## University of Toronto

**CSC488S/CSC2107S Compilers and Interpreters** **Winter 2009/2010**

**Mid Term Test ( 15% of course mark)** **March 2, 2010.**

**5** questions on **2** pages. **100 marks total. 50 minutes total**
**Open Book and Notes, Non-programmable calculators allowed, NO other electronic aids allowed**
**Answer all questions. WRITE LEGIBLY!**
If you need to make any additional assumptions to answer a question, be sure to state those assumptions in your test booklet.
The line numbers on the left side of programs are for reference only and not part of the program.

**1. [20 marks]** Given the declarations in C:

```
typedef struct {           /* define dataStruct */
            struct {
                 char name[5] ;
                 int  key ;
                 double value ;
               } data ;
            unsigned char tag ;
         }  dataStruct ;

dataStruct A[ 100 ] ;    /* array of dataStruct */
int i = 19 ;
```

Assume char is 8 bits aligned mod 8, int is 32 bits, aligned mod 32, double is 64 bits aligned mod 64 .
Given the base address of the array A, show in detail the address calculation for the subscript reference

```
A[ i + 7 ] . data . value
```

**2. [20 marks]** Given an LL(1) parsing table constructed using the method discussed in lecture and a labelling of the table rows (non terminal symbols) and the table columns (terminal symbols), give an algorithm to reconstruct the Predict sets for a given non terminal symbol $N$ .

**3. [20 marks]** Consider the following declarations in a Turing/Pascal-like language

```
type R : record
         ra : array 1 .. 100  of real
         rb : string ( 5 )
         rc : record
              i , j : 10 .. 45     /* subrange type */
              rcb : boolean
         rd : int
     end record     /* end R declaration */
```

```
var X : R
var Y : record
          ya : string( 1 )
          yb : R
     end record
```

Using a symbol and type table similar to the examples given in lecture, show the symbol and type tables that would be created for these declarations.

4. **[15 marks]** The Python programming language uses *indentation* rather than explicit **begin**/**end** or { } characters to mark the beginning and end of blocks. This includes delimiting the bodies of functions and the bodies of control statements. For example:

```
Python                              Description
    def calc( x ) ;                 define function calc
        n = x * x + 7               assignment statement in calc
        return n * n + 5            return statement in calc
                                    end of calc
    def map ( n , m )               define function map
        if n < m :                  begin body of map
            i = n - m               body of if statement
            j = n + m               if statement continues
            k = i * j               if statement continues
        if n > m :                  start new if statement
            i = n * m + 7           body of if statement
            j = i * 2 + 5           if statement continues
            k = i * j + 1           if statement continues
        return k - 17               end if statement
                                    end of map
    print map( 17, 23 )             start of main program
```

Describe a method for scanning and parsing this language. In particular how would the scanner and parser interact to delimit blocks based on indentation?

5. **[25 marks]** Describe the semantic analysis checks that a Java compiler would perform on the following piece of Java code

```java
class BreakDemo {
    public static void main(String[] args) {
        int[] arrayOfInts = { 32, 87, 3, 589, 12, 1076, 2000, 8, 622, 127 };
        int searchfor = 12;
        int i;
        boolean foundIt = false;
        for (i = 0 ; i < arrayOfInts.length ; i++) {
            if (arrayOfInts[i] == searchfor) {
                foundIt = true;
                break;
            }
        }
        if (foundIt) {
            System.out.println("Found " + searchfor + " at index " + i);
        } else {
            System.out.println(searchfor + " not in the array");
        }
    }
}
```