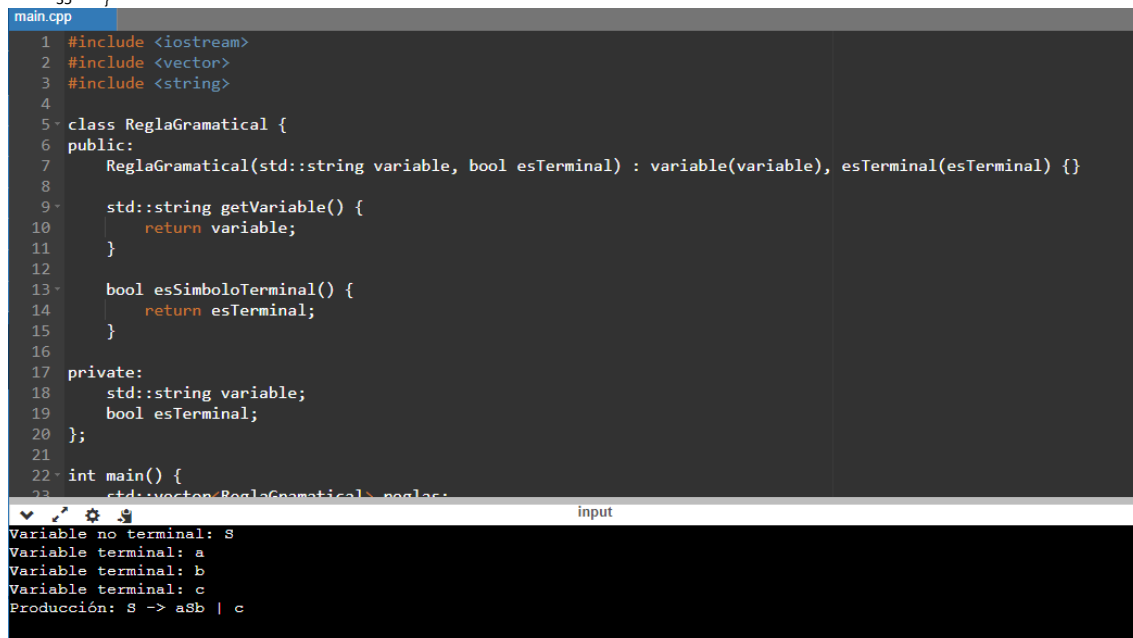


INGENIERIA ESTADISTICA E INFORMATICA

CREAR UN PROGRAMA CON COMPONENTES GRAMATICALES

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  class ReglaGramatical {
5  public:
6  ReglaGramatical(std::string variable, bool esTerminal) : variable(variable), esTerminal(esTerminal) {}
7  std::string getVariable() {
8  return variable;
9  }
10 bool esSimboloTerminal() {
11 return esTerminal;
12 }
13 private:
14 std::string variable;
15 bool esTerminal;
16 };
17 int main() {
18 std::vector<ReglaGramatical> reglas;
19 // Definir las reglas gramaticales
20 reglas.push_back(ReglaGramatical("S", false)); // Variable no terminal
21 reglas.push_back(ReglaGramatical("a", true)); // Variable terminal
22 reglas.push_back(ReglaGramatical("b", true));
23 reglas.push_back(ReglaGramatical("c", true));
24 // Mostrar las reglas gramaticales
25 for (int i = 0; i < reglas.size(); i++) {
26 if (reglas[i].esSimboloTerminal()) {
27 std::cout << "Variable terminal: " << reglas[i].getVariable() << std::endl;
28 } else {
29 std::cout << "Variable no terminal: " << reglas[i].getVariable() << std::endl;
30 }
31 }
32 // Definir producción
33 std::cout << "Producción: S -> aSb | c" << std::endl;
34 return 0;
35 }
```



```
main.cpp
1  #include <iostream>
2  #include <vector>
3  #include <string>
4
5  class ReglaGramatical {
6  public:
7      ReglaGramatical(std::string variable, bool esTerminal) : variable(variable), esTerminal(esTerminal) {}
8
9      std::string getVariable() {
10         return variable;
11     }
12
13     bool esSimboloTerminal() {
14         return esTerminal;
15     }
16 private:
17     std::string variable;
18     bool esTerminal;
19 };
20
21 int main() {
22     std::vector<ReglaGramatical> reglas;
23     reglas.push_back(ReglaGramatical("S", false)); // Variable no terminal
24     reglas.push_back(ReglaGramatical("a", true)); // Variable terminal
25     reglas.push_back(ReglaGramatical("b", true));
26     reglas.push_back(ReglaGramatical("c", true));
27     // Mostrar las reglas gramaticales
28     for (int i = 0; i < reglas.size(); i++) {
29         if (reglas[i].esSimboloTerminal()) {
30             std::cout << "Variable terminal: " << reglas[i].getVariable() << std::endl;
31         } else {
32             std::cout << "Variable no terminal: " << reglas[i].getVariable() << std::endl;
33         }
34     }
35     // Definir producción
36     std::cout << "Producción: S -> aSb | c" << std::endl;
37     return 0;
38 }
```

Variable no terminal: S
Variable terminal: a
Variable terminal: b
Variable terminal: c
Producción: S -> aSb | c

COMPONENTES GRAMATICALES

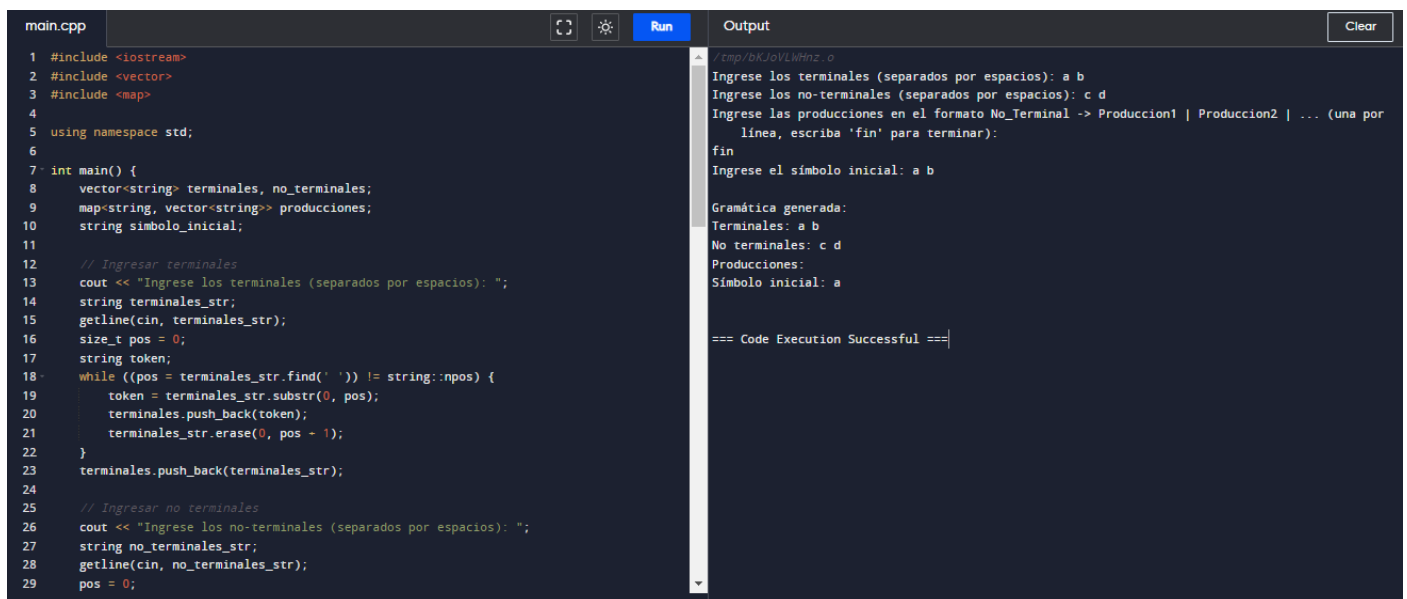
```

1  #include <iostream>
2  #include <vector>
3  #include <map>
4  using namespace std;
5  int main() {
6  vector<string> terminales, no_terminales;
7  map<string, vector<string>>> producciones;
8  string simbolo_inicial;
9  // Ingresar terminales
10 cout << "Ingrese los terminales (separados por espacios): ";
11 string terminales_str;
12 getline(cin, terminales_str);
13 size_t pos = 0;
14 string token;
15 while ((pos = terminales_str.find(' ')) != string::npos) {
16 token = terminales_str.substr(0, pos);
17 terminales.push_back(token);
18 terminales_str.erase(0, pos + 1);
19 }
20 terminales.push_back(terminales_str);
21 // Ingresar no terminales
22 cout << "Ingrese los no-terminales (separados por espacios): ";
23 string no_terminales_str;
24 getline(cin, no_terminales_str);
25 pos = 0;
26 while ((pos = no_terminales_str.find(' ')) != string::npos) {
27 token = no_terminales_str.substr(0, pos);
28 no_terminales.push_back(token);
29 no_terminales_str.erase(0, pos + 1);
30 }
31 no_terminales.push_back(no_terminales_str);
32 // Ingresar producciones
33 cout << "Ingrese las producciones en el formato No_Terminal -> Produccion1 | Produccion2 | ... (una por línea, escriba 'fin' para
terminar):" << endl;
34 string produccion_str;
35 while (true) {
36 getline(cin, produccion_str);
37 if (produccion_str == "fin") {
38 break;
39 }
40 string no_terminal, produccion;
41 pos = produccion_str.find(">");
42 no_terminal = produccion_str.substr(0, pos);
43 produccion = produccion_str.substr(pos + 3);
44 pos = 0;
45 while ((pos = produccion.find('|')) != string::npos) {
46 token = produccion.substr(0, pos);
47 producciones[no_terminal].push_back(token);
48 produccion.erase(0, pos + 2);
49 }
50 producciones[no_terminal].push_back(produccion);
51 }
52 // Ingresar símbolo inicial
53 cout << "Ingrese el símbolo inicial: ";
54 cin >> simbolo_inicial;
55 // Mostrar la gramática generada
56 cout << endl << "Gramática generada:" << endl;
57 cout << "Terminales: ";
58 for (string terminal : terminales) {
59 cout << terminal << " ";
60 }
61 cout << endl;
62 cout << "No terminales: ";

```

INGENIERIA ESTADISTICA E INFORMATICA

```
63 for (string no_terminal : no_terminales) {
64     cout << no_terminal << " ";
65 }
66 cout << endl;
67 cout << "Producciones:" << endl;
68 for (auto const& [no_terminal, produccion] : producciones) {
69     cout << no_terminal << " -> ";
70     for (string p : produccion) {
71         cout << p << " | ";
72     }
73     cout << endl;
74 }
75 cout << "Símbolo inicial: " << simbolo_inicial << endl;
76 return 0;
77 }
```



```
main.cpp
1 #include <iostream>
2 #include <vector>
3 #include <map>
4
5 using namespace std;
6
7 int main() {
8     vector<string> terminales, no_terminales;
9     map<string, vector<string>> producciones;
10    string simbolo_inicial;
11
12    // Ingresar terminales
13    cout << "Ingrese los terminales (separados por espacios): ";
14    string terminales_str;
15    getline(cin, terminales_str);
16    size_t pos = 0;
17    string token;
18    while ((pos = terminales_str.find(' ')) != string::npos) {
19        token = terminales_str.substr(0, pos);
20        terminales.push_back(token);
21        terminales_str.erase(0, pos + 1);
22    }
23    terminales.push_back(terminales_str);
24
25    // Ingresar no terminales
26    cout << "Ingrese los no-terminales (separados por espacios): ";
27    string no_terminales_str;
28    getline(cin, no_terminales_str);
29    pos = 0;
```

```
Output
/cmp/bKJovLWHz: 0
Ingrese los terminales (separados por espacios): a b
Ingrese los no-terminales (separados por espacios): c d
Ingrese las producciones en el formato No_Terminal -> Produccion1 | Produccion2 | ... (una por
línea, escriba 'fin' para terminar):
fin
Ingrese el símbolo inicial: a b

Gramática generada:
Terminales: a b
No terminales: c d
Producciones:
Símbolo inicial: a

=== Code Execution Successful ===
```