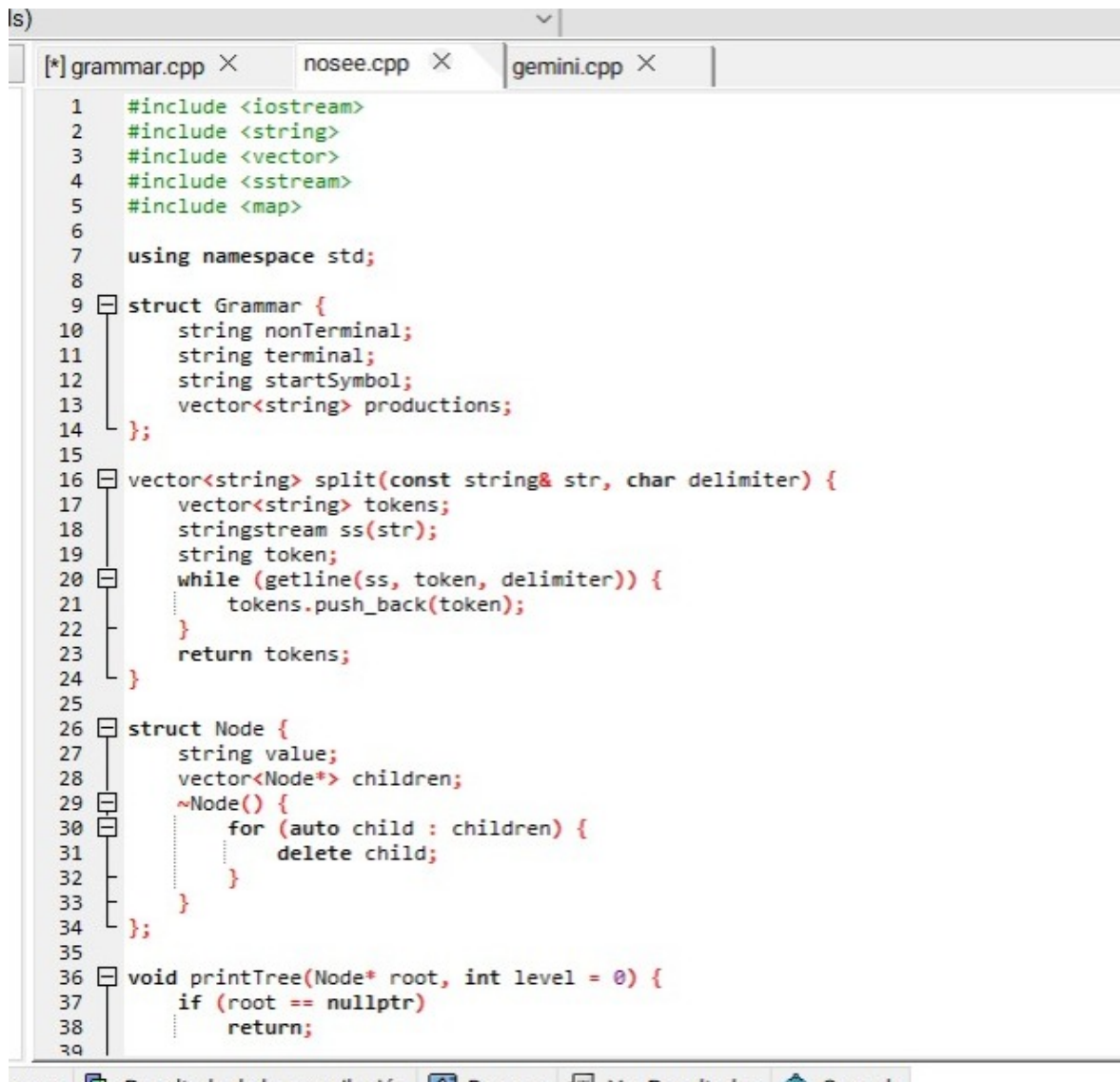


# 1 Grammar



```
ls)
[*] grammar.cpp x  nosee.cpp x  gemini.cpp x

1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #include <sstream>
5  #include <map>
6
7  using namespace std;
8
9  struct Grammar {
10     string nonTerminal;
11     string terminal;
12     string startSymbol;
13     vector<string> productions;
14 };
15
16 vector<string> split(const string& str, char delimiter) {
17     vector<string> tokens;
18     stringstream ss(str);
19     string token;
20     while (getline(ss, token, delimiter)) {
21         tokens.push_back(token);
22     }
23     return tokens;
24 }
25
26 struct Node {
27     string value;
28     vector<Node*> children;
29     ~Node() {
30         for (auto child : children) {
31             delete child;
32         }
33     }
34 };
35
36 void printTree(Node* root, int level = 0) {
37     if (root == nullptr)
38         return;
39 }
```

```
grammar.cpp × noseecpp × gemini.cpp ×
40     for (int i = 0; i < level; i++)
41         cout << " ";
42
43     cout << root->value << endl;
44
45     for (auto child : root->children)
46         printTree(child, level + 1);
47 }
48
49 int main() {
50     Grammar grammar;
51     cout << "Introduzca simbolos no terminales (separados por comas): ";
52     getline(cin, grammar.nonTerminal);
53     cout << "Ingresa los simbolos terminales (separados por comas) ";
54     getline(cin, grammar.terminal);
55     cout << "Introduzca el simbolo de arranque: ";
56     getline(cin, grammar.startSymbol);
57     cout << "Introduzca producciones (separadas por comas): ";
58     string productionsStr;
59     getline(cin, productionsStr);
60     grammar.productions = split(productionsStr, ',');
61
62     // Imprime la gramática
63     cout << "Gramatica:" << endl;
64     cout << "Simbolos no terminales: " << grammar.nonTerminal << endl;
65     cout << "Simbolos terminales: " << grammar.terminal << endl;
66     cout << "Simbolo de arranque: " << grammar.startSymbol << endl;
67     cout << "Producciones:" << endl;
68     for (const auto& production : grammar.productions) {
69         cout << production << endl;
70     }
71
72     // Construye el árbol de derivación
73     map<string, vector<string>> prodMap;
74     for (const auto& production : grammar.productions) {
75         string lhs = production.substr(0, production.find("->"));
76         string rhs = production.substr(production.find("->") + 2);
77         prodMap[lhs].push_back(rhs);
78     }
```

Resultado de la compilación Depurar Ver Resultados Console

```

[*] grammar.cpp x  nose.cpp x  gemini.cpp x
79
80     cout << "Ingrese una cadena para generar el arbol de derivacion: ";
81     string cadena;
82     getline(cin, cadena);
83
84     Node* root = new Node(grammar.startSymbol);
85     vector<Node*> currentLevel = {root};
86     for (int i = 0; i < cadena.length(); i++) {
87         vector<Node*> nextLevel;
88         for (auto node : currentLevel) {
89             if (node->value.length() == 1 && grammar.nonTerminal.find(node->value[0]) != string::npos)
90                 string var = node->value;
91                 vector<string> productions = prodMap[var];
92                 for (const auto& prod : productions) {
93                     bool allTerminals = true;
94                     vector<Node*> newNodes;
95                     for (char c : prod) {
96                         if (grammar.terminal.find(c) != string::npos) {
97                             newNodes.push_back(new Node(string(1, c)));
98                         } else if (grammar.nonTerminal.find(c) != string::npos) {
99                             allTerminals = false;
100                             newNodes.push_back(new Node(string(1, c)));
101                         }
102                     }
103                     if (allTerminals) {
104                         for (auto newNode : newNodes) {
105                             node->children.push_back(newNode);
106                         }
107                     } else {
108                         Node* prodNode = new Node(prod);
109                         for (auto newNode : newNodes) {
110                             prodNode->children.push_back(newNode);
111                         }
112                         node->children.push_back(prodNode);
113                         nextLevel.push_back(prodNode);
114                     }
115                 }
116             } else {
117                 nextLevel.push_back(node);
118             }
119         }
120         currentLevel = nextLevel;
121     }
122
123     // Imprime el árbol de derivación
124     cout << "Arbol de derivacion:" << endl;
125     printTree(root);
126
127     delete root;
128
129     return 0;
130 }

```

recursos Resultado de la compilación Depurar Ver Resultados Console

```
S | C:\Users\YEYITODELPERU\Documents\nosee.exe
Introduzca simbolos no terminales (separados por comas): S,A,B
Ingrese los simbolos terminales (separados por comas) a,b
Introduzca el simbolo de arranque: S
Introduzca producciones (separadas por comas): S->aA,A->bB,B->ab
Gramatica:
Simbolos no terminales: S,A,B
Simbolos terminales: a,b
Simbolo de arranque: S
Producciones:
S->aA
A->bB
B->ab
Ingrese una cadena para generar el arbol de derivacion: ab
Arbol de derivacion:
S
  aA
    a
    A

-----
Process exited after 15.91 seconds with return value 0
Presione una tecla para continuar . . .
```

```
iler paths - Output Filename: C:\Users\YEYITODELPERU\Documents\nosee.exe
           - Output Size: 2.46728706359863 MiB
           - Compilation Time: 1.42s
```