parenthesesMaskMemoryEfficient.cpp

```
/* FILE: parenthesesMaskMemoryEfficient.cpp   last change: 30-Jul-2013   author: Romeo Rizzi
 * This solver is a memory efficient implementation of the same DP algorithm proposed in the file pa
renthesesMask.cpp.
 * This extra-efficiency is gained by observing that the entries in one row of the DP matrix depend
only on the entries in the previous row.
 */

//#define NDEBUG   // NDEBUG definita nella versione che consegno
#include <cassert>
#ifndef NDEBUG
#  include <iostream>  // uso di cin e cout non consentito in versione finale
#endif
#include <fstream>

using namespace std;

const int BASE  = 100000;
const int MAX_N = 500000;  // massima lunghezza della stringa in input
const int MAX_JOLLIES = 1000;  // massimo numero di asterischi nella stringa in input
char p[MAX_N+1]; // la stringa pattern data in input
int n;  // lunghezza della stringa pattern in input
int n_open, n_closed, n_jollies;

int opt[2][MAX_JOLLIES +1]; /* see the file parenthesesMask.cpp
where we have opt[MAX_N+1][MAX_JOLLIES +1] and where:
opt[i][j] = the number of strings s over the alphabet {'(',')'} such that:
1. s is a prefix of a well formed parenthesis formula;
2. s matches the prefix p[1,..., i] of the pattern;
3. in s the number of ')' matching with an '*' in p is precisely j.
Thus opt[n][n/2 - n_closed] is the answer to our problem.
The values actually stored in opt[][] are correct only modulo BASE.
*/

int main() {
  ifstream fin("input.txt"); assert( fin );
  fin >> n;
  for(int i = 1; i <= n; i++)  fin >> p[i];
  fin.close();
  opt[0][0] = 1;  for(int j = 1; j <= MAX_JOLLIES; j++)  opt[0][j] = 0;
  n_open = 0; n_closed = 0;
  for(int i = 1; i <= n; i++) {
     switch( p[i] ) {
           case ')': n_closed++; break;
           case '(': n_open++; break;
           case '*': n_jollies++; break;
     default: cout << i << " " << p[i] << endl; assert( false );
      }
     for(int j = 0; j <= n_jollies; j++) {
       if(j + n_closed > (n_jollies -j) + n_open) opt[i%2][j] = 0;
       else switch( p[i] ) {
           case ')': opt[i%2][j] = opt[(i-1)%2][j]; break;
           case '(': opt[i%2][j] = opt[(i-1)%2][j]; break;
           case '*': opt[i%2][j] = opt[(i-1)%2][j]; if( j > 0 ) opt[i%2][j] = (opt[i%2][j] + opt[(i-
1)%2][j-1]) % BASE; break;
       }
       // cout << i << " " << j << " " << opt[i%2][j] << " " << p[i] << " " << opt[(i-1)%2][j] << "
" << opt[(i-1)%2][j-1] << endl;
     }
  }

  //  for(int i = 0; i <= n; i++) {
  //    for(int j = 0; j <= n - n_open - n_closed; j++)  cout << opt[i][j] << " ";
  //    cout << endl;
  //  }

  ofstream fout("output.txt");
  fout << opt[n%2][n/2 - n_closed] % BASE << endl;
  fout.close();
  return 0;
}
```