El algoritmo CYK

Teoría de la computación

Deberá implementar el algoritmo CYK y hacer la prueba para algunas cadenas.

Gramática

Para aplicar su algoritmo en la prueba de cadenas, deberá codificar en su programa la Forma Normal de Chomsky (**FNC**) de la siguiente gramática y determinar si sus cadenas están o no en el lenguaje. Esta es la gramática que se trabajó en la práctica anterior, por lo que deberá usar el resultado que obtuvo.

$$
\begin{aligned}
M &\to A \mid MA \\
A &\to V \texttt{ : } Q \texttt{ ;} \\
Q &\to P \texttt{ ? } Q \texttt{ : } Q \mid P \\
P &\to P \texttt{ | } T \mid T \\
T &\to T \texttt{ \& } F \mid F \\
F &\to \texttt{(} Q \texttt{)} \mid \texttt{! } F \mid V \mid B \\
V &\to \texttt{p} \mid \texttt{q} \mid \texttt{r} \\
B &\to \texttt{0} \mid \texttt{1}
\end{aligned}
$$

Para la codificación de los símbolos gramaticales, deberá usar valores enteros (`int`). Se sugiere usar el valor ASCII para los símbolos de la gramática. Para los símbolos auxiliares puede usar valores por encima de este rango. Por ejemplo:

- Un símbolo auxiliar que represente a otro puede codificarse sumándole 1000. Así, si $C_1 \to \texttt{p}$ es una producción, entonces C_1 se puede codificar como `'p' + 1000`.
- Un símbolo auxiliar que se usa para dividir una regla sintáctica puede codificarse con un valor consecutivo a partir de 2000.

En la **FNC**, las producciones son de la forma $N \times \Sigma$ o $N \times NN$, por lo que puede usarse una estructura como la siguiente:

```
typedef struct {
  int A;
  int B;
  int C;
} rule;
```

Finalmente, se sugiere usar un vector[1] para almacenar la colección de reglas de la **FNC** de su gramática.

```
typedef std::vector<rule> grammar;
```

Implementación del algoritmo

Para la codificación de los conjuntos N_{ij} se sugiere usar la clase set[2]:

```
typedef std::set<int> set;
```

Por otro lado, para la codificación de la tabla dinámica puede usar mapeos[3] que relacionen valores de *posición* (i) y *longitud* (j) con el conjunto correspondiente.

i \ j	1	2	3	4
1	$N_{1,1}$	$N_{1,2}$	$N_{1,3}$	$N_{1,4}$
2	$N_{2,1}$	$N_{2,2}$	$N_{2,3}$	
3	$N_{3,1}$	$N_{3,2}$		
4	$N_{4,1}$			

```
typedef std::map<int,set> row;
typedef std::map<int,row> tableau;

tableau N;
```

En la primer definición, se establece que una fila es una colección (indizada) de conjuntos, mientras que en la segunda se establece que la tabla es, a su vez, una colección (indizada) de filas. De esta manera, el conjunto N_{ij} podrá nombrarse `N[i][j]`, en donde el primer índice (i) hace referencia a alguna fila y el segundo (j) hace referencia a un conjunto dentro de ella.

Por último, escriba su algoritmo en una función con el siguiente prototipo:

```
bool cyk(grammar &g, string &w);
```

Haga que esta función retorne `true` si la prueba es positiva y `false` en el caso contrario.