Partial Exam 3

Questions 1.

Fortran and C

Question 2. [4 marks]

   a) x= 1
   list= {3, 2, 0, 1}
   b) x= 2
   list= {3, 1, 1, 0}

Question 3. [3 marks]

   a= [12]
   b= [5, 13]
   c= 14

Question 4. [4 marks]

Any number of segments can be executed in one execution of the construct (there is no implicit branch at the end of selectable segments).

Trade-off: less need of using a “goto”
Question 5. [5 marks]

```
switch (k)
{
    case 1: case 2:
        j = 2 * k - 1;
        break;
    case 3:
        j = 4 * k - 1;
        break;
    case 4: case 5: case 6: case 7:
        j = k - 2;
        break;
    default:
        printf("Error in switch, k =%d\n", k);
}
```

Question 6. [6 marks]

a) $((a * ((b -1)^1 / c)^2)^3 \text{ mod } d)^4$

b) $(a - ((b / (c & (d * (e / a)^1)^2)^3)^4 - 3)^5)^6$

c) $(a > ((b \text{ xor } (c \text{ or } d)^1)^2 \leq 17)^3)^4$
Question 7. [2 marks]

Any two of the following 7 issues are acceptable:

- Are local variables static or dynamic?
- Can subprogram definitions appear in other subprogram definitions? “nested subprograms”?
- What parameter passing methods are provided?
- Are parameter types checked?
- If subprograms can be passed as parameters and subprograms can be nested, what is the referencing environment of a passed subprogram?
- Can subprograms be overloaded?
- Can subprogram be generic?

Question 8. [5 marks]

```plaintext
index = 1;
while (index<=length) && (LIST[index] != value) index++;
```

When index==length, LIST [index] will cause an indexing problem (assuming LIST has length-1 elements)

Question 9. [3 marks]

There are several examples as discussed in class. One of them is as follows:

```plaintext
Subroutine mySub (x: Integer, y: Integer) {
  ...
}
mySub(a[i], a[j]);
```

When i and j are equal, x and y are aliases.
Question 10. [5 marks]

Syntax rule:
<assign> → <var> = <expr>

Semantic rules:
<expr>.expected_type ← <var>.actual_type

Syntax rule:

Semantic rules:
<expr>.actual_type ← <var>[1].actual_type

Predicates:
<var>[1].actual_type == <var>[2].actual_type
<var>[1].actual_type == <var>[3].actual_type
<expr>.expected_type == <expr>.actual_type

Syntax rule:
<var> → A | B | C

Semantic rule:
<var>.actual_type ← lookup (<var>.string)