Exercise 1: Parallel implementation of longest sequence of 1s using generalized scan

```
int   Array[N];
int   Result[N];
tally nodeval'[P];
tally ltally[P];
forall ( index in ( 0 .. P-1 ) )
{
   int size = mySize( Array[], 0 );
   int myData[size]   = localize( Array[] );
   int myResult[size] = localize( Result[] );
   tally myTally;
   tally ptally;
   // compute the local tally into myTally
   myTally = init( );
   for ( i = 0; i < size; i++ )
          {
          myTally = accum( myTally, myData[i] );
          }
   nodeval'[index] = myTally;
   int stride = 1;
   while ( stride < P )
          {
          if ( index % ( 2 * stride ) == 0 )
                    {
                    ltally[index+stride] = nodeval'[index];
                    nodeval'[index] = combine( ltally[index+stride], nodeval'[index+stride] );
                    stride = 2 * stride;
                    }
          else
                    {
                    break;
                    }
          }
   // root's parent value is the init tally
   if ( index == 0 )
          {
           dummy = nodeval'[0];  // just empty the FE variable
           nodeval'[0] = init( );
          }
   // propagate parent value through the tree to the leaf nodes
   stride = P/2;
   while ( stride >= 1 )
          {
           if ( index % ( stride * 2 ) == 0 )
                    {
                    ptally = nodeval'[index];
                    nodeval'[index] = ptally;
                    nodeval'[index+stride] =  combine( ptally, ltally[index+stride] );
                    }
          stride = stride / 2;
```

```
        }
   // leaf node, generate result for all local elements
   ptally = nodeval'[index];  // get the parent value
   for ( i = 0; i < size; i++ )
        {
                ptally = accum( ptally, myData[i] );
                myResult[i] = scanGen( ptally );
        }
}
typedef struct t{
        int currCount;
        int totalCount;
        int initSeq;
        int start;
}tally;
tally init( )
{
        tally t;
        t.start = 1;
        t.currCount = t.totalCount = t.initSeq = 0 ;
}

tally accum( tally t, int elem)
{
        if (elem == 1){
                if (t.start)
                        initSeq++;
                t.currCount++;
                if (t.currCount > t.totalCount)
                        t.totalCount  = t.currCount;
        }
        else
        {
                t.start = 0;
                t.currCount = 0;

        }
}
tally combine( tally left, tally right )
{
        tally ct;
        ct.initSeq = left.initSeq;
        ct.currCount = right.currCount;
        int combined = left.currCount + right.initSeq;
        ct.totalCount = max(left.totalCount, right.totalCount, combined);

}
int scanGen( tally t)
{
        return t.totalCount;
}
```