

E-Genting Programming Competition 2004

Pre-Competition Workshop, Week 5
19 October 2004



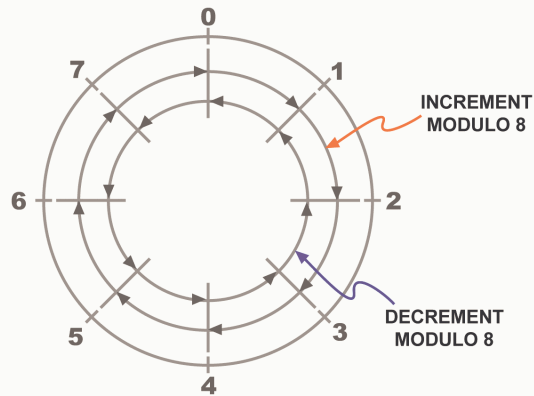
Using Tables

```
static final int  STEP_TABLE[] = {1, 4, 2, 8};
                                     // Step sequence table
static final int  NEXT_PHASE[][] =
    {{1, 2, 3, 0}, {3, 0, 1, 2}};
                                     // Next phase map

// To move one step to the right:
phase = NEXT_PHASE[0][phase];
wiperControl.setSCR(STEP_TABLE[phase]);

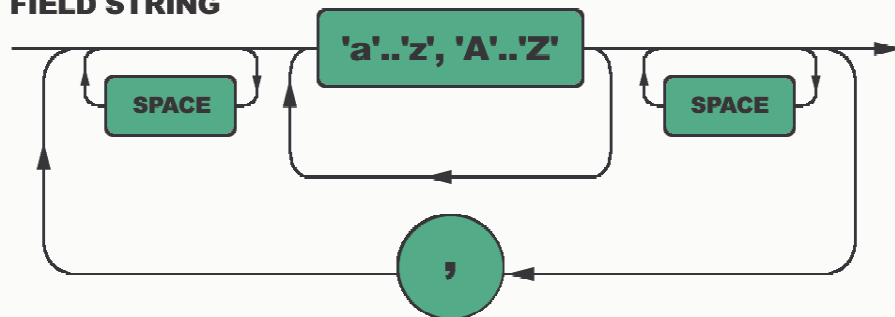
// To move one step to the left:
phase = NEXT_PHASE[1][phase];
wiperControl.setSCR(STEP_TABLE[phase]);
```

Arithmetic in a Cyclic Domain



Syntax Diagram

FIELD STRING



```

sr = new StringReader (fieldString);
ch = sr.read();
for (;;) {
    while (ch == ' ') ch = sr.read();
    if ( ! fieldChar(ch))
        reject ("invalid field identifier");
    sw = new StringWriter ();
    while (fieldChar(ch)) {
        sw.write (ch);
        ch = sr.read();
    }
    while (ch == ' ') ch = sr.read();
    // process the id in sw.toString()
    if (ch != ',') break;
    ch = sr.read();
}
if (ch != -1) reject ("unexpected char");

```



Tailoring Library Classes 1

```

struct Cust_t {
    unsigned long    custId;
    string           custSurname;
    string           custGivenNames;
};

```



Tailoring Library Classes 2

```
class CustIsLess_c {
public:
    bool
    operator () (
        const Cust_t    &c1,
        const Cust_t    &c2)
    const
    {
        if (c1.custSurname < c2.custSurname) return 1;
        if (c1.custSurname > c2.custSurname) return 0;
        return c1.custGivenNames < c2.custGivenNames;
    }
};
```



Tailoring Library Classes 3

```
// Declare a vector of Cust_t structures

vector <Cust_t> custVec(100);

// Instantiate the comparator and execute the sort

CustIsLess_c custIsLess;
sort (custVec.begin(), custVec.end(), custIsLess);

// Alternatively (but essentially the same thing)

sort (custVec.begin(), custVec.end(), CustIsLess_c());
```



Add Function

```
const int PRECISION = 128;

bool
Add (
    bool        acc[PRECISION], // Accumulator
    const bool  op[PRECISION])  // Operand
{
    int        i;                // Index
    bool       c;                // Carry bit
    c = 0;
    for (i = 0; i < PRECISION; i++) {
        acc[i] ^= op[i] ^ c;
        c = (!acc[i] & (op[i]|c)) | (op[i]&c);
    }
    return c;
}
```