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

\[ a[c1:c2] \text{ of integer} \]

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 \texttt{throw-exception} in case the reference is out of bounds. Assume that array is declared as \( a[20:90] \text{ of integer} \), and the right-hand-side reference you need to compile is

\[ a[i+1] \]

where \( i \) is an integer-valued scalar variable. Assume our usual memory layout with \( r0 \) as the base register, containing address 1024.

Problem 2 – Code Generation for Array Expressions

\begin{verbatim}
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*/
        }
    }
}
\end{verbatim}

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.

You can use the symbolic address \( @a \) to represent the base address of the array. The indexing is 0-based, i.e., \( a[0][0][0] \) is the first array element.
Problem 3 – Intermediate Representation and Code Generation

Assume that the following code is part of the body of a procedure:

```plaintext
i = 0;
sum = 0;
while (i < 10) {
  S1: a[i] = a[i] + 1;
  S2: sum = sum + a[i];
  S3: i = i + 1;
}
print sum;
```

Give the following intermediate representations for the code fragment above:

1. Abstract Syntax Tree (AST)
   There are many ways of doing this. Keep in mind that you want a data structure that allows easy reordering of statements and subexpressions.

2. Control Flow Graph (CFG)
   What representation do you choose for statements within each basic block?

3. Three-Address Code (ILOC)
   You may use any ILOC instruction.
   Note:
   - virtual addresses are byte addresses; all data types are integer (4 bytes)
   - array variable `a` is local, with offset address `@a`.
   - register `r0` contains the “base address”. You do not have to initialize this register.
   - scalar variable `i` and `sum` are local with offset address `@i` and `@sum`, respectively.

4. Stack machine code for statement S2 only