CS 415 Compilers: Problem Set 6

Spring 2022

Due date: Wednesday, April 27, 11:59pm

Problem 1 – Out-of-Bounds Type Error

Assume a language that supports 1-dimensional integer arrays, with 0-based indexing. Declarations of the array are in the form of

```
a[c1:c2] of integer
```

where c1 and c2 are compile-time constants, with c1 $\leq$ c2. Give the ILOC code that performs dynamic type checking for array references of the form `a[e]`, where `e` is an integer-valued expression. Use the "new" ILOC instruction `throw-exception` in case the reference is out of bounds. Assume that array is declared as `a[20:90] of integer`, and the right-hand-side reference you need to compile is

```
a[i+1]
```

where `i` is an interger-valued scalar variable with relative address (offset) 4. Assume our usual memory layout with r0 as the base register, containing address 1024.

Problem 2 – Code Generation for Array Expressions

```
var a[80][35][10] of integer;

for (k=0, k<10; skj++) {
  for (j=0, j<35; j++) {
    for (i=0; i<80; i++) {
       = ... a[i][j][k] /*1*/
    }
  }
}
```

Generate ILOC code for the reference to three-dimensional array **a** at program point /*1*/ assuming

1. row-major order (rightmost index has stride 1)
2. column-major order (leftmost index has stride 1)

You do not need to show the enclosing code for the loop.

Assume the following relative addresses (offsets): base address of **a** at 16, The indexing is 0-based, i.e., `a[0][0][0]` is the first array element. Offsets of `i, k, j` are 4, 8, and 12, respectively.

Probem 3 – Lexical Scoping Code Generation

Assume that all variables are lexically scoped.

```
program main()
{    int a, b;
     procedure f()
     {  int c;
        procedure g()
        {
           ... = b + c    //<<<-------- (*A*)
           print a,b,c;
           end g;
        }
        a = 0; c = 1;
        ... = b + c    //<<<-------- (*B*)
        call g();
        print c;
        end f;
     }
     procedure g()
     {  int a,b;
        a = 3; b = 7;
        call f();
        print a,b;
        end g;
     }
  a = 2;  b = 3;
  print a,b;
  call g();
  print a,b;
  end main;
}
```

1. Show the runtime stack with its stack frames, access and control links, and local variables when the execution reaches program point `(*A*)`.
2. Give the ILOC RISC code for the expressions at program points (*A*) and (*B*). The value of the expressions need to be loaded into a register. The particular register numbers are not important here. Use access links to resolve the non-local accesses.