University of Toronto

CSC 488S Compilers and Interpreters **Winter 2009/2010**

Mid Term Test Solution

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

```
 1        typedef struct {          /* define dataStruct */
 2                    struct {
 3                         char name[5] ;
 4                         int  key ;
 5                         double value ;
 6                       } data ;
 7                    unsigned char tag ;
 8                  }  dataStruct ;
 9
10        dataStruct A[ 100 ] ;    /* array of dataStruct */
11        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

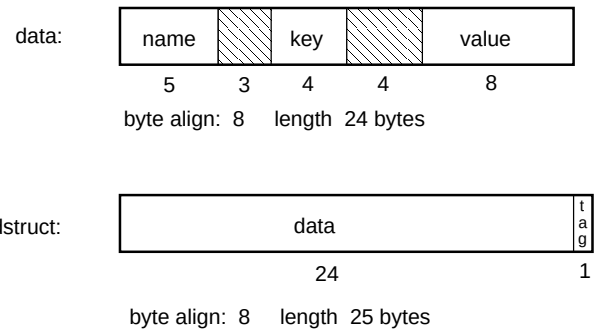
```
A[ i + 7 ] . data . value
```

First layout the structure. It should look like the figure on the right.

Then notice that every array element has to be padded with 7 bytes of fill so that all array elements are properly aligned. So every element of A is 32 bytes long. The subscript calculation is then:

```
A[ i + 7 ].data.value
A[ 26 ].data.value

baseA + ( 26 - 0 ) * 32  + 16
baseA + 832 + 16
baseA + 848
```



Almost everyone noticed the padding of the array elements to 32 bytes. A surprising common error was to mix bits and bytes in the array subscript calculation.

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 .

One possible algorithm is:

- Scan across the parse table row for non terminal symbol N
- Each column entry that isn't an error entry marks an alternative for N
- Collect together all of the entries that are the same. Each collection of entries corresponds to one rule for N
- For the columns corresponding to each of these collections the terminal symbols associated with the column are the members of the Predict set for that rule for N

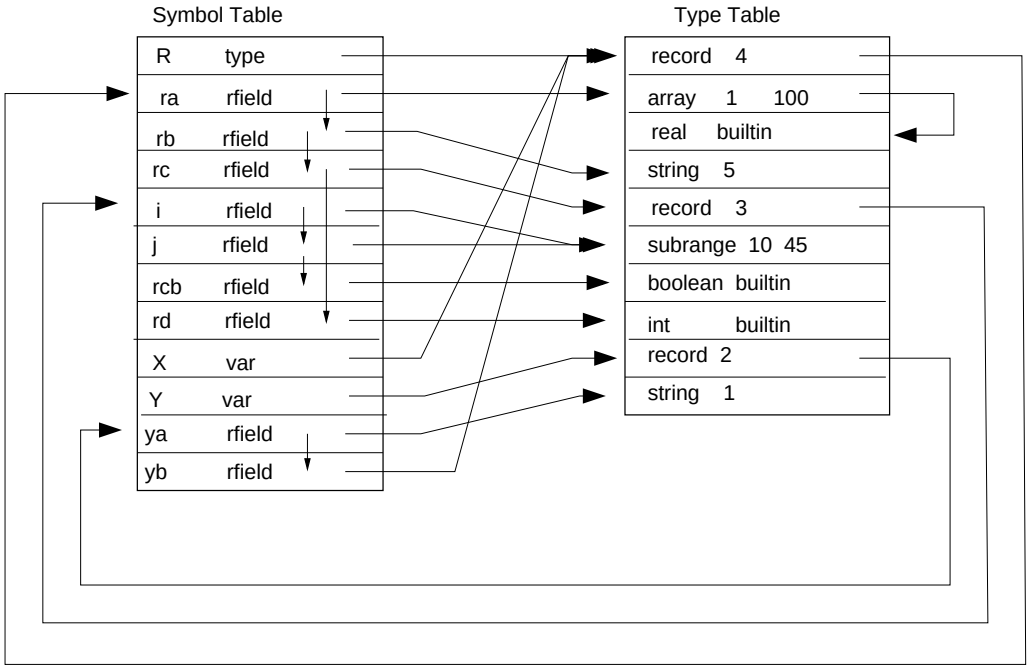
3. [20 marks] Consider the following declarations in a Turing/Pascal-like language

```
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.

One possible solution:



Most common problems were missing links and incomplete tables.

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?

Since the scanner is the only part of the compiler that knows about indentation most of the heavy lifting should be done there. One possible solution.

- Define two lexical tokens <INDENT> and <EXDENT> that are emitted by the scanner whenever the level of indentation changes.
- Define the compilers parsing grammar in terms of these lexical tokens, e.g.
  body $\rightarrow$ <INDENT> statements <EXDENT>
  statement $\rightarrow$ <INDENT> statements <EXDENT>
- Suppress these invented tokens in any compiler error messages

Solutions that involved the parser keeping track of indentation did less well because standard off the shelf parsers couldn't be used.

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

```
 1    class BreakDemo {
 2        public static void main(String[] args) {
 3            int[] arrayOfInts = { 32, 87, 3, 589, 12, 1076, 2000, 8, 622, 127 };
 4            int searchfor = 12;
 5            int i;
 6            boolean foundIt = false;
 7            for (i = 0 ; i < arrayOfInts.length ; i++) {
 8                if (arrayOfInts[i] == searchfor) {
 9                    foundIt = true;
10                    break;
11                }
12            }
13            if (foundIt) {
14                System.out.println("Found " + searchfor + " at index " + i);
15            } else {
16                System.out.println(searchfor + " not in the array");
17            }
18        }
19    }
```

Even this relatively simple piece of code requires a lot of checks

Line(s)	Check
1	Check that class BreakDemo is not already defined
2	Check that main with this argument signature is not already defined
3	Check that arrayOfInts is not already defined
	Check that all initial values are compatible with int
4	Check that searchFor is not already defined
	Check that initial value is compatible with int
5	Check that i not already defined
6	Check that foundIt is not already defined
	Check that initial value is compatible with boolean
7,8	Check that i has been declared, and that it is assignable
	Check that arrayOfInts has been declared
7	Check that arrayOfInts has a field named length
	Check that the type of length is compatible with int
	Check that 0 can be assigned to i
8	Check that arrayOfInts is a one dimensional array
	Check that i is a compatible subscript for arrayOfInts
	Check type compatibility of operands of ==
9,13	Check that foundIt has been declared and is of boolean type
	Check that true can be assigned to foundIt
10	Check that break occurs inside a loop
14,16	Check that class System has been declared
	Check that class System has a field named out
	Check that System.out.println is a defined function
	Check that System.out.println accepts a string argument