INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Cómputo

“PRÁCTICA 06-GLC Limpia y Bien Formada”

- **Alumnos:**
  - **Hernández Escobedo Fernando**
  - **Zanabria Ruiz Luis David**

- **Boletas:**
  - **2017630751**
  -

  - **Materia: Teoría Computacional**

  - **Profesora: Luz María Sánchez García**

    - **Fecha de entrega: mayo de 2018**

Introducción

Las GLC (Gramáticas Libres del Contexto) o GIC (Gramáticas Independientes del Contexto) son llamadas también “Gramática en la Forma de Backus-Naur (BNF)” (usado para describir lenguajes de programación). Las GLC se usan para inferir si ciertas cadenas están en el lenguaje expresado por la gramática. Hay dos tipos de inferencia: Inferencia recursiva (cuerpo a cabeza/de cadenas a variables). Derivación (cabeza a cuerpo, expansión de producciones). Como toda gramática se definen mediante una cuádrupla G = (VN, VT, P, S), siendo;

- VN es un conjunto finito de símbolos no terminales

- VT es un conjunto finito de símbolos terminales, $N \cap T = \emptyset$

- P es un conjunto finito de producciones

- S es el símbolo distinguido o axioma $S \notin (N \cup T)$.

Una gramática nos proporciona un conjunto de producciones que nos permitirán obtener cadenas de un determinado lenguaje.

Al diseñar la gramática nuestra principal preocupación es asegurar que se producen todas las cadenas.

De hecho, puede ocurrir que en el proceso de diseño se introduzcan producciones que no facilitan la obtención de cadenas o que introducen pasos innecesarios en el proceso de derivación.

El objetivo al simplificar una gramática de contexto libre es obtener una gramática equivalente, pero en la que se asegura que cada derivación es útil, y que no obliga a aplicar ningún paso innecesario.

Es decir que no halla reglas muertas(inútiles), inaccesibles y vacías(épsilon).

Y bien partiendo de esta pequeña introducción y puntos es que nos basamos para cumplir el objetivo siguiente: hacer un programa que deberá aceptar una GLC mediante el teclado y posteriormente mostrarla LIMPIA, es decir, se realizará la limpieza de la misma para obtener una GLC limpia y bien formada.

Planteamiento del problema

➔**Descripción:**

•Cualquier lenguaje de contexto libre, L, puede ser generado por medio de una GCL, G, que cumpla las siguientes condiciones:

1. Cada símbolo (terminal o auxiliar) de G se emplea en la derivación de alguna cadena de L.
2. Si $\varepsilon \notin L$, entonces en el conjunto de producciones de G no existen producciones vacías, es decir, producciones de la forma A -> ε.
3. En el conjunto de producciones de G no existen producciones unitarias, es decir, producciones de la forma A -> B donde A, B ϵ Σ.

•Si se obtiene una gramática que cumpla estas tres condiciones se puede asegurar que en cada derivación que se realiza se introduce información relevante.

➔**Objetivo:**

Poder estar seguros de que al final después de ya haberse:

- Eliminado los símbolos inútiles.
- Eliminado las producciones vacías.
- Eliminado las producciones unitarias.

Entonces;

- Cada símbolo (terminal o auxiliar) de G se emplea en la derivación de alguna cadena de L.
- Si $\varepsilon \notin L$ entonces en el conjunto de producciones de G no existen producciones de la forma A ->ε.
- En el conjunto de producciones de G no existen producciones unitarias

➔**Requisitos:**

•El programa deberá estar escrito en cualquier lenguaje de programación.

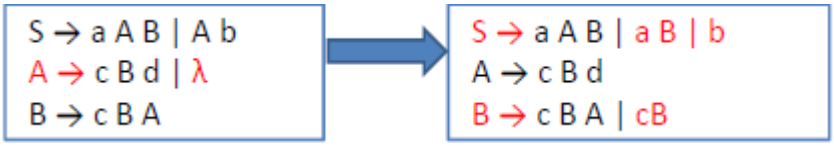
•Se programará de forma individual o por parejas.

•El código deberá estar documentado, colocando en la parte superior del código fuente autor, grupo y fecha.

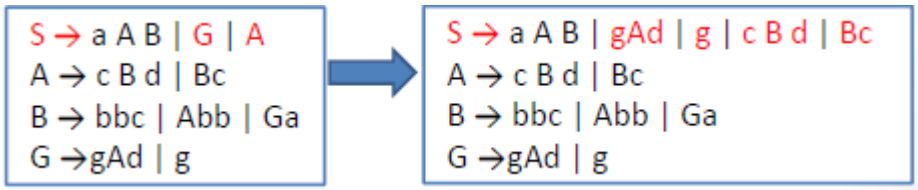
•Se compilará y ejecutará el código en el laboratorio de cómputo.

•Una vez que se dio el VoBo se deberá mandar el reporte de la práctica al espacio designado en la plataforma classroom.

- **Para la implementación de esta práctica se tomaron en cuenta algunos algoritmos, teoremas y condiciones:**
  - Las gramáticas bien formadas, además de ser limpias no incluyen reglas no generativas ni reglas de redenominación:
    - Eliminación de las reglas no generativas; Para eliminar reglas no generativas, se deberá de sustituir las eliminaciones no generativas por aquellas que dejan la gramática con el mismo sentido.



    - Eliminación de las reglas de redenominación; Para eliminar reglas de redenominación, se deberá de sustituir las reglas de redenominación por reglas que dejan la gramática con el mismo sentido.



  - Para realizar la limpieza de una gramática, se consideran los siguientes principios:
    - Teorema 1: si todos los símbolos de la parte derecha de una producción son símbolos vivos, entonces el símbolo de la parte izquierda también lo es.
    - Teorema 2: si el símbolo no-terminal de la parte izquierda de una producción es accesible, entonces todos los símbolos de la parte derecha también lo son.
  - Algoritmo para detectar símbolos muertos:
    1. Hacer una lista de no-terminales que tengan al menos una producción con sólo símbolos terminales en la parte derecha.
    2. Dada una producción, si todos los no-terminales de la parte derecha pertenecen a la lista, entonces podemos incluir en la lista al no-terminal de la parte izquierda de la producción.
    3. Cuando ya no se puedan incluir más símbolos en la lista mediante la aplicación del paso 2, la lista contendrá los símbolos no-terminales vivos y el resto serán símbolos muertos.

- Algoritmo para detectar símbolos inaccesibles:
    1. Se inicializa una lista de no-terminales que sabemos que son accesibles con el axioma.
    2. Si la parte izquierda de una producción está en la lista, entonces se incluye en la misma al no-terminal que aparece en la parte derecha de la producción.
    3. Cuando ya no se pueden incluir más símbolos a la lista mediante la aplicación del paso 2, entonces la lista contendrá todos los símbolos accesibles y el resto de los no terminales serán inaccesibles.

Algunos ejemplos de GLC sucias:

==>Gramática de ejemplo para limpiar:

Ejemplo 1:

Vocabulario no terminal={S, A, B, C}
Vocabulario terminal={a,b,c,d}
P1->abS | abA | abB
P2->cd
P3->aB
P4->dc

Ejemplo 2:

Vocabulario no terminal={S,A,B,C,D,E,F,G,H}
Vocabulario terminal={a,b,c,d,e,f,g,h,x,y,z,t}
P1->aAB|aA|cBd|cd|Ha|bH|AH|cB|c|FG
P2->cBd|cd|Ha|bH|AH|cB|c
P3->e|fS
P4->gD|hDt
P5->x|y|z
P6->AH|cB|c
P7->AB|cBd|cd|Ha|bH|AH|cB|c|Ga
P8->FG
P9->Ha|bH|AH|cB|c

**********SE LIMPIAN EN LA PARTE DE **FUNCIONAMIENTO**********

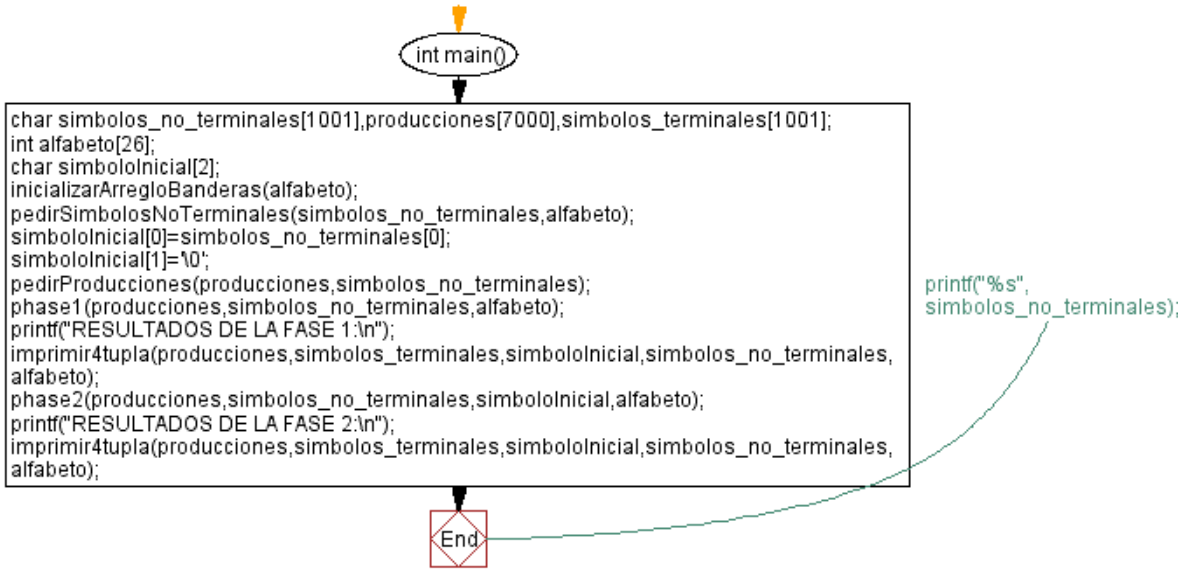
➔**Sea la cuádrupla G = (VN, VT, P, S)**

El usuario podrá ingresar de cada una:

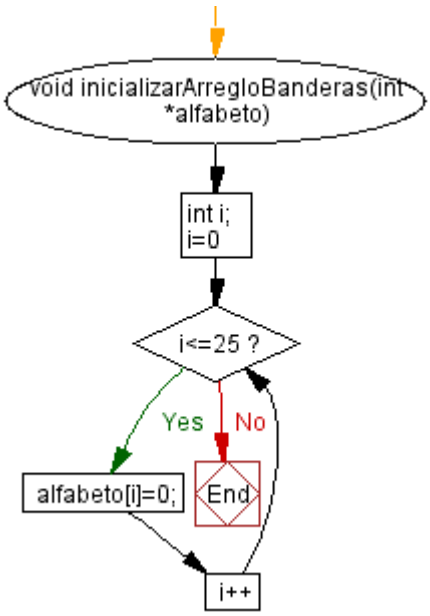
- **VN={..Los que el usuario desee..}**
- **VT={..Los que el usuario desee..}**
- **P={..Los que el usuario desee..}**
- **S={Se le da al usuario después de haber ingresado sus producciones}**

Diseño de la solución

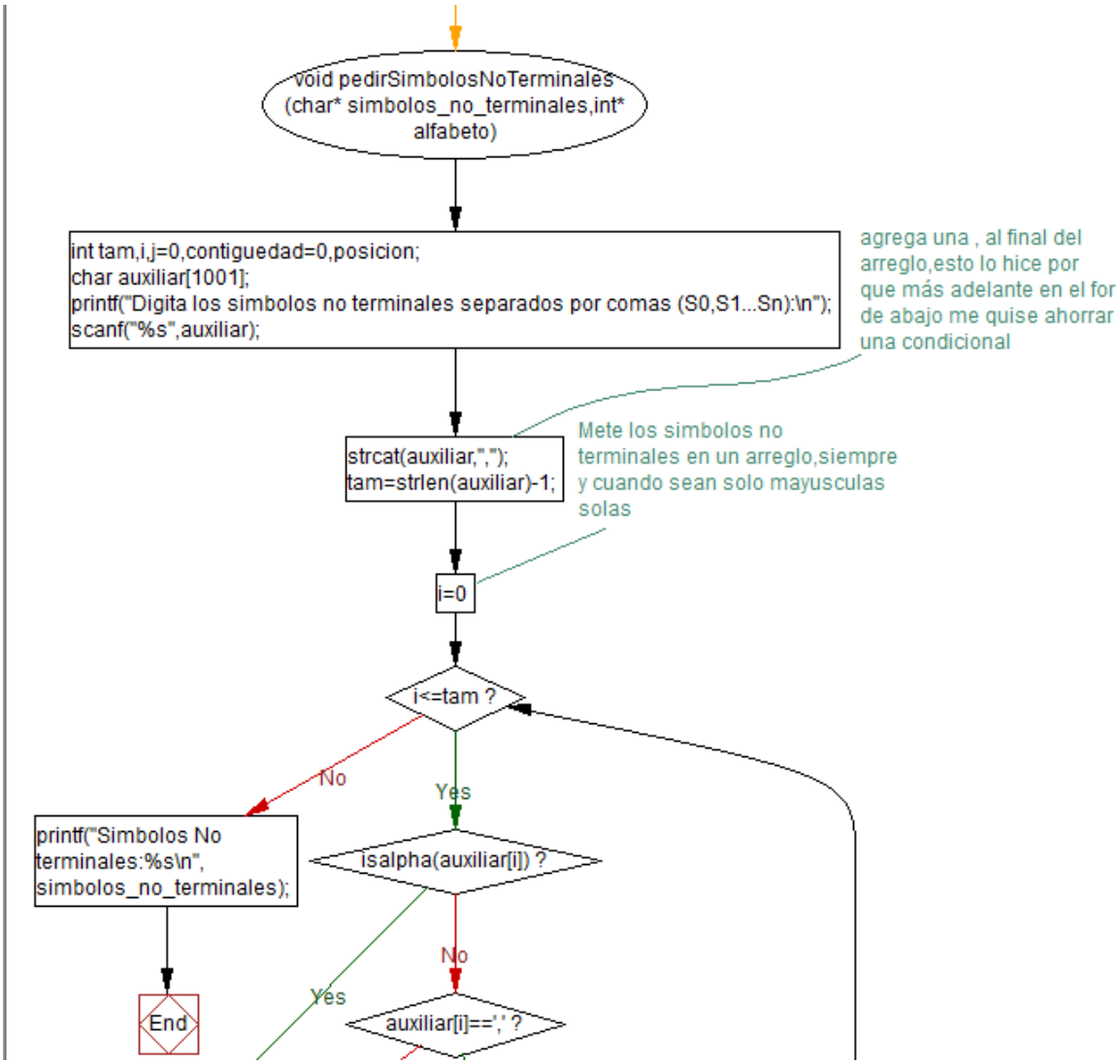
➔Main

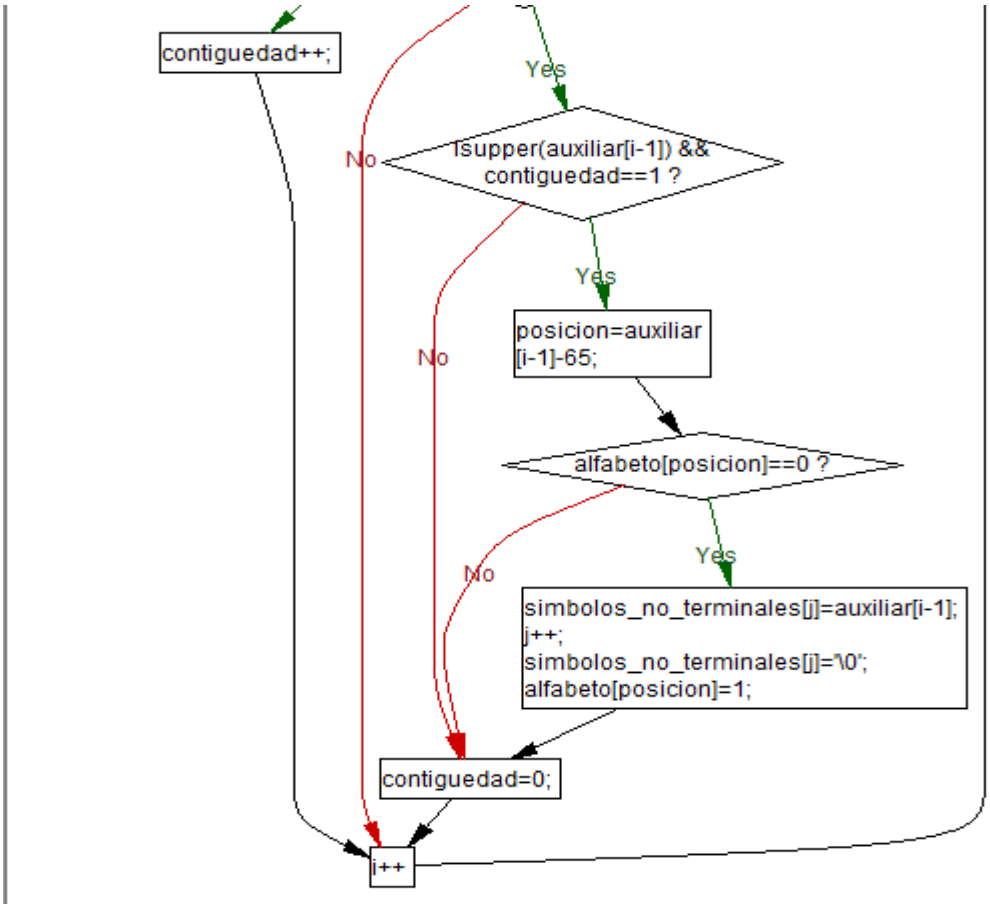


➔ void inicializarArregloBanderas

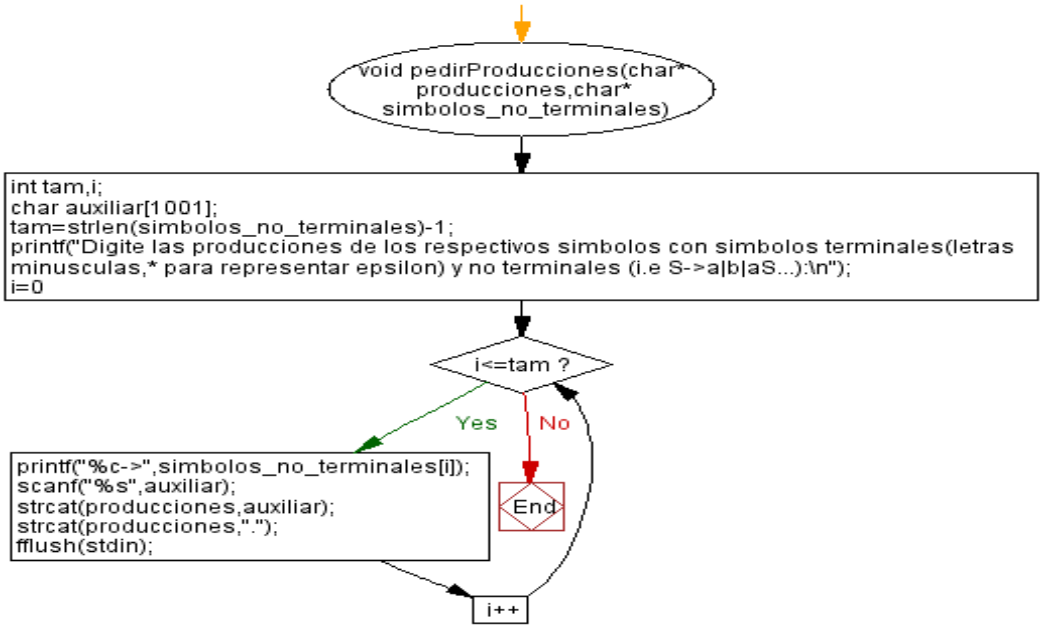


➔ void pedirSimbolosNoTerminales

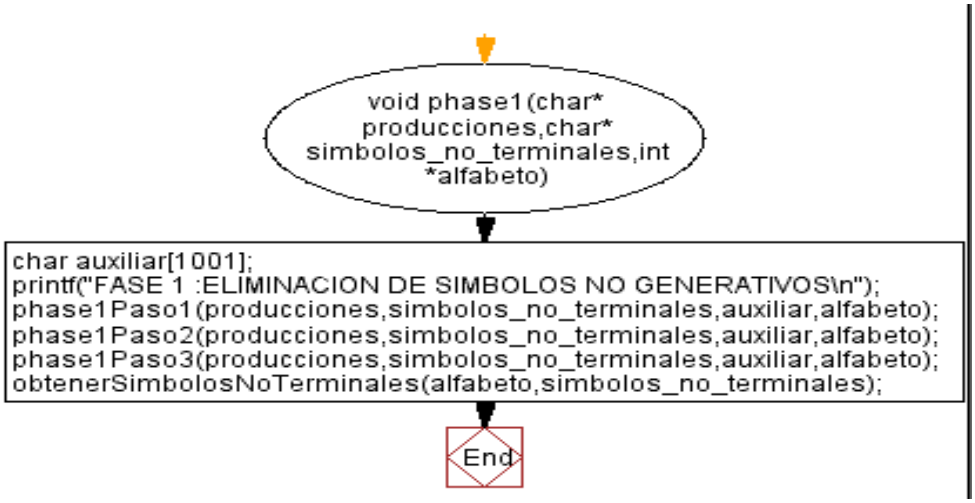




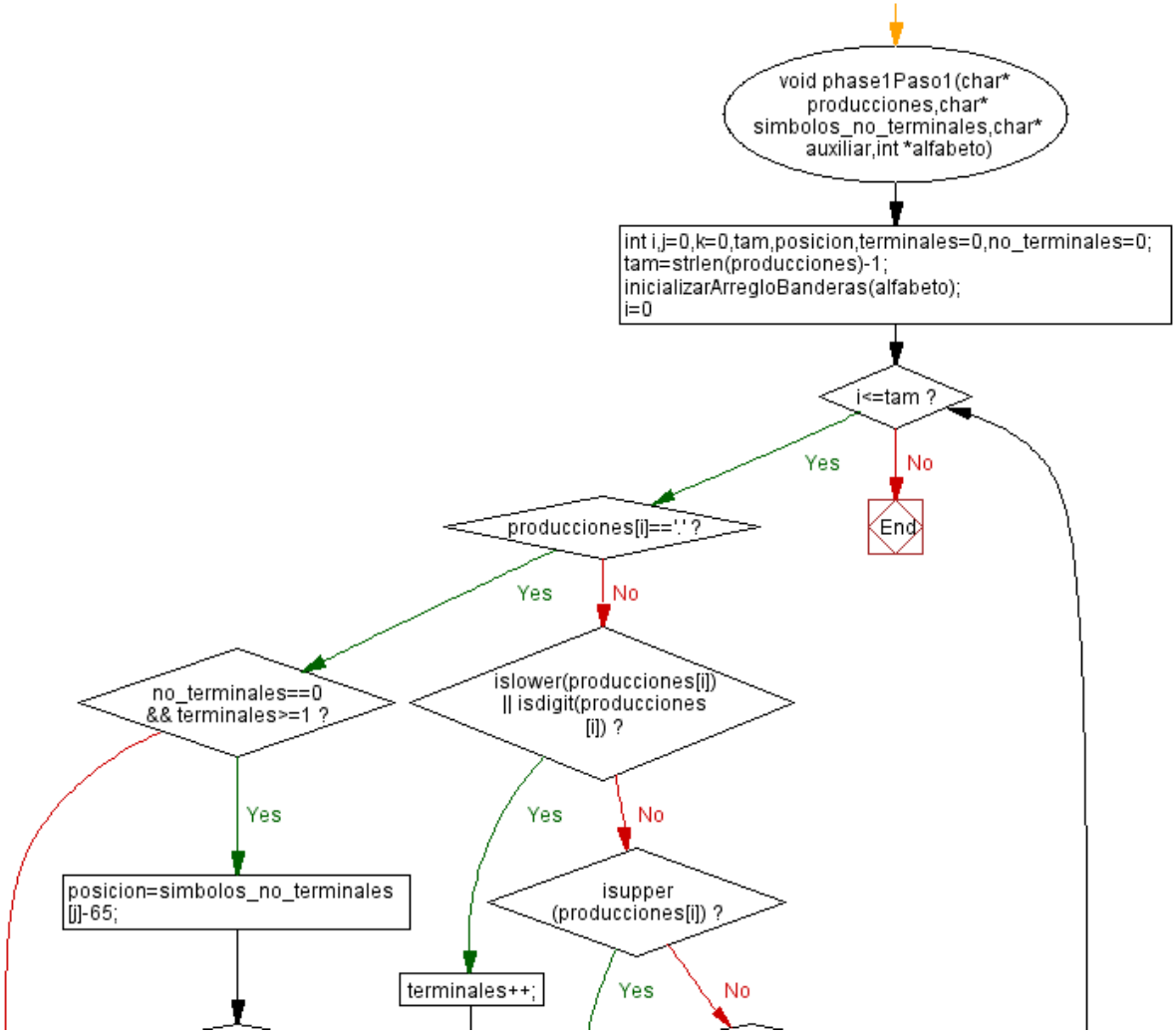
➔ void pedirProducciones

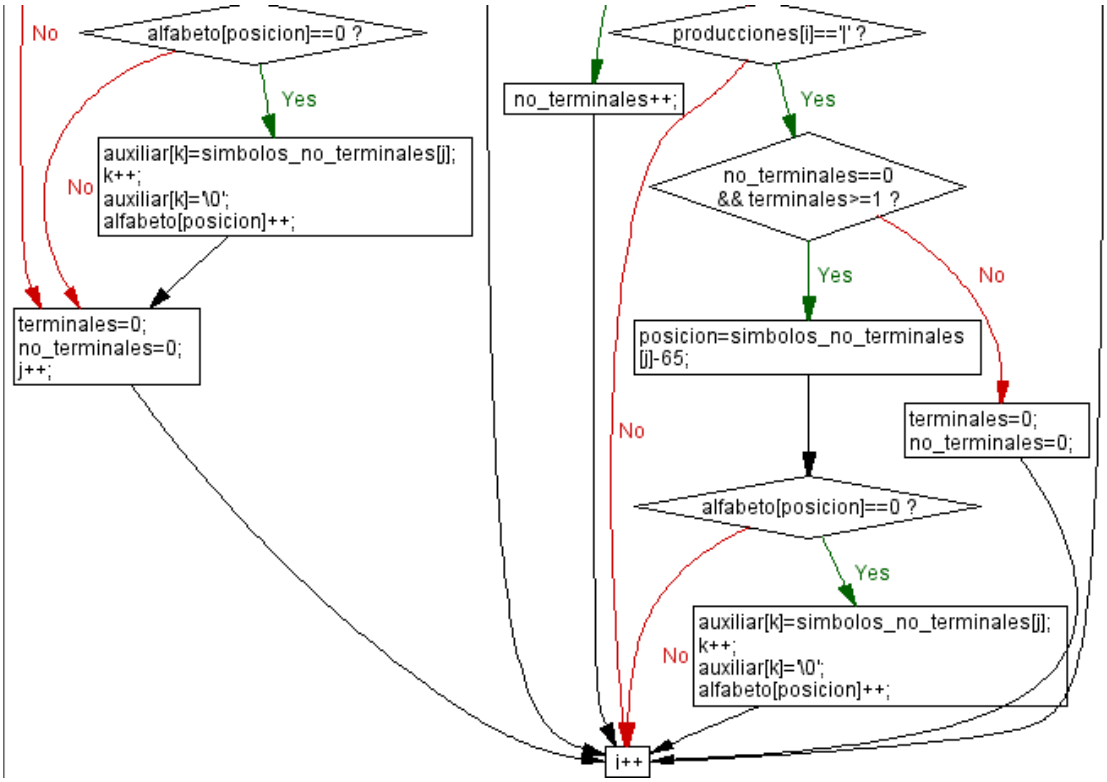


➔ void phase1

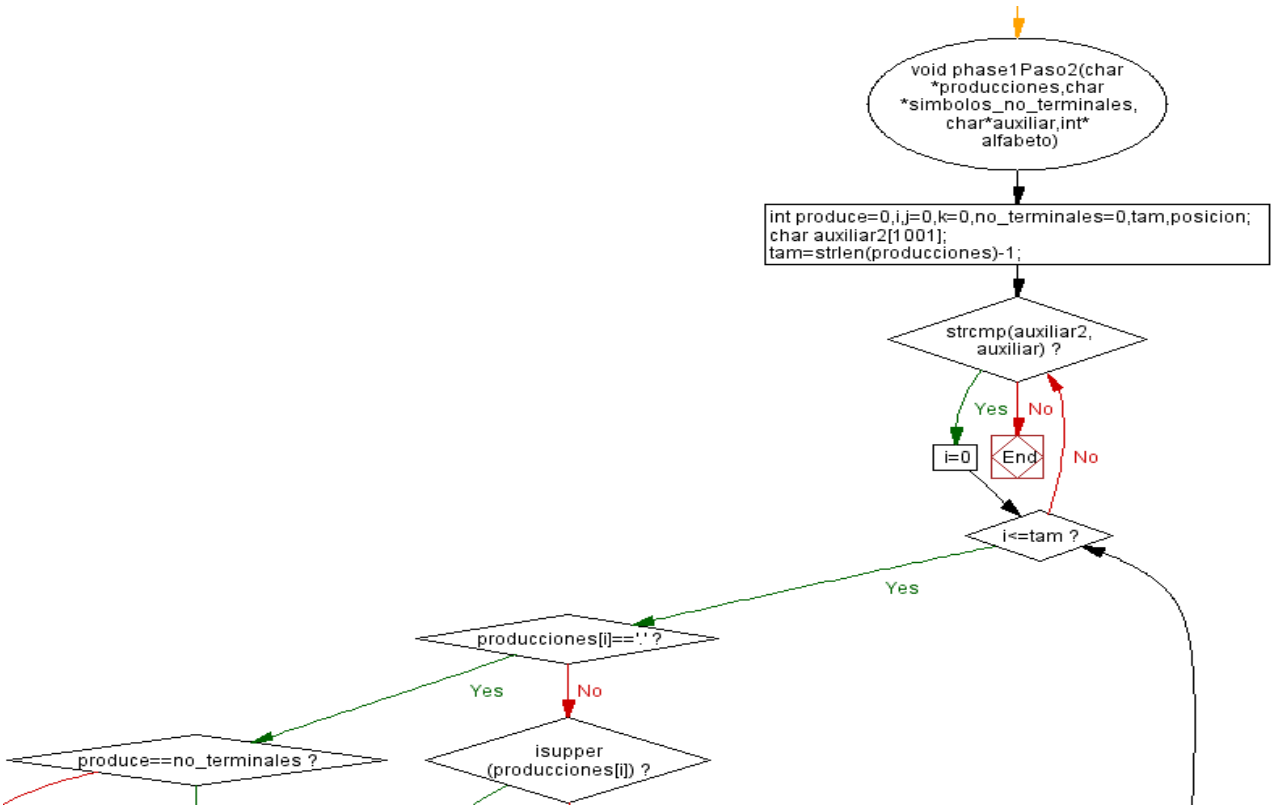


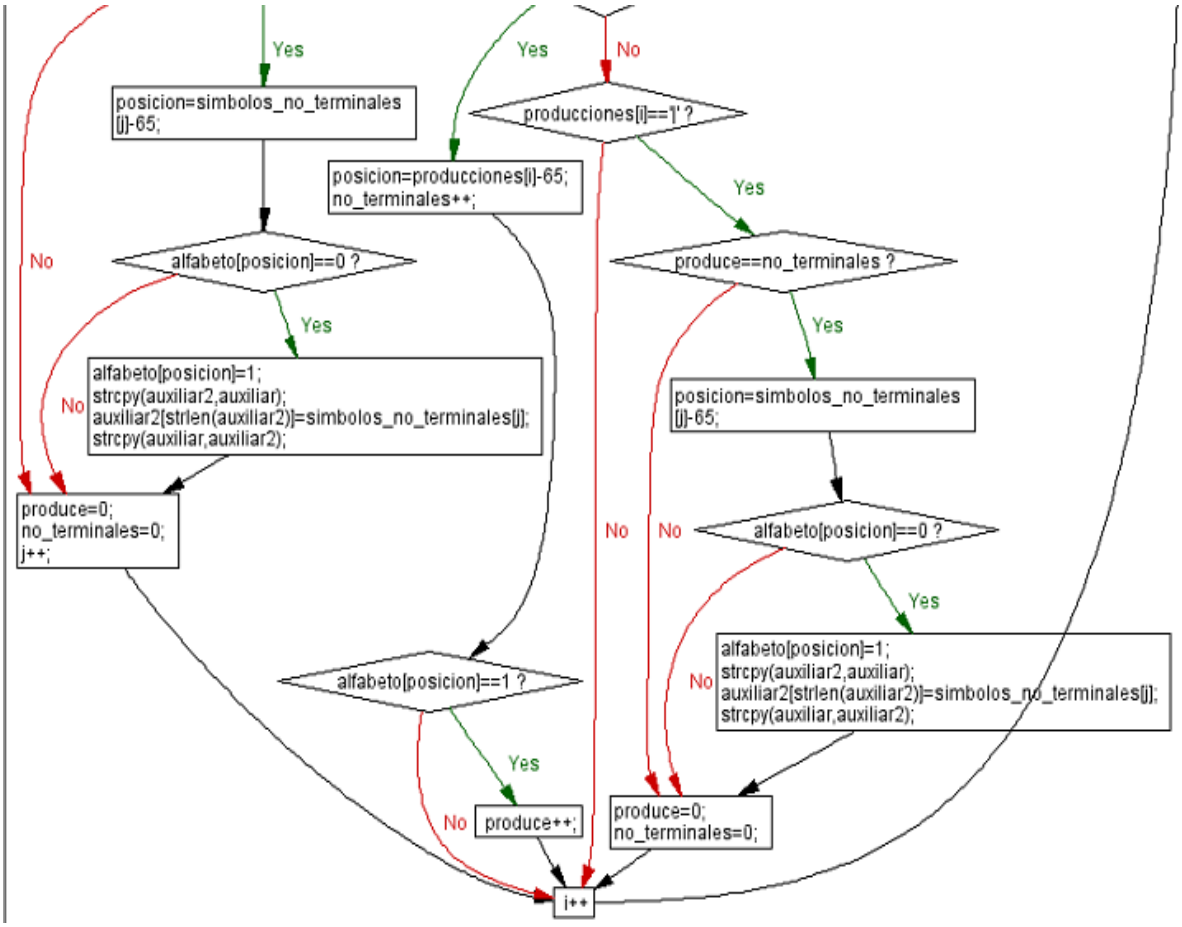
➔ void phase1Paso1



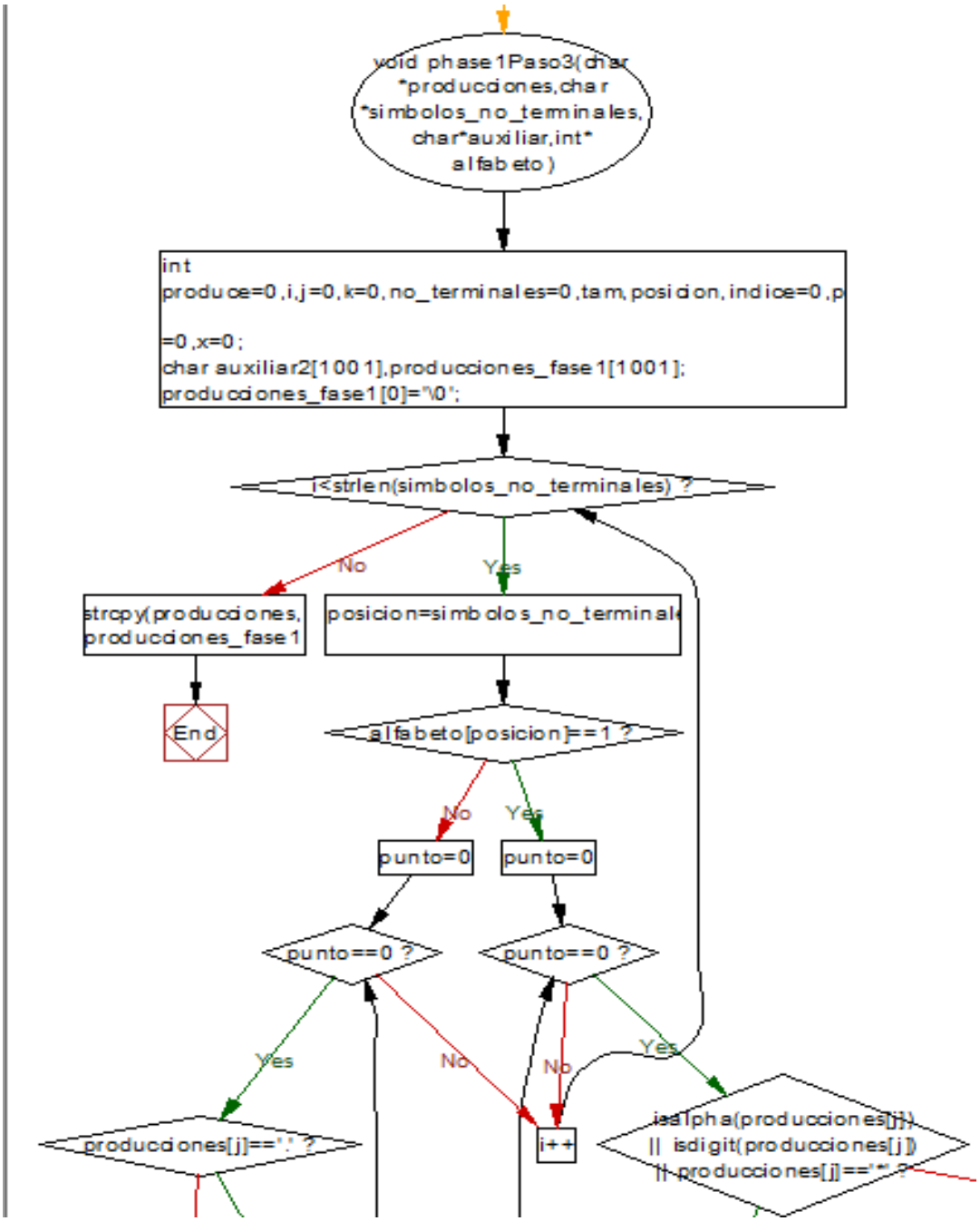


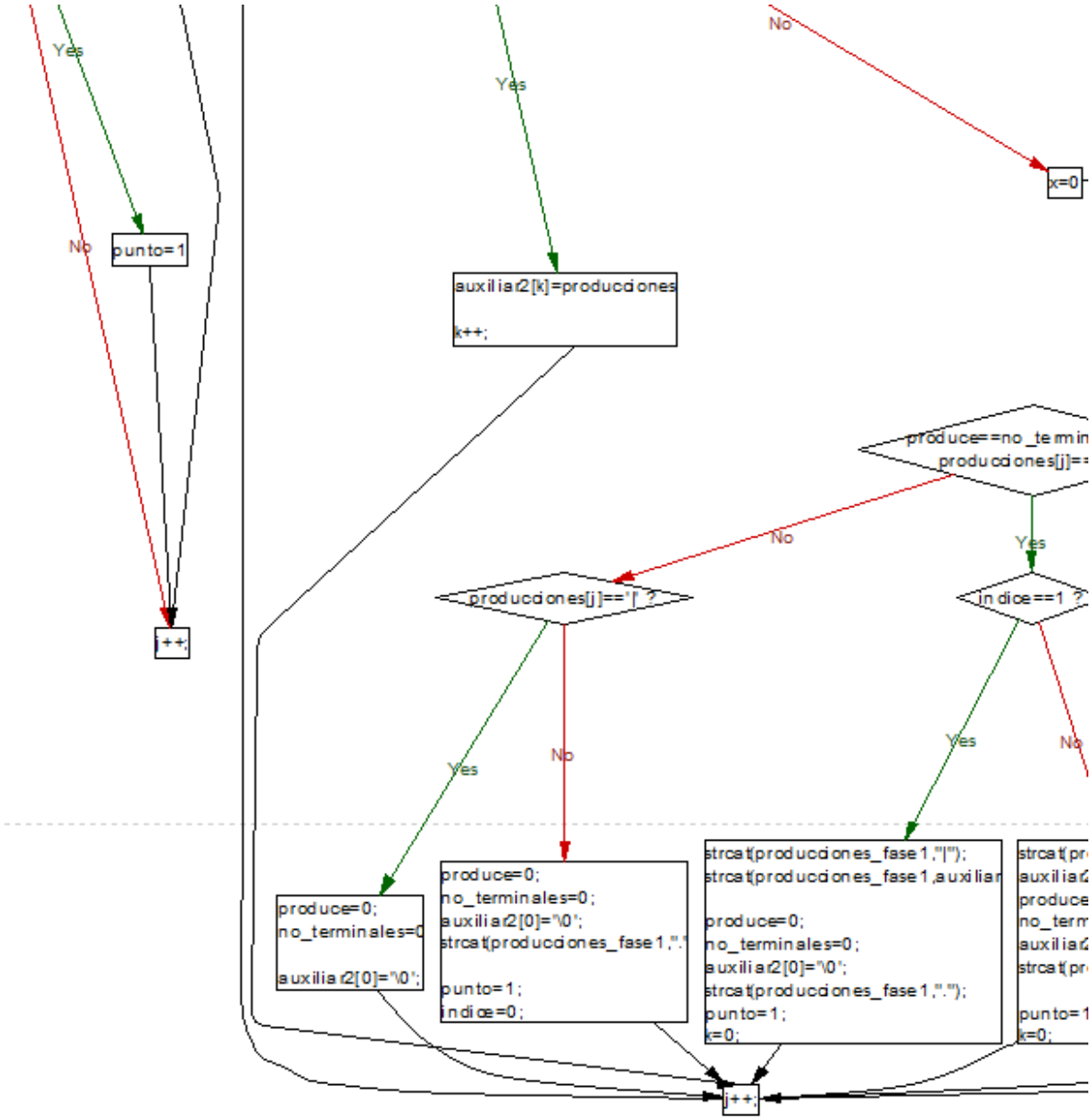
➔ void phase1Paso2

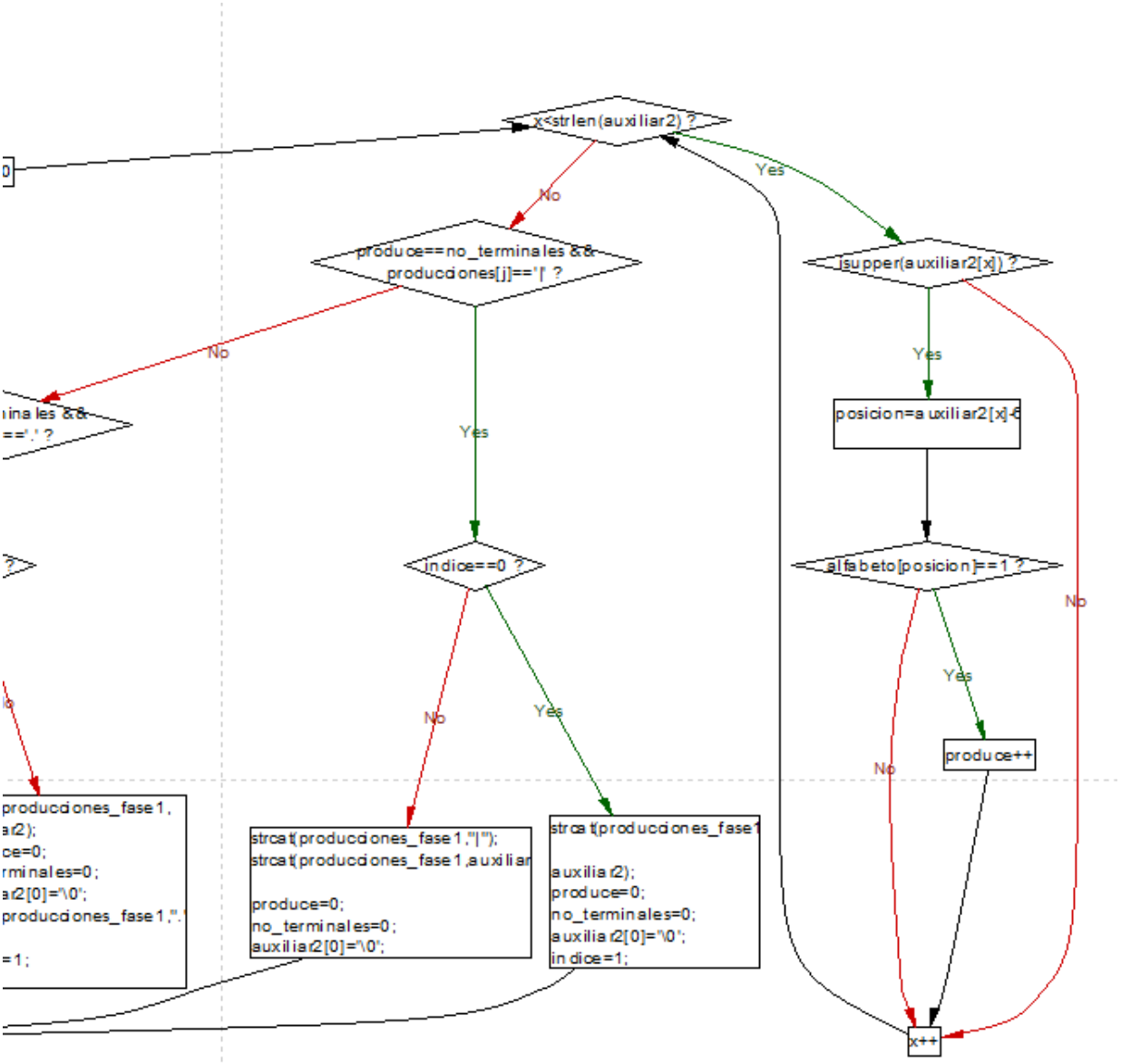



Yes
posicion=simbolos_no_terminales [j]-65;
alfabeto[posicion]==0 ?
No
Yes
alfabeto[posicion]=1;
strcpy(auxiliar2,auxiliar);
auxiliar2[strlen(auxiliar2)]=simbolos_no_terminales[j];
strcpy(auxiliar,auxiliar2);
No
produce=0;
no_terminales=0;
j++;
Yes
posicion=producciones[i]-65;
no_terminales++;
alfabeto[posicion]==1 ?
Yes
produce++;
No
Yes
No
producciones[i]=='|' ?
Yes
produce==no_terminales ?
No
Yes
posicion=simbolos_no_terminales [j]-65;
No
alfabeto[posicion]==0 ?
Yes
alfabeto[posicion]=1;
strcpy(auxiliar2,auxiliar);
auxiliar2[strlen(auxiliar2)]=simbolos_no_terminales[j];
strcpy(auxiliar,auxiliar2);
No
produce=0;
no_terminales=0;
i++

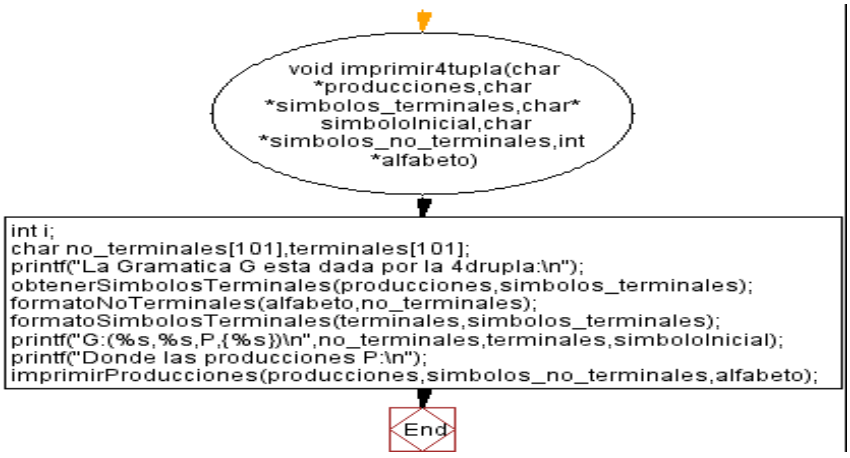
➔ void phase1Paso3



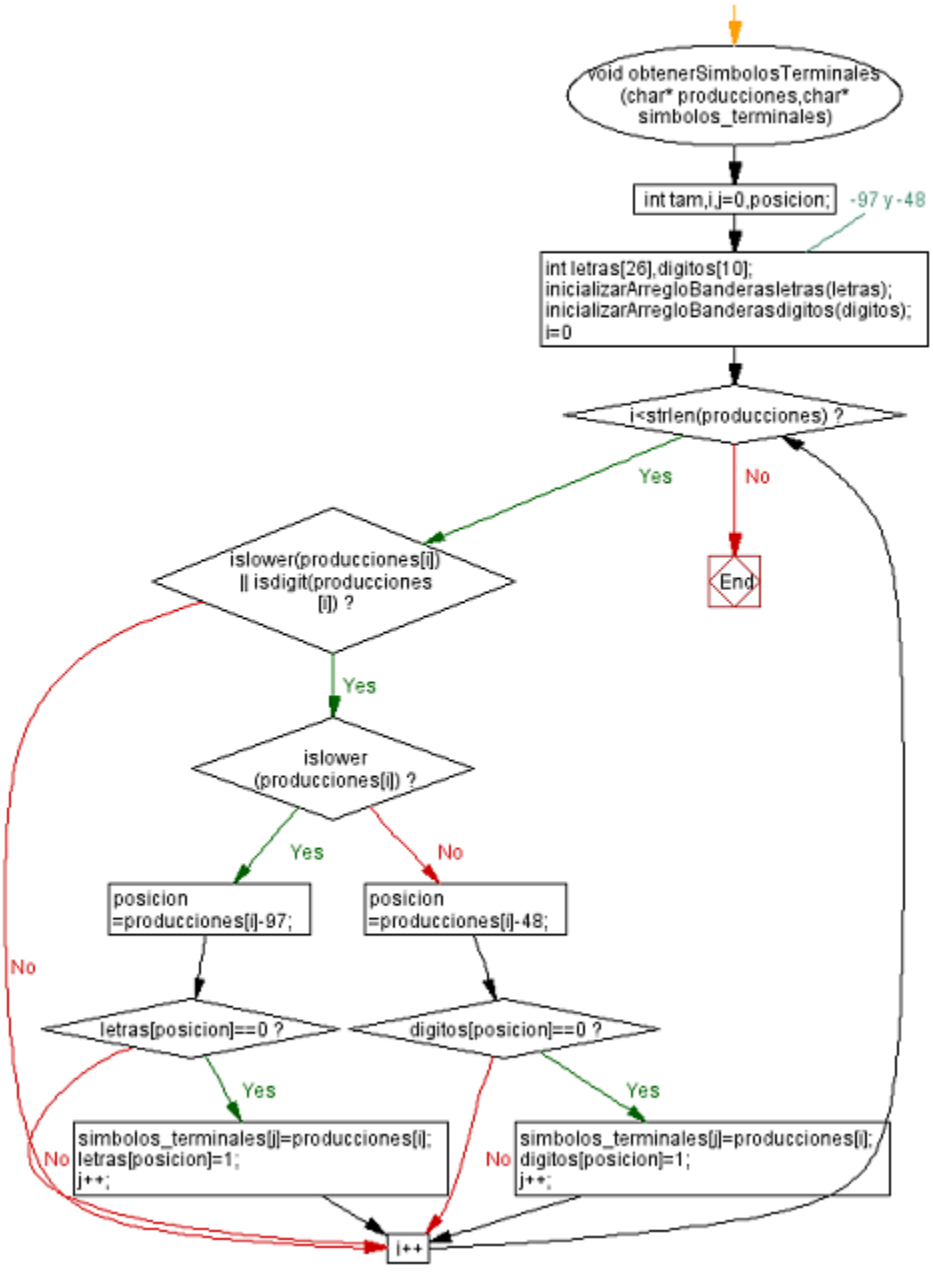

Yes
No
punto=1
j++;
Yes
auxiliar2[k]=producciones
k++;
No
x=0
produce==no_termin
producciones[j]==
No
Yes
producciones[j]=='|' ?
indice==1 ?
Yes
No
Yes
No
produce=0;
no_terminales=0;
auxiliar2[0]='\0';
produce=0;
no_terminales=0;
auxiliar2[0]='\0';
strcat(producciones_fase1,"
punto=1;
indice=0;
strcat(producciones_fase1,"|");
strcat(producciones_fase1,auxiliar
produce=0;
no_terminales=0;
auxiliar2[0]='\0';
strcat(producciones_fase1,".");
punto=1;
k=0;
j++;



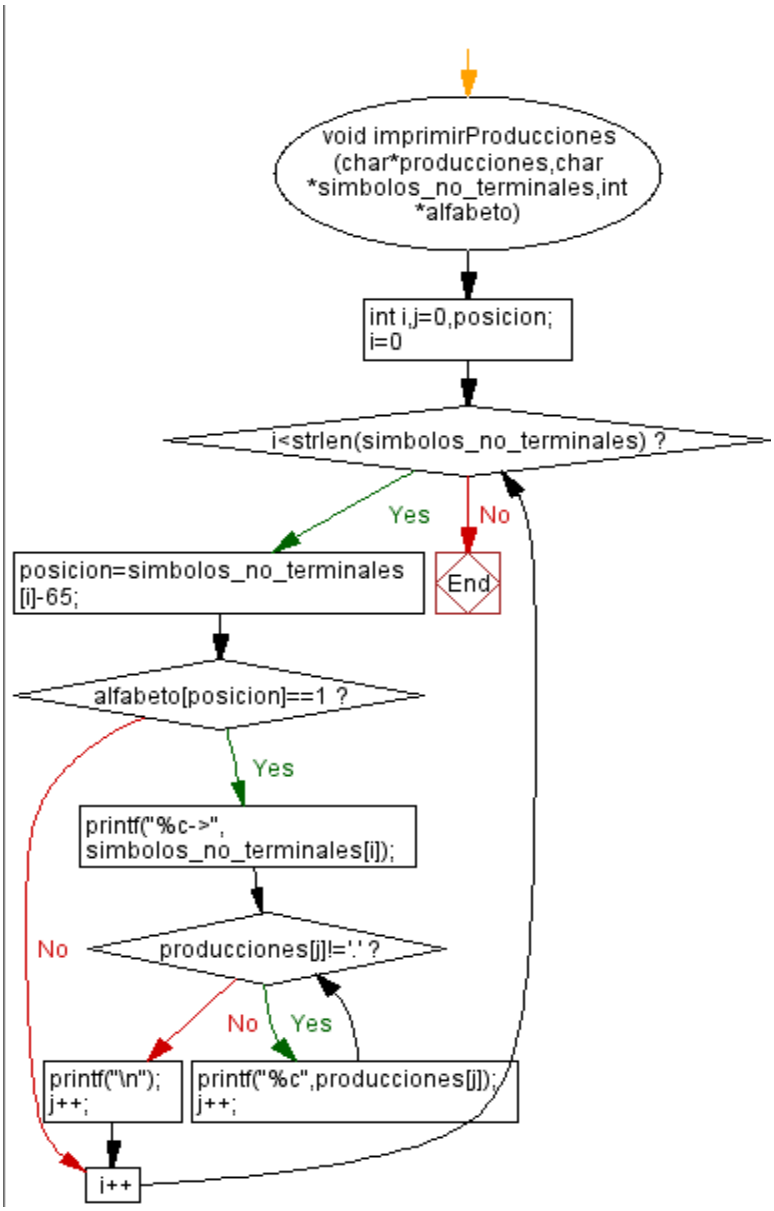
➔ void imprimir4tupla



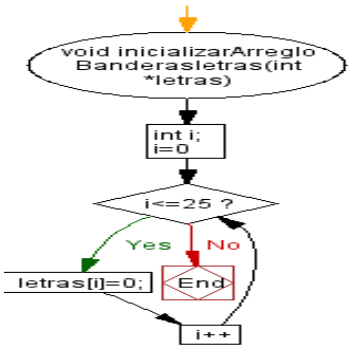
➔ void obtenerSimbolosTerminales



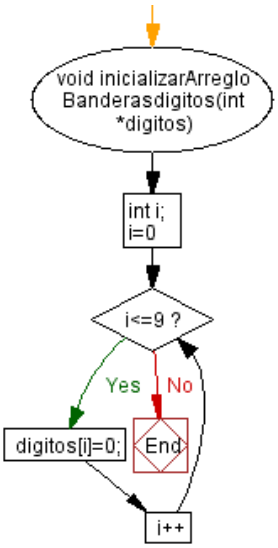
➔ void imprimirProducciones



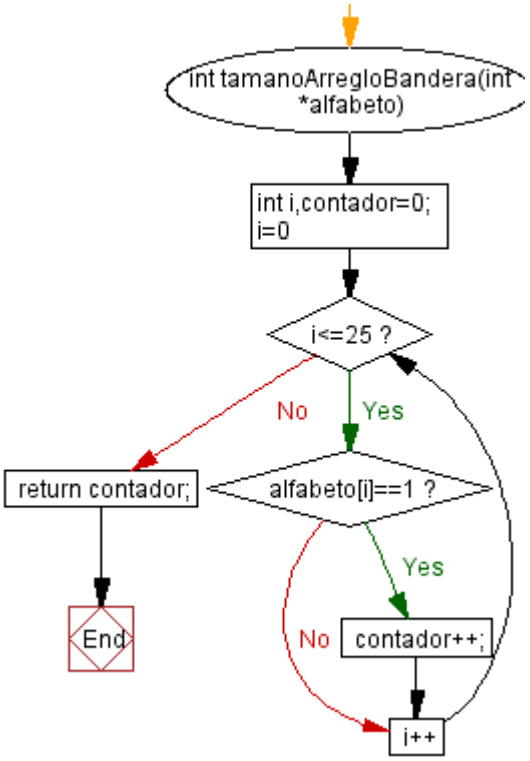
➔ void inicializarArregloBanderasletras



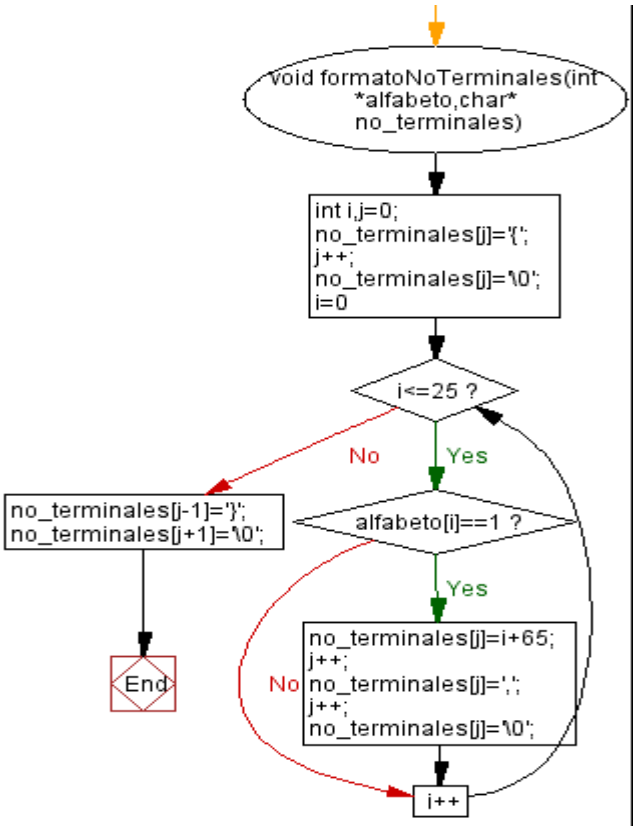
➔ void inicializarArregloBanderasdigitos



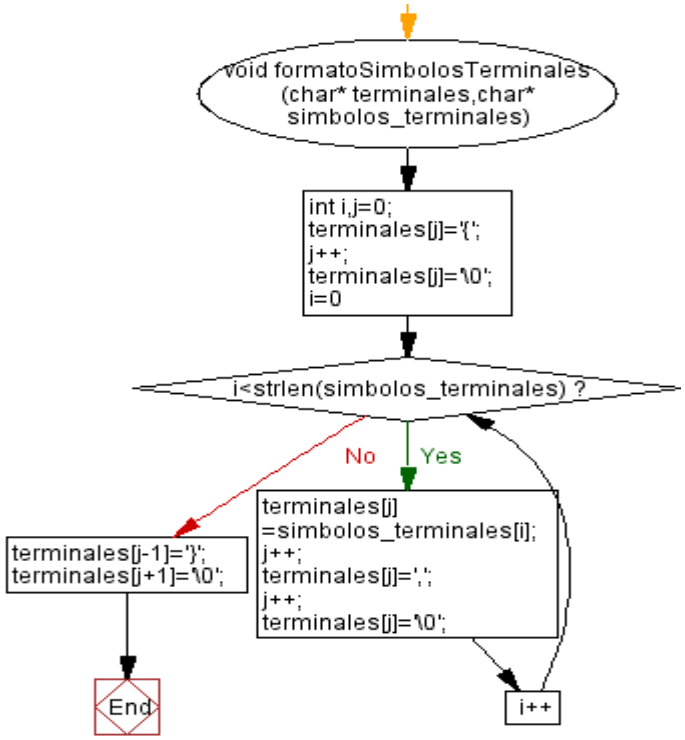
➔ int tamanoArregloBandera



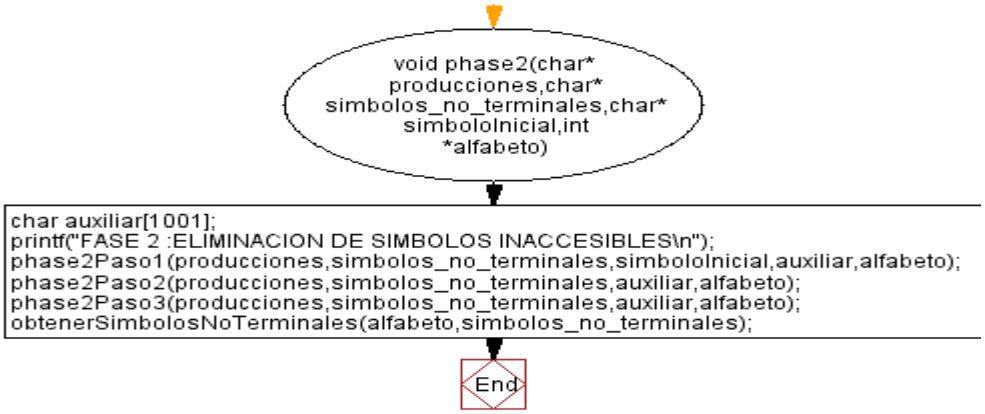
➔ void formatoNoTerminales



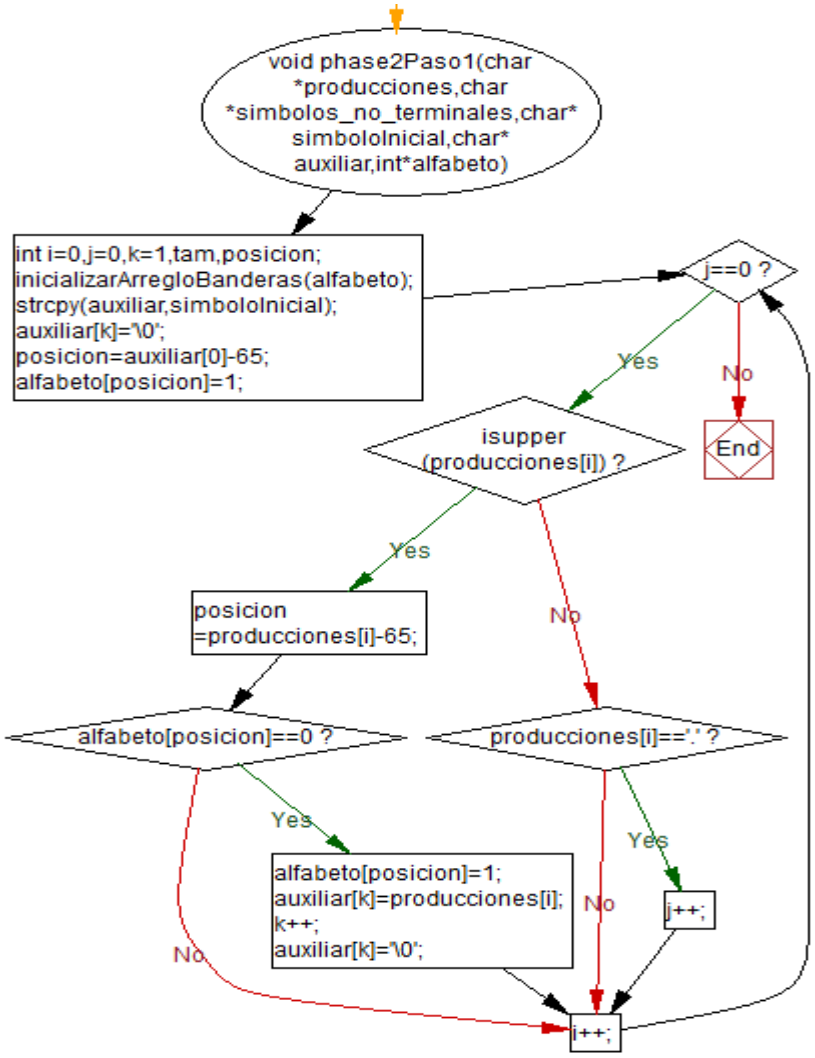
➔ void formatoSimbolosTerminales



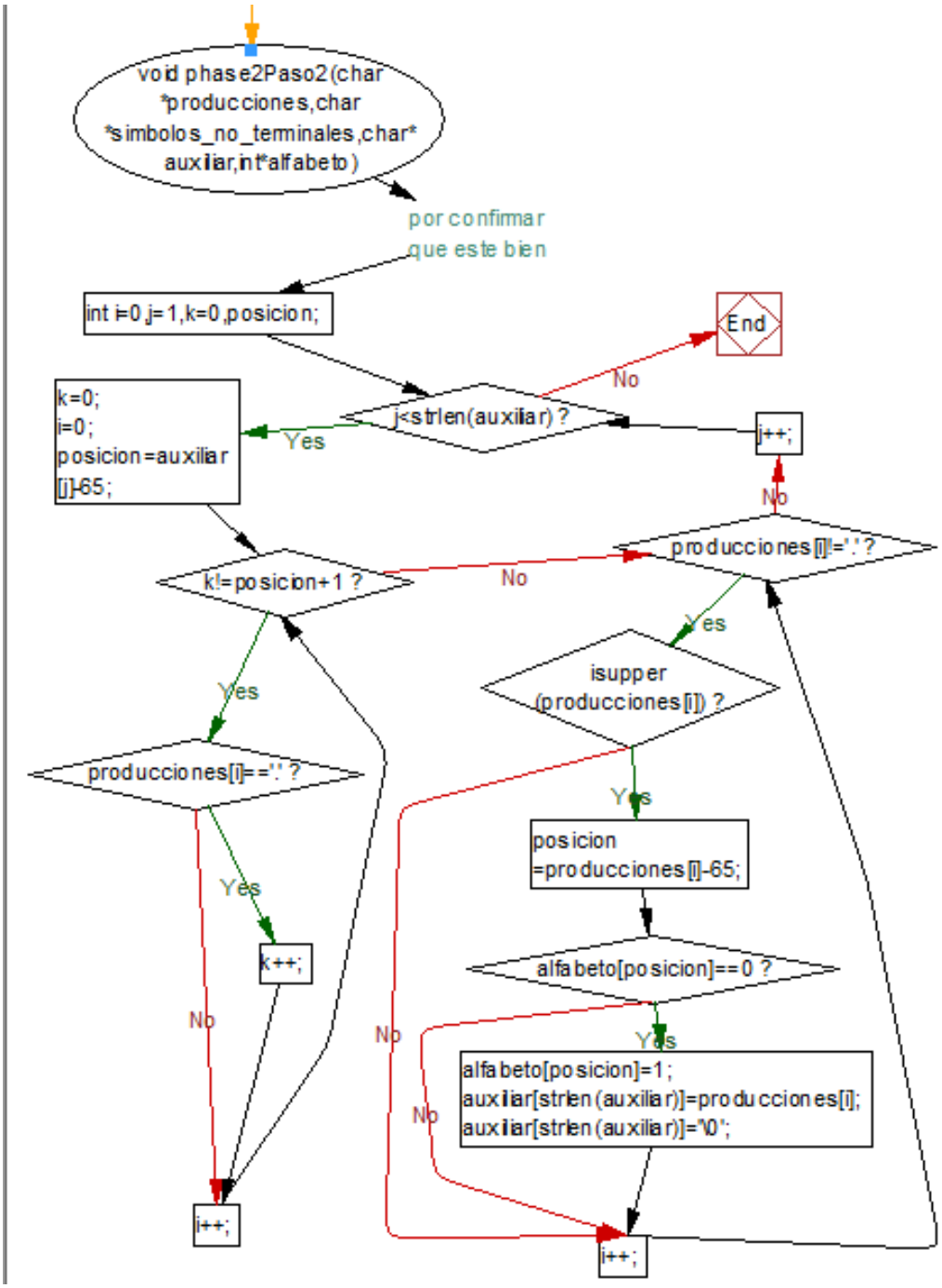
➔ void phase2



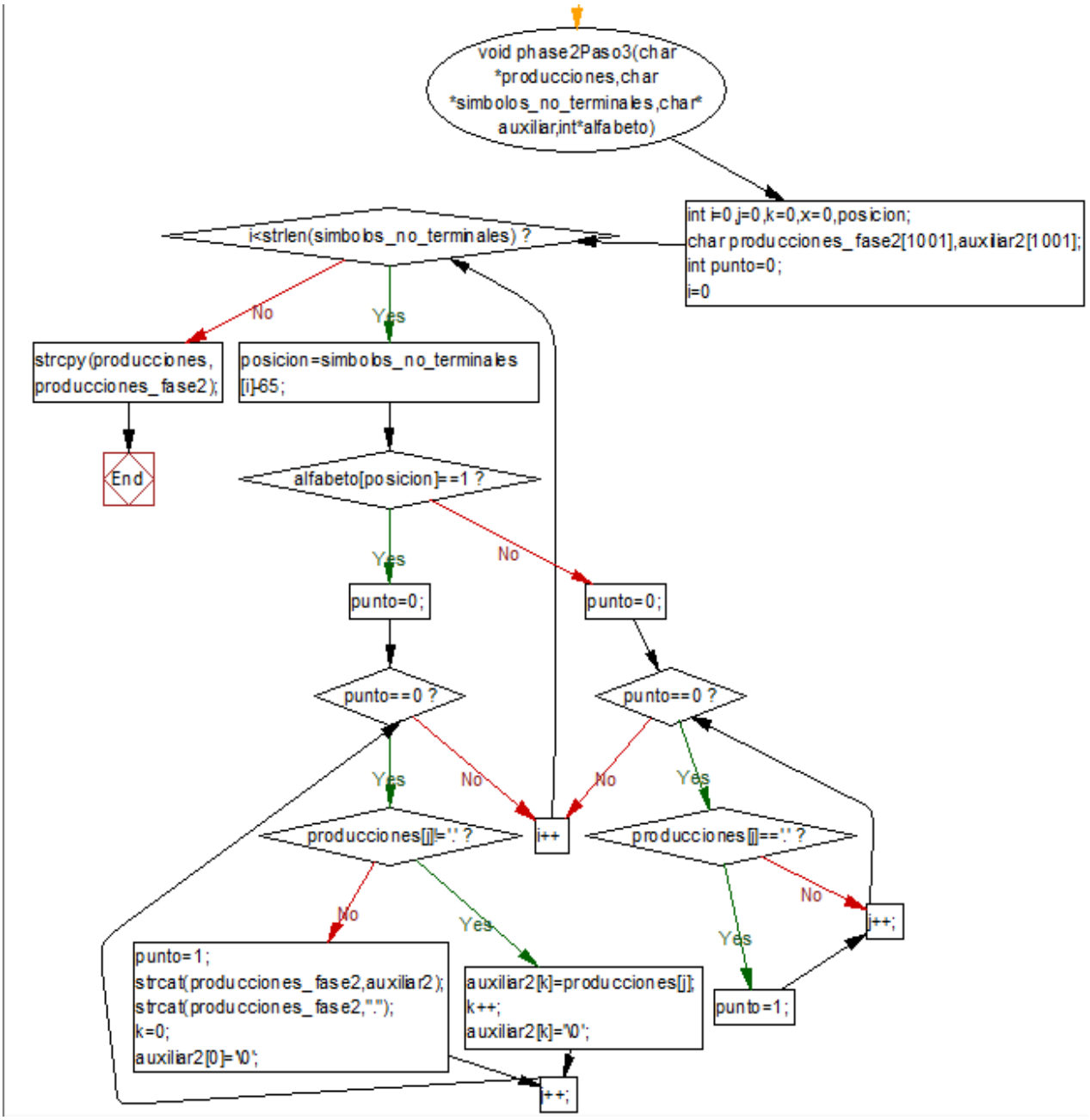
➔ void phase2Paso1



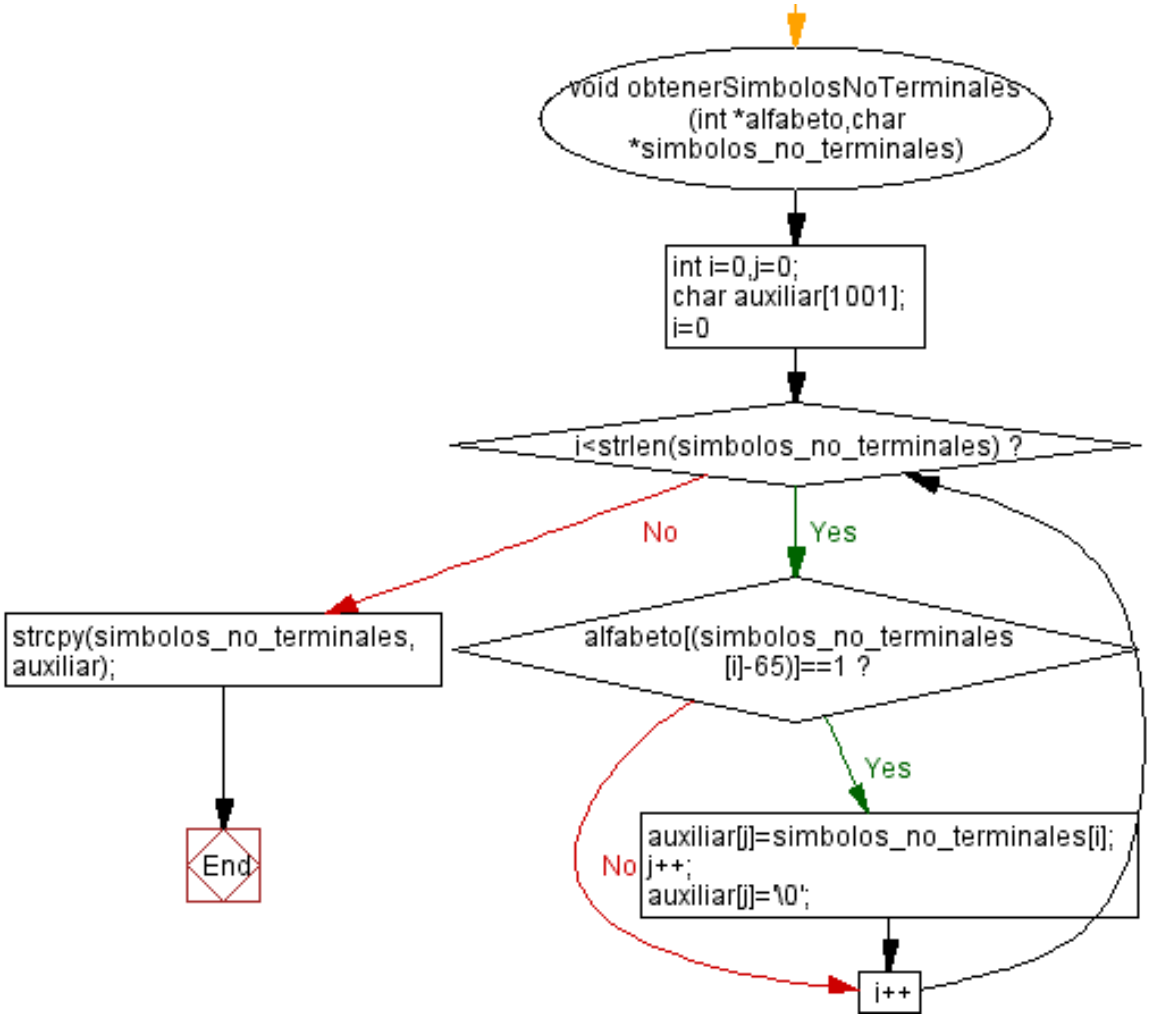
➔ void phase2Paso2



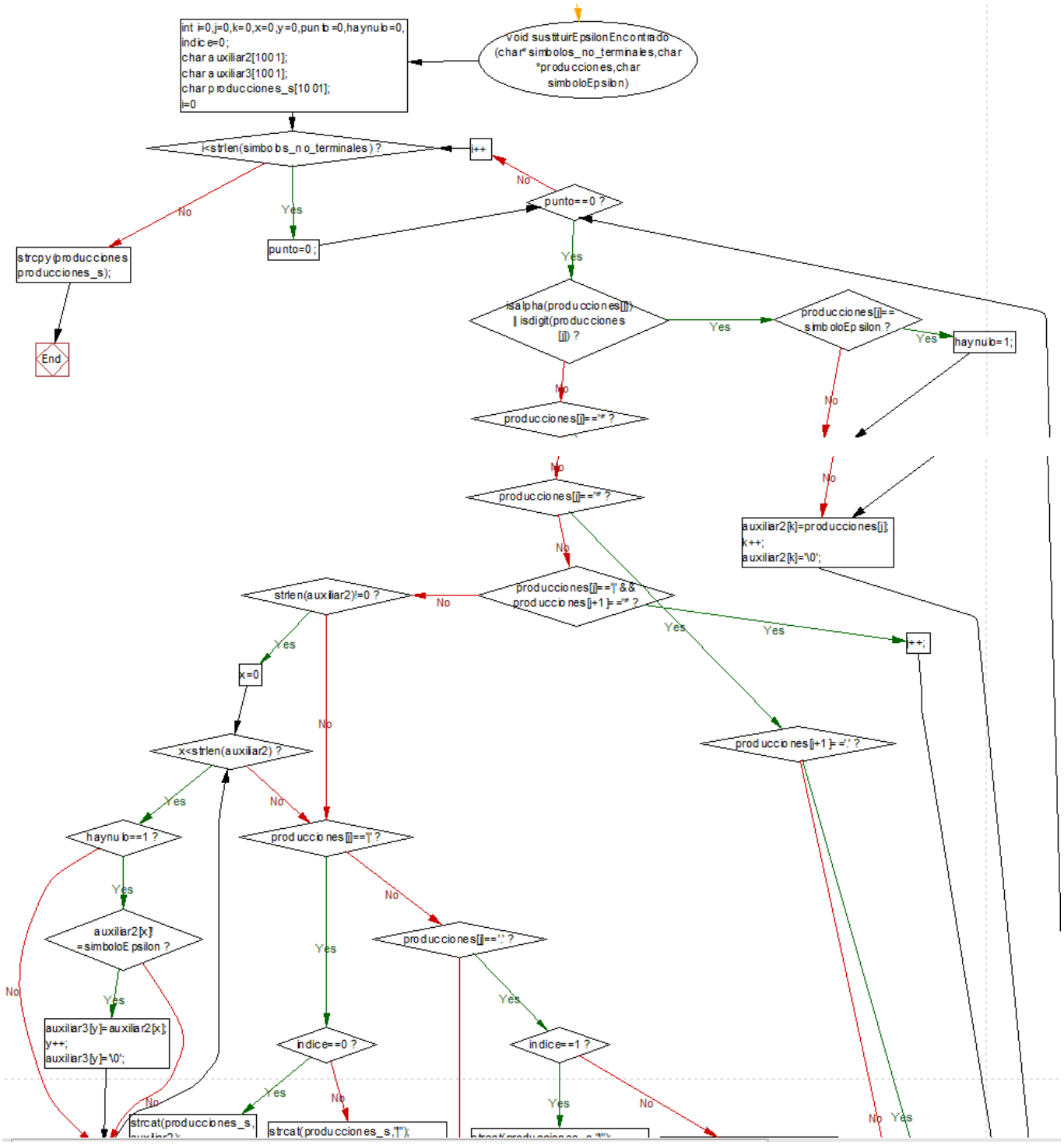
➔ void phase2Paso3

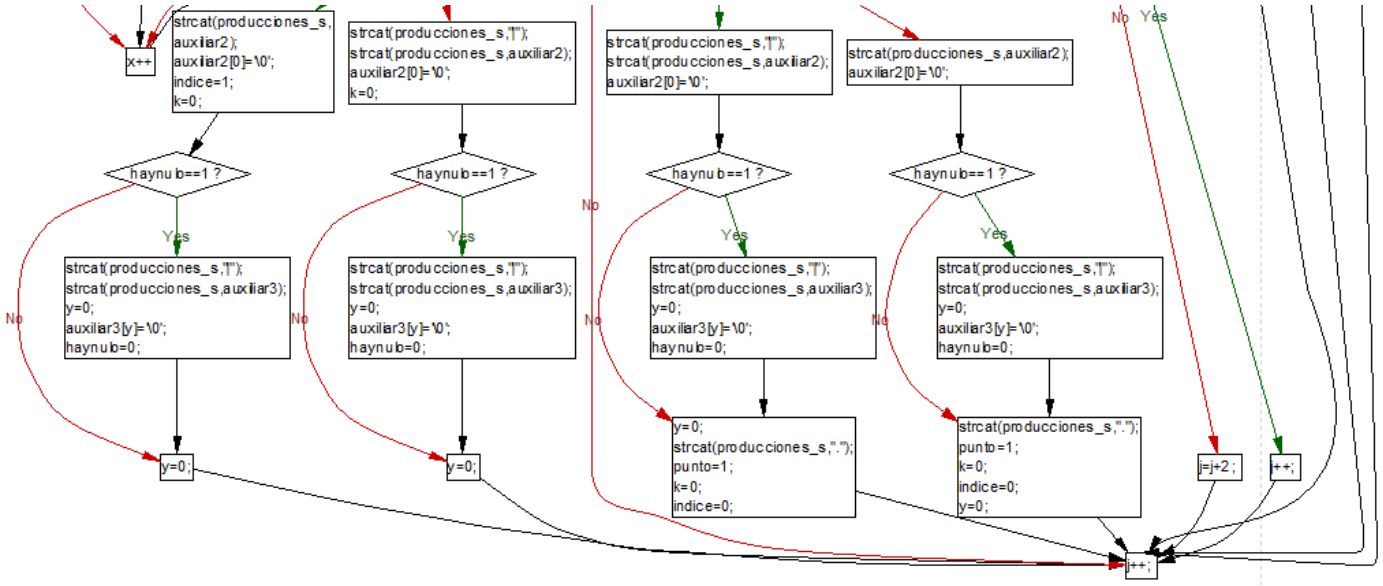


➔void obtenerSimbolosNoTerminales

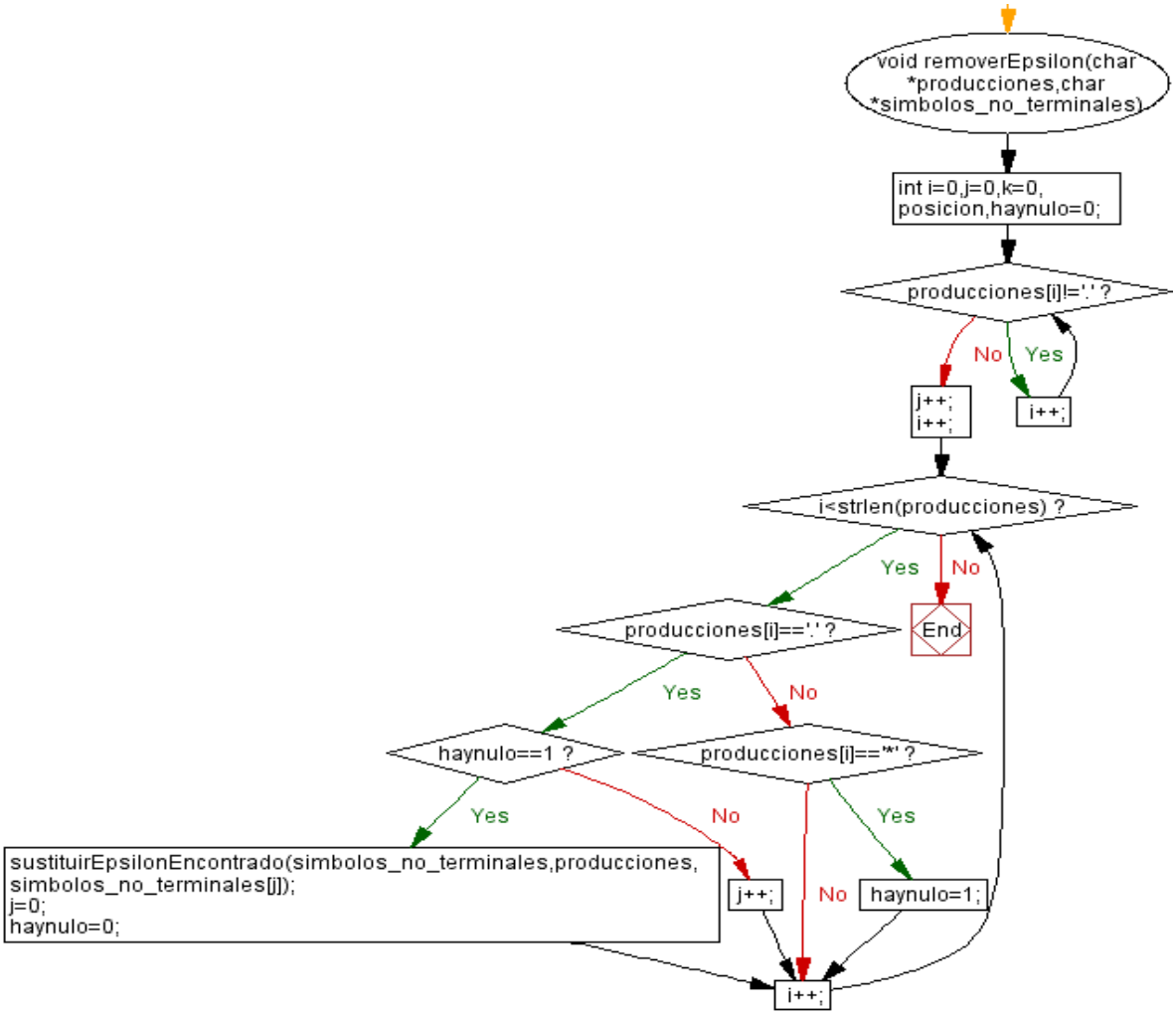


➔void sustituirEpsilonEncontrado(char* ,char *,char)

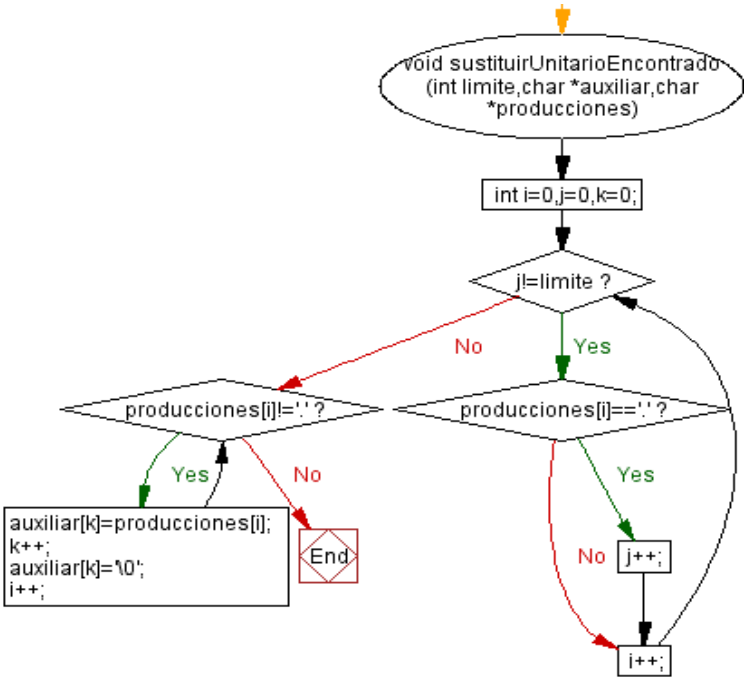




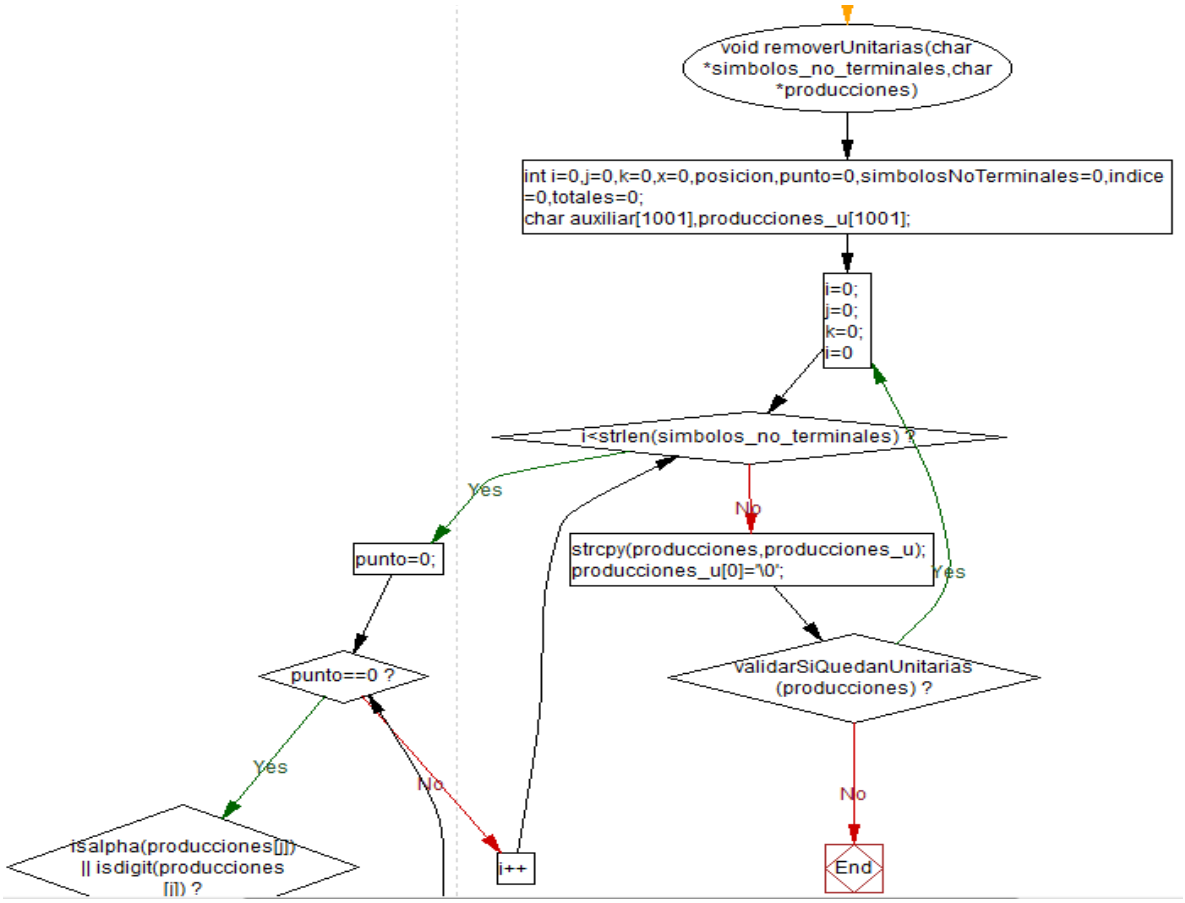
➔void removerEpsilon(char *,char *)

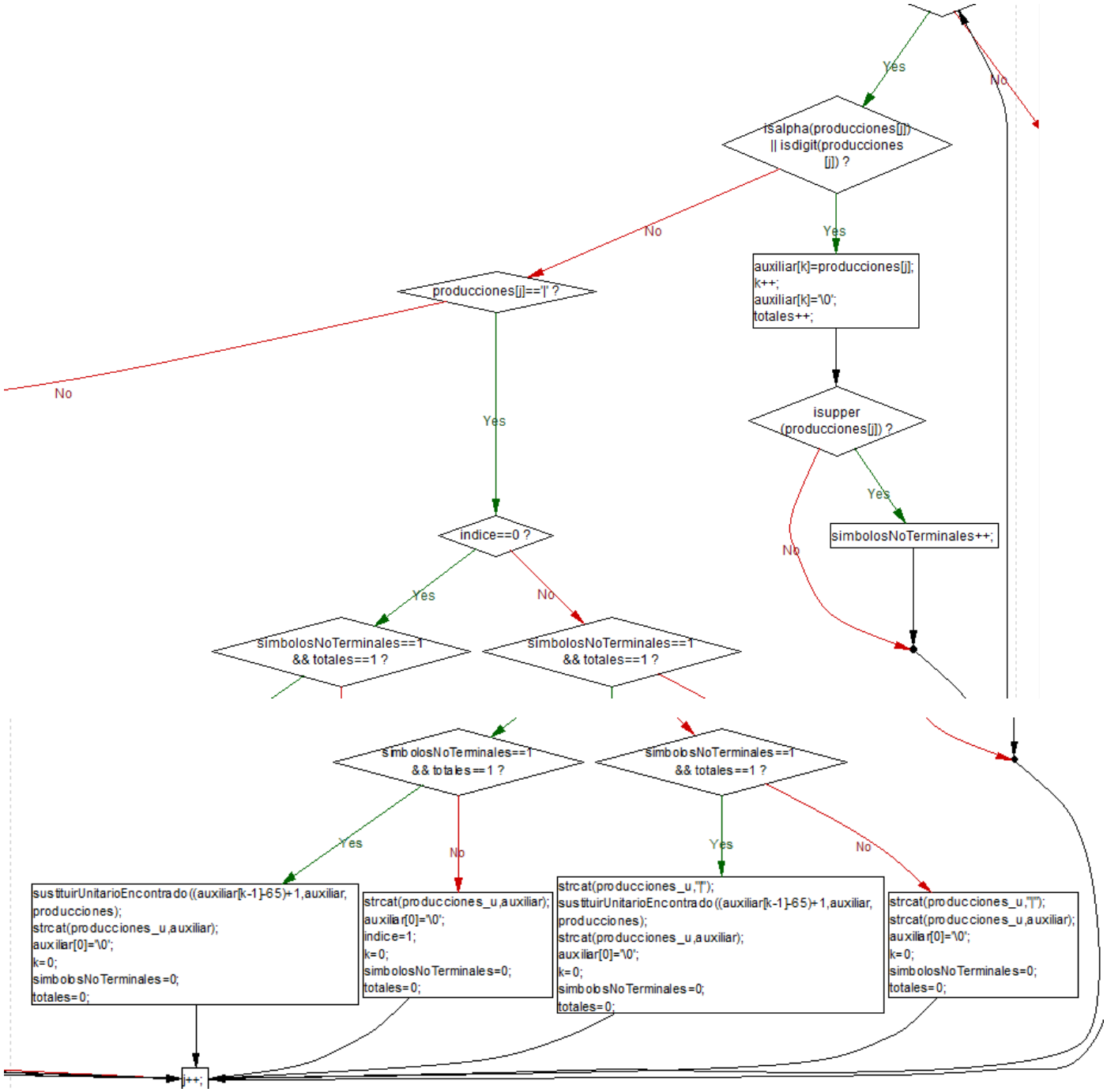


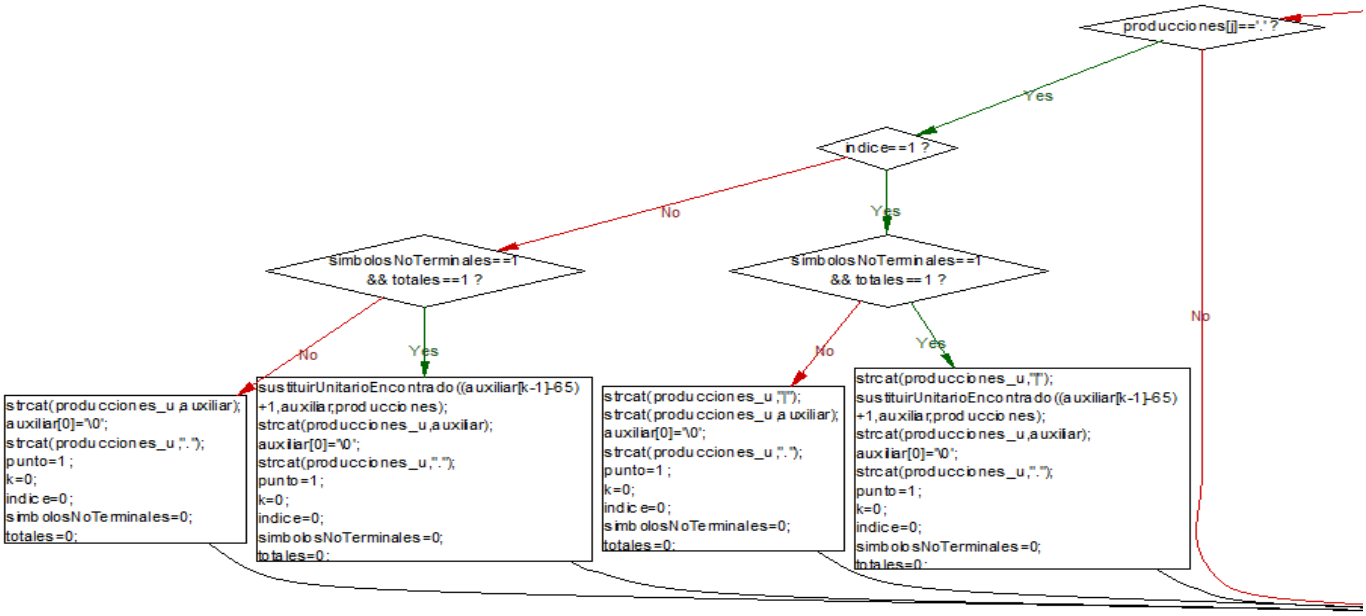
➔void sustituirUnitarioEncontrado(int ,char *,char*)



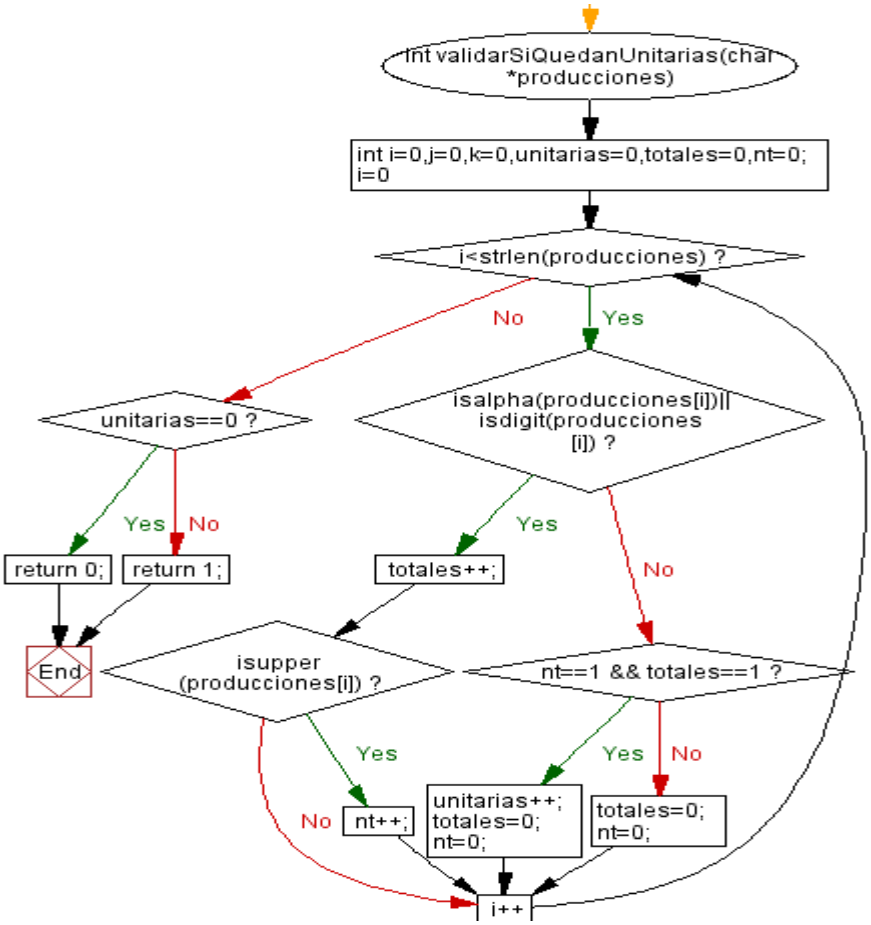
➔void removerUnitarias(char *,char *)




Yes
No
isalpha(producciones[j]) || isdigit(producciones [j]) ?
Yes
auxiliar[k]=producciones[j];
k++;
auxiliar[k]='\0';
totales++;
No
producciones[j]=='|' ?
No
Yes
isupper (producciones[j]) ?
Yes
simbolosNoTerminales++;
No
indice==0 ?
Yes
No
simbolosNoTerminales==1 && totales==1 ?
simbolosNoTerminales==1 && totales==1 ?
simbolosNoTerminales==1 && totales==1 ?
simbolosNoTerminales==1 && totales==1 ?
Yes
No
Yes
No
sustituirUnitarioEncontrado((auxiliar[k-1]-65)+1,auxiliar, producciones);
strcat(producciones_u,auxiliar);
auxiliar[0]='\0';
k=0;
simbolosNoTerminales=0;
totales=0;
strcat(producciones_u,auxiliar);
auxiliar[0]='\0';
indice=1;
k=0;
simbolosNoTerminales=0;
totales=0;
strcat(producciones_u,"|");
sustituirUnitarioEncontrado((auxiliar[k-1]-65)+1,auxiliar, producciones);
strcat(producciones_u,auxiliar);
auxiliar[0]='\0';
k=0;
simbolosNoTerminales=0;
totales=0;
strcat(producciones_u,"|");
strcat(producciones_u,auxiliar);
auxiliar[0]='\0';
k=0;
simbolosNoTerminales=0;
totales=0;
j++;



➔int validarSiQuedanUnitarias(char *)



Implementación de la solución

```
/*************************************************************************************
***********
PRÁCTICA #6
Versión 1.0 abril 2018

==>Autores:
    Hernández Escobedo Fernando
    Zanabria Ruis Luis David

Grupo 2CM4

Descripción: Programa capaz de limpiar GLC com 3 producciones (Maximo y minimo)

Compilación:
            gcc Practica6.c -o P6
Ejecución:
            P6.exe (En Windows) - ./6 (En Linux)
*************************************************************************************
**********/

//MACROS DEL PROCESADOR
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>

//DECLARACIÓN DE FUNCIONES O PROTOTIPOS DE FUNCIÓN
void inicializarArregloBanderas(int *);
void pedirSimbolosNoTerminales(char* ,int*);
void pedirProducciones(char*,char*);
void phase1(char* ,char* ,int *);
void phase1Paso1(char* ,char* ,char* ,int *);
void phase1Paso2(char* ,char* ,char* ,int *);
void phase1Paso3(char* ,char* ,char* ,int *);
void imprimir4tupla(char *,char *,char*,char*,int *);
void obtenerSimbolosTerminales(char* ,char*);
void imprimirProducciones(char*,char *,int *);
void inicializarArregloBanderasletras(int *);
void inicializarArregloBanderasdigitos(int*);
int tamanoArregloBandera(int *);
void formatoNoTerminales(int*,char*);
void formatoSimbolosTerminales(char* ,char*);
void phase2(char* ,char* ,char* ,int *);
void phase2Paso1(char *,char*,char* ,char* ,int*);
void phase2Paso2(char *,char*,char* ,int*);
void phase2Paso3(char *,char*,char* ,int*);
void obtenerSimbolosNoTerminales(int *,char *);
void sustituirEpsilonEncontrado(char* ,char *,char );
void removerEpsilon(char *,char *);
void sustituirUnitarioEncontrado(int ,char *,char*);
void removerUnitarias(char *,char *);
int validarSiQuedanUnitarias(char *);

//PROGRAMA PRINCIPAL
int main()
{
```

```
    char
simbolos_no_terminales[1001],producciones[7000],simbolos_terminales[1001];
    int alfabeto[26];
    char simboloInicial[2];
    inicializarArregloBanderas(alfabeto);
    pedirSimbolosNoTerminales(simbolos_no_terminales,alfabeto);
    simboloInicial[0]=simbolos_no_terminales[0];
    simboloInicial[1]='\0';
    pedirProducciones(producciones,simbolos_no_terminales);

    removerEpsilon(producciones,simbolos_no_terminales);
    removerUnitarias(simbolos_no_terminales,producciones);
    phase1(producciones,simbolos_no_terminales,alfabeto);
    printf("RESULTADOS DE LA FASE 1:\n");

imprimir4tupla(producciones,simbolos_terminales,simboloInicial,simbolos_no_termin
ales,alfabeto);
    phase2(producciones,simbolos_no_terminales,simboloInicial,alfabeto);
    printf("RESULTADOS DE LA FASE 2:\n");

imprimir4tupla(producciones,simbolos_terminales,simboloInicial,simbolos_no_termin
ales,alfabeto);
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void inicializarArregloBanderas(int *alfabeto)
Descripción:Inicializa un arreglo con 0's que tiene utilidad de bandera
Recibe: Referencia al arreglo de enteros "alfabeto"
Devuelve:NADA
Observaciones:
*/
void inicializarArregloBanderas(int *alfabeto)
{
    int i;
    for(i=0;i<=25;i++)
    {
        alfabeto[i]=0;
    }
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void pedirSimbolosNoTerminales(char* simbolos_no_terminales,int* alfabeto)
Descripción:Pide los simbolos no terminales que tendran producciones
Recibe: Referencia al arreglo de enteros "alfabeto" y referencia a la cadena
"simbolos_no_terminales".
Devuelve: NADA
Observaciones:
*/
void pedirSimbolosNoTerminales(char* simbolos_no_terminales,int* alfabeto)
{
    int tam,i,j=0,contiguedad=0,posicion;
    char auxiliar[1001];
    printf("Digita los simbolos no terminales separados por comas
(S0,S1...Sn):\n");
    scanf("%s",auxiliar);
    strcat(auxiliar,",");//agrega una , al final del arreglo,esto lo hice por que
más adelante en el for de abajo me quise ahorrar una condicional
    tam=strlen(auxiliar)-1;
    //Mete los simbolos no terminales en un arreglo,siempre y cuando sean solo
mayusculas solas
    for(i=0;i<=tam;i++)
```

```
    {
        if(isalpha(auxiliar[i]))
            contiguedad++;
        else if(auxiliar[i]==',')
        {
            if(isupper(auxiliar[i-1]) && contiguedad==1)
            {
                posicion=auxiliar[i-1]-65;
            if(alfabeto[posicion]==0)
            {
                simbolos_no_terminales[j]=auxiliar[i-1];
                j++;
                simbolos_no_terminales[j]='\0';
                alfabeto[posicion]=1;
            }
            }
            contiguedad=0;
        }
    }
    printf("Simbolos No terminales:%s\n",simbolos_no_terminales);
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void pedirProducciones(char* producciones,char* simbolos_no_terminales)
Descripción:Pide las producciones de los simbolos no terminales pedidos
anteriormente
Recibe: Referencia a la cadena "producciones" y referencia a la cadena
"simbolos_no_terminales".
Devuelve: NADA
Observaciones:
*/
void pedirProducciones(char* producciones,char* simbolos_no_terminales)
{
    int tam,i;
    char auxiliar[1001];
    tam=strlen(simbolos_no_terminales)-1;
    printf("Digite las producciones de los respectivos simbolos con simbolos
terminales(letras minusculas,* para representar epsilon) y no terminales (i.e S-
>a|b|aS...):\n");
    for(i=0;i<=tam;i++)
    {
        printf("%c->",simbolos_no_terminales[i]);
        scanf("%s",auxiliar);
        strcat(producciones,auxiliar);
        strcat(producciones,".");
        fflush(stdin);
    }
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void phase1(char* producciones,char* simbolos_no_terminales,int *alfabeto)
Descripción:En la fase 1 se eliminan simbolos muertos por medio de subpasos
Recibe: Referencia a la cadena "producciones", referencia a la cadena
"simbolos_no_terminales" y referencia al arreglo de enteros "alfabeto".
Devuelve: NADA
Observaciones:
*/
void phase1(char* producciones,char* simbolos_no_terminales,int *alfabeto)
{
    char auxiliar[1001];
    printf("FASE 1 :ELIMINACION DE SIMBOLOS NO GENERATIVOS\n");
```

```
    phase1Paso1(producciones,simbolos_no_terminales,auxiliar,alfabeto);

    phase1Paso2(producciones,simbolos_no_terminales,auxiliar,alfabeto);

    phase1Paso3(producciones,simbolos_no_terminales,auxiliar,alfabeto);

    obtenerSimbolosNoTerminales(alfabeto,simbolos_no_terminales);
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void phase1Paso1(char* producciones,char* simbolos_no_terminales,char*
auxiliar,int *alfabeto)
Descripción:Identifica que simbolos  producen al menos un simbolo terminal
Recibe: Referencia a la cadena "producciones", referencia a la cadena
"simbolos_no_terminales", referencia a la cadena auxiliar y referencia al arreglo
de enteros "alfabeto".
Devuelve: NADA
Observaciones:
*/
void phase1Paso1(char* producciones,char* simbolos_no_terminales,char*
auxiliar,int *alfabeto)
{
    int i,j=0,k=0,tam,posicion,terminales=0,no_terminales=0;
    tam=strlen(producciones)-1;
    inicializarArregloBanderas(alfabeto);
    for(i=0;i<=tam;i++)
    {
        if(producciones[i]=='.')
        {
            if(no_terminales==0 && terminales>=1)
            {
                posicion=simbolos_no_terminales[j]-65;
                if(alfabeto[posicion]==0)
                {
                    auxiliar[k]=simbolos_no_terminales[j];
                    k++;
                    auxiliar[k]='\0';
                    alfabeto[posicion]++;
                }
            }
            terminales=0;
            no_terminales=0;
            j++;
        }
        else if(islower(producciones[i]) || isdigit(producciones[i]))
            terminales++;
        else if(isupper(producciones[i]))
        {
            no_terminales++;
        }
        else if(producciones[i]=='|')
        {
            if(no_terminales==0 && terminales>=1)
            {
                posicion=simbolos_no_terminales[j]-65;
                if(alfabeto[posicion]==0)
                {
                    auxiliar[k]=simbolos_no_terminales[j];
                    k++;
                    auxiliar[k]='\0';
                    alfabeto[posicion]++;
```

```
                }
            }
            else
            {
                terminales=0;
                no_terminales=0;
            }
        }
    }
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void phase1Paso2(char*producciones,char*simbolos_no_terminales,char*auxiliar,int*
alfabeto)
Descripción:Verifica que simbolos producen los simbolos que se guardaron en el
paso anterior
Recibe: Referencia a la cadena "producciones", referencia a la cadena
"simbolos_no_terminales", referencia a la cadena auxiliar y referencia al arreglo
de enteros "alfabeto".
Devuelve: NADA
Observaciones:
*/
void phase1Paso2(char*producciones,char*simbolos_no_terminales,char*auxiliar,int*
alfabeto)
{
    int produce=0,i,j=0,k=0,no_terminales=0,tam,posicion;
    char auxiliar2[1001];
    tam=strlen(producciones)-1;
    while(strcmp(auxiliar2,auxiliar))
    {
        for(i=0;i<=tam;i++)
        {
            if(producciones[i]=='.')
            {
                if(produce==no_terminales )
                {
                    posicion=simbolos_no_terminales[j]-65;
                    if(alfabeto[posicion]==0)
                    {
                        alfabeto[posicion]=1;
                        strcpy(auxiliar2,auxiliar);
                        auxiliar2[strlen(auxiliar2)]=simbolos_no_terminales[j];
                        strcpy(auxiliar,auxiliar2);
                        }
                }
                produce=0;
                no_terminales=0;
                j++;
            }
            else if(isupper(producciones[i]))
            {
                posicion=producciones[i]-65;
                no_terminales++;
                if(alfabeto[posicion]==1)
                    produce++;
            }
            else if(producciones[i]=='|')
            {
                if(produce==no_terminales)
                {
                    posicion=simbolos_no_terminales[j]-65;
                    if(alfabeto[posicion]==0)
```

```
                    {
                        alfabeto[posicion]=1;
                        strcpy(auxiliar2,auxiliar);
                        auxiliar2[strlen(auxiliar2)]=simbolos_no_terminales[j];
                        strcpy(auxiliar,auxiliar2);
                    }
                }
                produce=0;
                no_terminales=0;
            }
        }
    }
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void phase1Paso3(char*producciones,char*simbolos_no_terminales,char*auxiliar,int*
alfabeto)
Descripción:actualiza los nuevos simbolos no terminales
Recibe: Referencia a la cadena "producciones", referencia a la cadena
"simbolos_no_terminales", referencia a la cadena auxiliar y referencia al arreglo
de enteros "alfabeto".
Devuelve: NADA
Observaciones:
*/
void phase1Paso3(char*producciones,char*simbolos_no_terminales,char*auxiliar,int*
alfabeto)
{
    int produce=0,i,j=0,k=0,no_terminales=0,tam,posicion,indice=0,punto=0,x=0;
    char auxiliar2[1001],producciones_fase1[1001];
    producciones_fase1[0]='\0';
    tam=strlen(producciones)-1;

    for(i=0;i<strlen(simbolos_no_terminales);i++)
    {
        posicion=simbolos_no_terminales[i]-65;
        if(alfabeto[posicion]==1)
        {
            punto=0;
            while(punto==0)
            {
                if(isalpha(producciones[j]) || isdigit(producciones[j]) ||
producciones[j]=='*')
                {
                    auxiliar2[k]=producciones[j];
                    k++;
                    auxiliar2[k]='\0';
                }
                else
                {
                    for(x=0;x<strlen(auxiliar2);x++)
                    {
                        if(isupper(auxiliar2[x]))
                        {
                            posicion=auxiliar2[x]-65;
                            no_terminales++;
                            if(alfabeto[posicion]==1)
                                produce++;
                        }
                    }
                    if(produce==no_terminales && producciones[j]=='|')
                    {
                        if(indice==0)
```

```
                    {
                        strcat(producciones_fase1,auxiliar2);
                        produce=0;
                        no_terminales=0;
                        auxiliar2[0]='\0';
                        indice=1;
                        k=0;
                    }
                    else
                    {
                        strcat(producciones_fase1,"|");
                        strcat(producciones_fase1,auxiliar2);
                        produce=0;
                        no_terminales=0;
                        auxiliar2[0]='\0';
                        k=0;
                    }
                }
                else if(produce==no_terminales && producciones[j]=='.')
                {
                    if(indice==1)
                    {
                        strcat(producciones_fase1,"|");
                        strcat(producciones_fase1,auxiliar2);
                        produce=0;
                        no_terminales=0;
                        auxiliar2[0]='\0';
                        strcat(producciones_fase1,".");
                        punto=1;
                        k=0;
                        indice=0;
                    }
                    else
                    {
                        strcat(producciones_fase1,auxiliar2);
                        produce=0;
                        no_terminales=0;
                        auxiliar2[0]='\0';
                        strcat(producciones_fase1,".");
                        punto=1;
                        k=0;
                        indice=0;
                    }
                }
                else
                {
                    if(producciones[j]=='|')
                    {
                        produce=0;
                        no_terminales=0;
                        auxiliar2[0]='\0';
                        k=0;
                    }
                    else
                    {
                        produce=0;
                        no_terminales=0;
                        auxiliar2[0]='\0';
                        strcat(producciones_fase1,".");
                        punto=1;
                        indice=0;
                        k=0;
                    }
```

```
                    }
                }
                j++;
            }
        }
        else
        {
            punto=0;
            while(punto==0)
            {
                if(producciones[j]=='.')
                {
                    punto=1;
                }
                j++;
            }
        }
    }
    strcpy(producciones,producciones_fase1);
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void imprimir4tupla(char *producciones,char *simbolos_terminales,char*
simboloInicial,char*simbolos_no_terminales,int *alfabeto)
Descripción:imprime la 4drupla con sus respectivas descripciones
Recibe: Referencia a la cadena "producciones", referencia a la cadena
"simbolos_terminales", referencia a la cadena "simboloInicial",
referencia a la cadena "simbolos_no_terminales" y referencia al arreglo de
enteros "alfabeto".
Devuelve: NADA
Observaciones:
*/
void imprimir4tupla(char *producciones,char *simbolos_terminales,char*
simboloInicial,char*simbolos_no_terminales,int *alfabeto)
{
    int i;
    char no_terminales[101],terminales[101];
    printf("La Gramatica G esta dada por la 4drupla:\n");
    obtenerSimbolosTerminales(producciones,simbolos_terminales);
    formatoNoTerminales(alfabeto,no_terminales);
    formatoSimbolosTerminales(terminales,simbolos_terminales);
    printf("G:(%s,%s,P,{%s})\n",no_terminales,terminales,simboloInicial);
    printf("Donde las producciones P:\n");
    imprimirProducciones(producciones,simbolos_no_terminales,alfabeto);
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void obtenerSimbolosTerminales(char* producciones,char* simbolos_terminales)
Descripción:obtiene los simbolos terminales de una produccion
Recibe: Referencia a la cadena "producciones" y referencia a la cadena
"simbolos_terminales".
Devuelve: NADA
Observaciones:
*/
void obtenerSimbolosTerminales(char* producciones,char* simbolos_terminales)
{
    int tam,i,j=0,posicion;
    int letras[26],digitos[10];//-97 y  -48
    inicializarArregloBanderasletras(letras);
    inicializarArregloBanderasdigitos(digitos);
    for(i=0;i<strlen(producciones);i++)
```

```
    {
        if(islower(producciones[i]) || isdigit(producciones[i]))
        {
            if(islower(producciones[i]))
            {
                posicion=producciones[i]-97;
                if(letras[posicion]==0)
                {
                    simbolos_terminales[j]=producciones[i];
                    letras[posicion]=1;
                    j++;
                }
            }
            else
            {
                posicion=producciones[i]-48;
                if(digitos[posicion]==0)
                {
                    simbolos_terminales[j]=producciones[i];
                    digitos[posicion]=1;
                    j++;
                }
            }

        }
    }
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void imprimirProducciones(char*producciones,char *simbolos_no_terminales,int
*alfabeto)
Descripción:imprime las producciones de los simbolos no terminales
Recibe: Referencia a la cadena "producciones", referencia a la cadena
"simbolos_no_terminales" y referencia al arreglo de enteros "alfabeto".
Devuelve: NADA
Observaciones:
*/
void imprimirProducciones(char*producciones,char *simbolos_no_terminales,int
*alfabeto)
{
    int i,j=0,posicion;
    for(i=0;i<strlen(simbolos_no_terminales);i++)
        {
            posicion=simbolos_no_terminales[i]-65;
            if(alfabeto[posicion]==1)
            {
                printf("%c->",simbolos_no_terminales[i]);
                while(producciones[j]!='.')
                {
                    printf("%c",producciones[j]);
                    j++;
                }
                printf("\n");
                j++;
            }
        }
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void inicializarArregloBanderasletras(int *letras)
Descripción:inicializa un arreglo de banderas para letras minusculas con 0´s
```

```
Recibe: Referencia al arreglo de enteros "letras".
Devuelve: NADA
Observaciones:
*/
void inicializarArregloBanderasletras(int *letras)
{
    int i;
    for(i=0;i<=25;i++)
    {
        letras[i]=0;
    }
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void inicializarArregloBanderasdigitos(int *digitos)
Descripción:inicializa un arreglo de banderas para digitos con 0's
Recibe: Referencia al arreglo de enteros "digitos".
Devuelve: NADA
Observaciones:
*/
void inicializarArregloBanderasdigitos(int *digitos)
{
    int i;
    for(i=0;i<=9;i++)
    {
        digitos[i]=0;
    }
};

/*
DEFINICIÓN DE PROCEDIMIENTO
int tamanoArregloBandera(int *alfabeto)
Descripción:muestra el "tamano" del arreglo de banderas,esto es,cuando sus
posiciones son diferentes de 0
Recibe: Referencia al arreglo de enteros "alfabeto".
Devuelve:
Observaciones:
*/
int tamanoArregloBandera(int *alfabeto)
{
    int i,contador=0;
    for(i=0;i<=25;i++)
    {
        if(alfabeto[i]==1)
        {
            contador++;
        }
    }
    return contador;
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void formatoNoTerminales(int *alfabeto,char* no_terminales)
Descripción:Da formato para impresión de simbolos no terminales de la gramatica
Recibe: Referencia al arreglo de enteros "alfabeto" y a la cadena "no_terminales"
Devuelve: NADA
Observaciones:
*/
void formatoNoTerminales(int *alfabeto,char* no_terminales)
{
    int i,j=0;
```

```
    no_terminales[j]='{';
    j++;
    no_terminales[j]='\0';
    for(i=0;i<=25;i++)
    {
        if(alfabeto[i]==1)
        {
            no_terminales[j]=i+65;
            j++;
            no_terminales[j]=',';
            j++;
            no_terminales[j]='\0';
        }
    }
    no_terminales[j-1]='}';
    no_terminales[j+1]='\0';
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void formatoSimbolosTerminales(char* terminales,char* simbolos_terminales)
Descripción:Da formato para imprimir los simbolos terminales de la gramatica
Recibe: Referencia a la cadena "terminales" y a la cadena "simbolos_terminales".
Devuelve: NADA
Observaciones:
*/
void formatoSimbolosTerminales(char* terminales,char* simbolos_terminales)
{
    int i,j=0;
    terminales[j]='{';
    j++;
    terminales[j]='\0';
    for(i=0;i<strlen(simbolos_terminales);i++)
    {
        terminales[j]=simbolos_terminales[i];
        j++;
        terminales[j]=',';
        j++;
        terminales[j]='\0';
    }
    terminales[j-1]='}';
    terminales[j+1]='\0';
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void phase2(char* producciones,char* simbolos_no_terminales,char*
simboloInicial,int *alfabeto)
Descripción:En la fase 2 se eliminan las producciones inalcanzables
Recibe: Referencia a la cadena "producciones", referencia a la cadena
"simbolos_no_terminales", referencia a la cadena "simboloInicial"
y al arreglo de enteros "alfabeto".
Devuelve: NADA
Observaciones:
*/
void phase2(char* producciones,char* simbolos_no_terminales,char*
simboloInicial,int *alfabeto)
{
    char auxiliar[1001];
    printf("FASE 2 :ELIMINACION DE SIMBOLOS INACCESIBLES\n");

phase2Paso1(producciones,simbolos_no_terminales,simboloInicial,auxiliar,alfabeto)
;
```

```
    phase2Paso2(producciones,simbolos_no_terminales,auxiliar,alfabeto);
    phase2Paso3(producciones,simbolos_no_terminales,auxiliar,alfabeto);
    obtenerSimbolosNoTerminales(alfabeto,simbolos_no_terminales);
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void phase2Paso1(char *producciones,char*simbolos_no_terminales,char*
simboloInicial,char* auxiliar,int*alfabeto)
Descripción:En este paso se agrega a un nuevo conjunto,
el simbolo inicial y los simbolos no terminales que produce
Recibe: Referencia a la cadena "producciones", referencia a la cadena
"simbolos_no_terminales", referencia a la cadena "simboloInicial",
referencia a la cadena "auxiliar" y al arreglo de enteros "alfabeto".
Devuelve: NADA
Observaciones:
*/
void phase2Paso1(char *producciones,char*simbolos_no_terminales,char*
simboloInicial,char* auxiliar,int*alfabeto)
{
    int i=0,j=0,k=1,tam,posicion;
    inicializarArregloBanderas(alfabeto);
    strcpy(auxiliar,simboloInicial);
    auxiliar[k]='\0';
    posicion=auxiliar[0]-65;
    alfabeto[posicion]=1;
    while(j==0)
    {
        if(isupper(producciones[i]))
        {
            posicion=producciones[i]-65;
            if(alfabeto[posicion]==0)
            {
            alfabeto[posicion]=1;
            auxiliar[k]=producciones[i];
            k++;
            auxiliar[k]='\0';
            }
        }
        else if(producciones[i]=='.')
        {
            j++;
        }
        i++;
    }
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void phase2Paso2(char *producciones,char*simbolos_no_terminales,char*
auxiliar,int*alfabeto)
Descripción:Se agregan los simbolos no terminales que producen los
simbolos agregados en el paso anterior
Recibe:Referencia a la cadena "producciones", referencia a la cadena
"simbolos_no_terminales", referencia a la cadena "simboloInicial"
y al arreglo de enteros "alfabeto".
Devuelve: NADA
Observaciones:
*/
void phase2Paso2(char *producciones,char*simbolos_no_terminales,char*
auxiliar,int*alfabeto)//por confirmar que este bien
{
    int i=0,j=1,k=0,posicion;
```

```
    while(j<strlen(auxiliar))
    {
        k=0;
        i=0;
        posicion=auxiliar[j]-65;
        while(k!=posicion+1)
        {
            if(producciones[i]=='.')
                k++;
            i++;
        }
        while(producciones[i]!='.')
        {
            if(isupper(producciones[i]))
            {
            posicion=producciones[i]-65;
            if(alfabeto[posicion]==0)
                {
                alfabeto[posicion]=1;
                auxiliar[strlen(auxiliar)]=producciones[i];
                auxiliar[strlen(auxiliar)]='\0';
                }
            }
            i++;
        }
        j++;
    }
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void phase2Paso3(char *producciones,char*simbolos_no_terminales,char*
auxiliar,int*alfabeto)
Descripción:Actualiza las producciones nuevas ,omitiendo los simbolos
no terminales que quedaron fuera del conjunto final
Recibe: Referencia a la cadena "producciones", referencia a la cadena
"simbolos_no_terminales", referencia a la cadena "auxiliar"
y al arreglo de enteros "alfabeto".
Devuelve: NADA
Observaciones:
*/
void phase2Paso3(char *producciones,char*simbolos_no_terminales,char*
auxiliar,int*alfabeto)
{
    int i=0,j=0,k=0,x=0,posicion;
    char producciones_fase2[1001],auxiliar2[1001];
    int punto=0;
    for(i=0;i<strlen(simbolos_no_terminales);i++)
    {
        posicion=simbolos_no_terminales[i]-65;
        if(alfabeto[posicion]==1)
        {
            punto=0;
            while(punto==0)
            {
                if(producciones[j]!='.')
                {
                    auxiliar2[k]=producciones[j];
                    k++;
                    auxiliar2[k]='\0';
                }
                else
                {
```

```
                    punto=1;
                    strcat(producciones_fase2,auxiliar2);
                    strcat(producciones_fase2,".");
                    k=0;
                    auxiliar2[0]='\0';
                }
                j++;
            }
        }
        else
        {
            punto=0;
            while(punto==0)
            {
                if(producciones[j]=='.')
                {
                    punto=1;
                }
                j++;
            }
        }
    }
    strcpy(producciones,producciones_fase2);
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void obtenerSimbolosNoTerminales(int *alfabeto,char *simbolos_no_terminales)
Descripción:Obtiene los simbolos no terminales de un conjunto dado
Recibe: Referencia a la cadena "simbolos_no_terminales" y al arreglo de enteros
"alfabeto".
Devuelve: NADA
Observaciones:
*/
void obtenerSimbolosNoTerminales(int *alfabeto,char *simbolos_no_terminales)
{
    int i=0,j=0;
    char auxiliar[1001];
    for(i=0;i<strlen(simbolos_no_terminales);i++)
    {
        if(alfabeto[(simbolos_no_terminales[i]-65)]==1)
        {
            auxiliar[j]=simbolos_no_terminales[i];
            j++;
            auxiliar[j]='\0';
        }
    }
    strcpy(simbolos_no_terminales,auxiliar);
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void removerEpsilon(char *producciones,char *simbolos_no_terminales)
Descripción:remueve los epsilons de la gramatica
Recibe: Referencia a la cadena "producciones" y al la cadena
"simbolos_no_terminales".
Devuelve: NADA
Observaciones:
*/
void removerEpsilon(char *producciones,char *simbolos_no_terminales)
{
    int i=0,j=0,k=0,posicion,haynulo=0;
    while(producciones[i]!='.')
```

```
    {
        i++;
    }
    j++;
    i++;
    while(i<strlen(producciones))
    {
        if(producciones[i]=='.')
        {
            if(haynulo==1)
            {

sustituirEpsilonEncontrado(simbolos_no_terminales,producciones,simbolos_no_termin
ales[j]);
                j=0;
                haynulo=0;
            }
            else
            {
            j++;
            }
        }
        else if(producciones[i]=='*')
        {
            haynulo=1;
        }
        i++;
    }
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void sustituirEpsilonEncontrado(char* simbolos_no_terminales,char
*producciones,char simboloEpsilon)
Descripción:sustituye un epsilon encontrado en todas las producciones
,sustituyendo el simbolo no terminal que genera a este
en las producciones que contengan a este simbolo
Recibe: Referencia a la cadena "producciones", referencia a la cadena
"simbolos_no_terminales", referencia a la cadena "simboloEpsilon"
Devuelve: NADA
Observaciones:
*/
void sustituirEpsilonEncontrado(char* simbolos_no_terminales,char
*producciones,char simboloEpsilon)
{
    int i=0,j=0,k=0,x=0,y=0,punto=0,haynulo=0,indice=0;
    char auxiliar2[1001];
    char auxiliar3[1001];
    char producciones_s[1001];
    for(i=0;i<strlen(simbolos_no_terminales);i++)
    {
            punto=0;
            while(punto==0)
            {
                if(isalpha(producciones[j]) || isdigit(producciones[j]))
                {
                    if(producciones[j]==simboloEpsilon)
                        haynulo=1;
                    auxiliar2[k]=producciones[j];
                    k++;
                    auxiliar2[k]='\0';
                }
                else if(producciones[j]=='*')
```

```
                {
                    if(producciones[j+1]=='.')
                        j++;
                    else
                        j=j+2;
                }
                else if(producciones[j]=='|' && producciones[j+1]=='*')
                {
                    j++;
                }
                else
                {
                if(strlen(auxiliar2)!=0)
                {
                    for(x=0;x<strlen(auxiliar2);x++)
                    {
                        if(haynulo==1)
                        {
                            if(auxiliar2[x]!=simboloEpsilon)
                                {
                                auxiliar3[y]=auxiliar2[x];
                                y++;
                                auxiliar3[y]='\0';
                                }
                        }
                    }
                }
                    if(producciones[j]=='|')
                    {
                        if(indice==0)
                        {
                            strcat(producciones_s,auxiliar2);
                            auxiliar2[0]='\0';
                            indice=1;
                            k=0;
                            if(haynulo==1)
                            {
                                strcat(producciones_s,"|");
                                strcat(producciones_s,auxiliar3);
                                y=0;
                                auxiliar3[y]='\0';
                                haynulo=0;
                            }
                            y=0;
                        }
                        else
                        {
                            strcat(producciones_s,"|");
                            strcat(producciones_s,auxiliar2);
                            auxiliar2[0]='\0';
                            k=0;
                            if(haynulo==1)
                            {
                                strcat(producciones_s,"|");
                                strcat(producciones_s,auxiliar3);
                                y=0;
                                auxiliar3[y]='\0';
                                haynulo=0;
                            }
                            y=0;
                        }
                    }
                    else if(producciones[j]=='.')
```

```
                    {
                        if(indice==1)
                        {
                            strcat(producciones_s,"|");
                            strcat(producciones_s,auxiliar2);
                            auxiliar2[0]='\0';
                            if(haynulo==1)
                            {
                                strcat(producciones_s,"|");
                                strcat(producciones_s,auxiliar3);
                                y=0;
                                auxiliar3[y]='\0';
                                haynulo=0;
                            }
                            y=0;
                            strcat(producciones_s,".");
                            punto=1;
                            k=0;
                            indice=0;
                        }
                        else
                        {
                            strcat(producciones_s,auxiliar2);
                            auxiliar2[0]='\0';
                            if(haynulo==1)
                            {
                                strcat(producciones_s,"|");
                                strcat(producciones_s,auxiliar3);
                                y=0;
                                auxiliar3[y]='\0';
                                haynulo=0;
                            }
                            strcat(producciones_s,".");
                            punto=1;
                            k=0;
                            indice=0;
                            y=0;
                        }
                    }
                }
                j++;
            }
    }
    strcpy(producciones,producciones_s);
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void removerUnitarias(char *simbolos_no_terminales,char *producciones)
Descripción:Remueve producciones unitarias
Recibe: Referencia a la cadena "producciones" y referencia a la cadena
"simbolos_no_terminales".
Devuelve: NADA
Observaciones:
*/
void removerUnitarias(char *simbolos_no_terminales,char *producciones)
{
    int
i=0,j=0,k=0,x=0,posicion,punto=0,simbolosNoTerminales=0,indice=0,totales=0;
    char auxiliar[1001],producciones_u[1001];
    do
    {
    i=0;
```

```
    j=0;
    k=0;
    for(i=0;i<strlen(simbolos_no_terminales);i++)
    {
        punto=0;
        while(punto==0)
        {
            if(isalpha(producciones[j]) || isdigit(producciones[j]))
            {
                auxiliar[k]=producciones[j];
                k++;
                auxiliar[k]='\0';
                totales++;
                if(isupper(producciones[j]))
                {
                    simbolosNoTerminales++;
                }
            }
            else if(producciones[j]=='|')
            {
                if(indice==0)
                        {
                            if(simbolosNoTerminales==1 && totales==1)
                            {
                                sustituirUnitarioEncontrado((auxiliar[k-1]-
65)+1,auxiliar,producciones);
                                strcat(producciones_u,auxiliar);
                                auxiliar[0]='\0';
                                k=0;
                                simbolosNoTerminales=0;
                                totales=0;
                            }
                            else
                            {
                                strcat(producciones_u,auxiliar);
                                auxiliar[0]='\0';
                                indice=1;
                                k=0;
                                simbolosNoTerminales=0;
                                totales=0;
                            }
                        }
                        else
                        {
                            if(simbolosNoTerminales==1 && totales==1)
                            {
                                strcat(producciones_u,"|");
                                sustituirUnitarioEncontrado((auxiliar[k-1]-
65)+1,auxiliar,producciones);
                                strcat(producciones_u,auxiliar);
                                auxiliar[0]='\0';
                                k=0;
                                simbolosNoTerminales=0;
                                totales=0;
                            }
                            else
                            {
                                strcat(producciones_u,"|");
                                strcat(producciones_u,auxiliar);
                                auxiliar[0]='\0';
                                k=0;
                                simbolosNoTerminales=0;
                                totales=0;
```

```
                            }
                        }
            }
            else if(producciones[j]=='.')
            {
                if(indice==1)
                        {
                            if(simbolosNoTerminales==1 && totales==1)
                            {
                                strcat(producciones_u,"|");
                                sustituirUnitarioEncontrado((auxiliar[k-1]-
65)+1,auxiliar,producciones);
                                strcat(producciones_u,auxiliar);
                                auxiliar[0]='\0';
                                strcat(producciones_u,".");
                                punto=1;
                                k=0;
                                indice=0;
                                simbolosNoTerminales=0;
                                totales=0;
                            }
                            else
                            {
                                strcat(producciones_u,"|");
                                strcat(producciones_u,auxiliar);
                                auxiliar[0]='\0';
                                strcat(producciones_u,".");
                                punto=1;
                                k=0;
                                indice=0;
                                simbolosNoTerminales=0;
                                totales=0;
                            }
                        }
                        else
                        {
                            if(simbolosNoTerminales==1 && totales==1)
                            {
                                sustituirUnitarioEncontrado((auxiliar[k-1]-
65)+1,auxiliar,producciones);
                                strcat(producciones_u,auxiliar);
                                auxiliar[0]='\0';
                                strcat(producciones_u,".");
                                punto=1;
                                k=0;
                                indice=0;
                                simbolosNoTerminales=0;
                                totales=0;
                            }
                            else
                            {
                                strcat(producciones_u,auxiliar);
                                auxiliar[0]='\0';
                                strcat(producciones_u,".");
                                punto=1;
                                k=0;
                                indice=0;
                                simbolosNoTerminales=0;
                                totales=0;
                            }
                        }
            }
            j++;
```

```
        }
    }
    strcpy(producciones,producciones_u);
    producciones_u[0]='\0';
    }while(validarSiQuedanUnitarias(producciones));
};

/*
DEFINICIÓN DE PROCEDIMIENTO
void sustituirUnitarioEncontrado(int limite,char *auxiliar,char *producciones)
Descripción:Sustituye el valor de las producciones
del simbolo no terminal que haya sido identificado como unitario
Recibe: Referencia a la cadena "auxiliar", referencia a la cadena "producciones"
y el entero "limite".
Devuelve: NADA
Observaciones:
*/
void sustituirUnitarioEncontrado(int limite,char *auxiliar,char *producciones)
{
    int i=0,j=0,k=0;
    while(j!=limite)
    {
        if(producciones[i]=='.')
            j++;
        i++;
    }
    while(producciones[i]!='.')
    {
        auxiliar[k]=producciones[i];
        k++;
        auxiliar[k]='\0';
        i++;
    }
};

/*
DEFINICIÓN DE FUNCIÓN
int validarSiQuedanUnitarias(char *producciones)
Descripción:
Recibe: Referencia a la cadena "producciones"
Devuelve:valida si quedan producciones unitarias en las producciones de la
gramática
Observaciones:
*/
int validarSiQuedanUnitarias(char *producciones)
{
    int i=0,j=0,k=0,unitarias=0,totales=0,nt=0;
    for(i=0;i<strlen(producciones);i++)
    {
        if(isalpha(producciones[i])|| isdigit(producciones[i]))
        {
            totales++;
            if(isupper(producciones[i]))
            {
                nt++;
            }
        }
        else
        {
            if(nt==1 && totales==1)
            {
                unitarias++;
                totales=0;
```

```
                nt=0;
            }
            else
            {
                totales=0;
                nt=0;
            }
        }
    }
    if(unitarias==0)
        return 0;
    else
        return 1;
};
```

Funcionamiento

*Trabajando con las siguentes producciones:

Ejemplo 1:

Vocabulario no terminal={S, A, B, C}

Vocabulario terminal={a,b,c,d}

P1->abS | abA | abB

P2->cd

P3->aB

P4->dc

Ejemplo 2:

Vocabulario no terminal={S,A,B,C,D,E,F,G,H}

Vocabulario terminal={a,b,c,d,e,f,g,h,x,y,z,t}

P1->aAB|aA|cBd|cd|Ha|bH|AH|cB|c|FG

P2->cBd|cd|Ha|bH|AH|cB|c

P3->e|fS

P4->gD|hDt

P5->x|y|z

P6->AH|cB|c

P7->AB|cBd|cd|Ha|bH|AH|cB|c|Ga

P8->FG

P9->Ha|bH|AH|cB|c

➔Probando los ejemplos anteriores de GLC sucias:

1.-Ejemplo 1:

```
C:\Users\FERNANDO\Desktop\TERCER SEMESTRE\Teoria_Computacional\LABORATORIO\Práctica_06\Versión-2_FINAL>a
Digita los simbolos no terminales separados por comas (S0,S1...Sn):
S,A,B,C
Simbolos No terminales:SABC
Digite las producciones de los respectivos simbolos con simbolos terminales(letras minusculas,* para representar epsilon)
no terminales (i.e S->a|b|aS...):
S->abS|abA|abB
A->cd
B->aB
C->dc
FASE 1 :ELIMINACION DE SIMBOLOS NO GENERATIVOS
RESULTADOS DE LA FASE 1:
La Gramatica G esta dada por la 4drupla:
G:({A,C,S},{a,b,c,d},P,{S})
Donde las producciones P:
S->abS|abA
A->cd
C->dc
FASE 2 :ELIMINACION DE SIMBOLOS INACCESIBLES
RESULTADOS DE LA FASE 2:
La Gramatica G esta dada por la 4drupla:
G:({A,S},{a,b,c,d},P,{S})
Donde las producciones P:
S->abS|abA
A->cd
```

2.- Ejemplo 2:

```
Digita los simbolos no terminales separados por comas (S0,S1...Sn):
S,A,B,C,D,E,F,G,H
Simbolos No terminales:SABCDEFGH
Digite las producciones de los respectivos simbolos con simbolos terminales(letras minusculas,* para representar epsilon)
no terminales (i.e S->a|b|aS...):
S->aAB|aA|cBd|cd|Ha|bH|AH|cB|c|FG
A->cBd|cd|Ha|bH|AH|cB|c
B->e|fS
C->gD|hDt
D->x|y|z
E->AH|cB|c
F->AB|cBd|cd|Ha|bH|AH|cB|c|Ga
G->FG
H->Ha|bH|AH|cB|c
FASE 1 :ELIMINACION DE SIMBOLOS NO GENERATIVOS
RESULTADOS DE LA FASE 1:
La Gramatica G esta dada por la 4drupla:
G:({A,B,C,D,E,F,H,S},{a,c,d,b,e,f,g,h,t,x,y,z},P,{S})
Donde las producciones P:
S->aAB|aA|cBd|cd|Ha|bH|AH|cB|c
A->cBd|cd|Ha|bH|AH|cB|c
B->e|fS
C->gD|hDt
D->x|y|z
E->AH|cB|c
F->AB|cBd|cd|Ha|bH|AH|cB|c
H->Ha|bH|AH|cB|c
FASE 2 :ELIMINACION DE SIMBOLOS INACCESIBLES
RESULTADOS DE LA FASE 2:
La Gramatica G esta dada por la 4drupla:
G:({A,B,H,S},{a,c,d,b,e,f,g,h,t,x,y,z},P,{S})
Donde las producciones P:
S->aAB|aA|cBd|cd|Ha|bH|AH|cB|c
A->cBd|cd|Ha|bH|AH|cB|c
B->e|fS
H->Ha|bH|AH|cB|c
```

Conclusión

Fernando

Como bién se obsevó que una gramática limpia y bién formada facilita el correcto tratamiento y detección a la hora de ser impuesta en algún lenguage, para nosotros poder construir una gramática adecuada debemos verificar la correcta escritura de las reglas de producción de gramática, esto depues de ya haber evitado la redundancia, simbolos inutiles así como producciónes entonces nuestra gramatica estará en condiciones de producirnos las mismas cadenas que antes sin ningun problema. Y pues para este programa limpiador se tomaron muy en cuneta estas condiciónes.

Luis

La limpieza de una gramática libre de contexto nos facilita el trabajo de estas al momento de querer producir cadenas que formen parte del lenguaje, a demás de que nos ayudan a evitar redundancia y simbolos inútiles o inaccesibles en gramáticas,tambien más adelante nos ayuda para proximos procesos de normalización.

Anexo

Bibliografia

- “Teoría Computacional”, notas de la clase " Gramáticas Libres de Contexto Limpias y Bien Formadas ", Ingeniería en Sistemas Computacionales, ESCOM IPN, 2018
- “Teoría Computacional”, notas de la clase " Unidad III: Gramáticas Libres de Contexto", Ingeniería en Sistemas Computacionales, ESCOM IPN, 2018
- UNIDAD II: LENGUAGES REGULARES, ”Autómatas Finitos Deterministas”,2018.[Online].Disponible: http://www.ia.urjc.es/grupo/docencia/automatas_itis/apuntes/capitulo5.pdf

- “Teoría Computacional”, notas de la clase " Autómata Finito Determinista ", Ingeniería en Sistemas Computacionales, ESCOM IPN, 2018

- “Teoría Computacional”, notas de la clase " Lenguajes Regulares", Ingeniería en Sistemas Computacionales, ESCOM IPN, 2018

- “Teoría Computacional”, notas de la clase " Cadenas y sus Operaciones", Ingeniería en Sistemas Computacionales, ESCOM IPN, 2018

- TablaAscii, ”El código ASSCII”,2017.[Online].Disponible: http://www.elcodigoascii.com.ar/codigo-americano-estandar-intercambio-informacion/codigo-ascii.png

- Librerías ANSI C, “C++ CON CLASE”, 2014. [Online]. Disponible: http://c.conclase.net/librerias/?ansifun=pow#inicio

- String.h, “string.h”,26 octubre de 2016. [Online]. Disponible: https://es.wikipedia.org/wiki/String.h