repeat** is parametrized with a function pointer:

```
template <class A> class repetition  {
   public:
     A repeat ( A (*f)(A) , int N, A x) {
        if (N >0) return repeat (f, N-1, f (x));
        else
           return x;
        }
};
```

c) If the type is known to be integer, all we need is a function parameter, which in C is best written as follows:

```
typedef int (*FPTR) (int);

int repeat (FPTR f, int N, int x) {
   if (N > 0) return repeat (f, N-1, f (x));
   else return x;
}
```

d) In Java, the equivalent of a reference to a method is obtained through dynamic dispatching. Declare an interface I with one method Func, declare a class that implements I and provides a method Func, and use an object of the class in the call to repeat.

Question 2

Consider the following grammar for the language FLAIL (Fortran-Like Artificial-Intelligence Language). The notation {x} means zero or more occurrences of x. The notation [x] means optional x (0 or 1 occurences of x).

```
LETTER ::= a | b | c | .... | y | z
IDENT  ::= LETTER {LETTER}
DIGIT  ::= 0 | 1 | 2 ..     | 8 | 9
LIT    ::= DIGIT {DIGIT} [. DIGIT {DIGIT}]

PROGRAM ::= {DECL}
            STMSEQ
            End

DECL   ::= TYPE IDENT
TYPE   ::= Real | Integer

STMT   ::= ASSIGNSTM | IFSTM | LOOPSTM | BACKSTM
STMSEQ ::= STMT {STMT}

ASSIGNSTM ::= IDENT = EXPR

IFSTM ::= If EXPR Then
             STMSEQ
          Else
             STMSEQ
          Endif

LOOPSTM ::= While EXPR do
               STMSEQ
            Endwhile

BACKSTM ::= Backup
EXPR    ::= PRIMARY | ADD | MUL | SUB | DIV
PRIMARY ::= IDENT | LIT | (EXPR)
ADD     ::= PRIMARY {+ PRIMARY}
MUL     ::= PRIMARY {* PRIMARY}
SUB     ::= PRIMARY - PRIMARY
DIV     ::= PRIMARY / PRIMARY
```

Answer the following questions (see next page) based on this grammar. Questions on FLAIL programs:

1. Identify the *syntactic* errors in the following program

```
Real a
Integer b, c
Integer d
If b > c Then
   a = 1.0
   b = b + 11.
ENDIF
b = b + c * d
d = d - 2 - b
End
```

2. The production

```
IDENT ::= LETTER {LETTER}
```

   would typically be handled by which phase of the compiler?

3. The production

```
STMSEQ ::= STMT {STMT}
```

   would typically be handled by which phase of the compiler?

4. Why is the answer to 2 different from the answer to 3?

5. Given the expression a * b * c, it is unclear from the syntax whether it means (a * b) * c or a * (b * c)

   (a) Given that multiplication is associative, why does it make a difference?

   (b) Suppose that we intend it to mean (a * b) * c. Rewrite the grammar so that this assocation is clear from the grammar.

6. The grammar as it stands is ambiguous. Point out the ambiguity and explain how you would modify the grammar to fix it.

Answer 2

1. Syntax errors:

```
Real a
Integer b, c
         ^
Error: only one identifier per declaration

Integer d
If b > c Then
     ^
Error: No such operator

   a = 1.0
^
Error: Missing Else in If statement

ENDIF
^
Error: Badly spelled keyword, should be Endif

D = d - 2 - b
          ^
Error: multiple subtractions require parens
End
```

2. The production IDENT ::= LETTER LETTER would typically be handled by the lexical analyzer

3. The production STMSEQ ::= STMT STMT would typically be handled by the parser

4. Because IDENT is a token with no internal semantic structure, and can easily be recognized by the finite state processing the lexical analyzer

5. Given the expression a * b * c, it is unclear from the syntax whether it means (a * b) * c or a * (b * c)

   (a) The association (left or right) makes a difference when computing floating point expressions. There are two possible results: overflow, or precision effects. In either case, the result can depend on the order of evaluation.

   (b) To impose left association: (a * b) * c, we can rewrite the grammar

```
Instead of      MUL ::= PRIMARY {* PRIMARY}
Write           MUL ::= PRIMARY | MUL * PRIMARY
```

6. The ambiguity is the following: ADD and MUL can match a PRIMARY by having zero repetitions. So if we see a PRIMARY, we can parse it as an expression three ways.

   (a) Directly as PRIMARY

   (b) As PRIMARY + PRIMARY with zero repetitions

   (c) As PRIMARY * PRIMARY with zero repetitions

Question 3

Consider the following definition of a C++ class that acts like an array with bounds-checking (i.e. ensuring that a valid array index is used each time an array element is referenced).

```
class my_array {
  public:
     my_array(int s): size(s) { a = new int[size]; }
     int &operator[](int i);
     int get_size() { return size; }
  private:
     int size;
     int *a;
};
```

1. Give the code for `my_array`'s `operator[]` method, as one would define it outside the class.

2. Why is it necessary for the return type of `operator[]` to be `int &` ?

3. Suppose there is a procedure `sum()`, not defined within the `my_array` class, as follows:

   ```
   int sum(my_array &a)
   {
     int n = 0;
     int size = a.get_size();
     for(int i=0; i<size; i++) {
       n += a[i];
     }
     return n;
   }
   ```

   Notice that a bounds check will occur every time `a[i]` is evaluated, even though bounds checking isn't really required in this case. Modify the code of `sum()` and of class `my_array` so that a bounds check isn't performed during the execution of `sum()`. Don't make `sum()` a member function of `my_array`, though.

Answer 3

1.
```
int &my_array::operator[](int i)
{
  if ((i < 0) || (i>= size)) {
    cout << "Error: Array reference out of bounds" << endl;
    exit(1);
    }
  else return a[i];
}
```

2. Since an array reference can appear on the left hand side of an assignment, e.g.

   ```
   a[i] = 6;
   ```

   operator[] must return something that can be assigned to, i.e. an "L-value". T he use of `&` specifies that what is being returned by `operator []` is not a copy of the value of `a[i]`, but rather a reference to tt a[i], allowing `a[i]` to be modified.

3. Rather than making the data member `a` of class `my_array` public, which would allow every procedure to avoid bounds checking, we can make `sum()` a friend of `my_array`, and to modify `sum()` to reference the data member `a` directly.

   ```
   class my_array {
     friend int sum(my_array &);
     public:
        my_array(int s): size(s) { a = new int[size]; }
        int &operator[](int i);
        int get_size() { return size; }
     private:
        int size;
        int *a;
   };

   int sum(my_array &a)
   {
     int n = 0;
     int size = a.get_size();
     for(int i=0; i<size; i++) {
       n += a.a[i];
     }
     return n;
   }
   ```