UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Universidad Tecnológica Nacional

Facultad Regional Buenos Aires

Sintaxis y Semántica de los Lenguajes

Curso K2102

Trabajo práctico Nro. 1

Generador de lenguajes

Fecha de entrega: 18/06/2024

Profesor: Esp. Ing. Pablo D. Mendez

Integrantes:

Albornoz, Tobias Abel (legajo: 209.930-5) @tobias-albornoz
<talbornoz@frba.utn.edu.ar>

Figueredo Chirinos, Jesus Antonio (legajo: 176.605-3) @phosph
<jfigueredochirinos@frba.utn.edu.ar>

Scarfo, Ignacio Alejo (legajo: 214.123-1) @iscarfo
<iscarfo@frba.utn.edu.ar>

Venero Alosilla, Nicolás Alejandro (legajo: 202.792-6) @nickvenero
<nveneroalosilla@frba.utn.edu.ar>

El presente trabajo consiste en realizar un programa en lenguaje C que permita generar aleatoriamente palabras de un lenguaje regular a partir de una gramática ingresada por el usuario.

A fin organizativo, decidimos ordenar el trabajo práctico en diferentes archivos de código fuente, en donde se tienen en cuenta diversos aspectos:

- la carpeta `src/` contiene el código en C. Dentro de este se encontrarán los archivos .h que contienen las cabeceras de funciones y definiciones de estructuras y tipos de datos, y los archivos .c en los cuales se desarrollan los subprogramas.

- El código fue dividido en diferentes archivos de cabeceras (.h) y fuentes (.c) para mejorar la legibilidad del código. Estos son:

    - `src/`**`defs.h`**,
    - `src/`**`gramatica.h`** y `src/`**`gramatica.c`**
    - `src/`**`producciones.h`** y `src/`**`producciones.c`**
    - `src/`**`utils.h`** y `src/`**`utils.c`**
    - `src/`**`main.c`**.

- Todos archivos .h contienen una protección contra inclusiones múltiples de archivo de cabeceras (#include guards), utilizando las macros `#ifndef` y `#define`.

- Makefile que tiene algunos comandos para simplificar el proceso de compilación.

  Comandos:

    - `make build`: compila con flags de gcc útiles para debugging.
    - `make release` : compila con flags de gcc de optimización.
    - `make clean` : elimina los archivos generados por la compilación.
    - `make run` : compila y ejecuta el programa compilado.
    - `make all` : equivalente a make clean build.

- Fue desarrollado utilizando el compilador gcc, sin embargo la implementación no depende del mismo ni de las librerías de GNU.

- El código fuente fue hecho de modo siguiendo el estándar C99.

src/**defs.h**

```
#include "src/defs.h"
```

Tipos de datos

- `bool_t`
  equivalente a `char`, creado con fines semánticos para representar valores booleanos.

Macros

`true`	`1`
`false`	`0`

Se definen macros para representar valores booleanos (true y false), y un tipo de dato personalizado (`bool_t`) que es usado para variables booleanas. Estas prácticas son comunes en la programación en C para mejorar la legibilidad y la seguridad del código.

src/**gramatica.h** y src/**gramatica.c**

```
#include "src/gramatica.h"
```

Tipos de datos:

```
gramatica_t;
```

Funciones:

```
gramatica_t * inicializarGramtica();
```

En la estructura `gramatica_t` se definen: lista de no terminales, lista de terminales, lista de producciones. y el axioma. También posee variables que almacenan la longitud de cada una de las listas. En estas variables de longitud utilizamos el tipo estandar `size_t`, y las utilizamos para saber cuántos datos tenemos en cada una de las listas y no analizar de más y generar overhead innecesario al sistema

En el archivo src/gramatica.c se define la función `inicializarGramtica`. Esta se encarga de inicializar y devolver una instancia única y persistente de la estructura `gramatica_t`. Utiliza una variable estática (`static`) para asegurar que solo exista una instancia de la gramática en todo el programa. Inicializa todos los miembros de la estructura, incluyendo los arreglos de terminales y no terminales, la cantidad de estos símbolos, las producciones y el axioma, dejándolos en un estado limpio y vacío al inicio del uso de la gramática. Cabe aclarar que devuelve un puntero a `gramatica_t` ya que la estructura es demasiado grande como para devolverla por valor.

src/**producciones.h** y src/**producciones.c**

```
#include "src/producciones.h"
```

Se definen estructuras y funciones para la representación y manipulación de producciones de una gramática formal.

Tipos de datos:

`produccion_t`

representa a una producción de una gramática formal. En esta implementación se busca representar diferentes opciones para un mismo no terminal en una única estructura. similar a la notación formal: S→sT | aB | a .
En esta implementación, cada una de estas producciones con lado izquierdo igual, se lo denomina “opción” de una producción y se permite como máximo 10.

`lista_produciones_t`

representa una lista de producciones. se permite como máximo 10 producciones con lado izquierdo distinto

Funciones:

```
bool_t agregarProduccion(lista_produciones_t *, produccion_t *);
```

agrega una producción pasada como segundo argumento a la lista de producciones pasada como primer argumento. La implementación asegura que se respeten los límites predefinidos (como un máximo de 10 producciones con un No Terminal distinto en el lado derecho).
Devuelve 1 (true) como señal de error si no se pudo agregar la la producción a la lista, en caso contrario devuelve 0 (false).

```
produccion_t * crearProduccion(char);
```

Crea una nueva instancia de la estructura `produccion_t` alocando memoria dinámica, la inicializa y devuelve un puntero a esta.

```
produccion_t * buscarProduccion(char, lista_produciones_t *);
```

busca una producción cuyo lado derecho coincida con el no terminal pasado como primer argumento, en la lista de producciones pasada como segundo argumento.

```
bool_t agregarOpcionAProduccion(produccion_t *, char *);
```

La implementación asegura que se respeten los límites predefinidos (como un máximo de 10 opciones por produccion, o 10 producciones para un mismo no terminal).
Devuelve 1 (true) como señal de error si no se pudo agregar la opción a la producción, en caso contrario devuelve 0 (false).

Este conjunto de funciones implementa la manipulación de producciones gramaticales dentro de una lista. Permite agregar producciones a una lista, crear nuevas producciones, buscar producciones por su no terminal y agregar opciones (o casos) al lado derecho de una producción. La implementación asegura que se respeten los límites predefinidos (como un máximo de 10 producciones u opciones→precondición) y maneja la inicialización adecuada de las estructuras para evitar errores relacionados con datos basura en la memoria asignada dinámicamente.

src/**utils.h** y src/**utils.c**

```
#includes "src/utils.h"
```

Funciones:

`bool_t esNoTerminalValido(char)`
verifica si el caracter provisto como argumento, es un caracter válido como no terminal. Se consideran caracteres válidos a aquellos que cumplan con `[A-Z]`

`bool_t esTerminalValido(char)`
verifica si el caracter provisto como argumento, es un caracter válido como terminal. Se consideran caracteres válidos a aquellos que cumplan con `[a-z]`

`bool_t existeNoTerminal(gramatica_t *, char)`
verifica si el caracter provisto como segundo argumento es un no terminal de la gramática provista como primer argumento.

`bool_t existeTerminal(gramatica_t *, char)`
verifica si el caracter provisto como segundo argumento es un terminal de la gramática provista como primer argumento.

`bool_t contieneNoTerminal(char *, gramatica_t *)`
Verifica si una cadena contiene al menos un no terminal válido de la gramática.

`void imprimirNoTerminales(gramatica_t *)`
imprime por la salida estándar la lista de No terminales de una gramática,

`void imprimirTerminales(gramatica_t *)`
imprime por la salida estándar la lista de terminales de una gramática,

`void imprimirProducciones(lista_produciones_t *)`
imprime por la salida estándar la lista de producciones de una gramática,

`void imprimirGramatica(gramatica_t *)`

imprime por la salida estándar la gramática provista como argumento.

`void limpiarBufferDeEntrada()`

busca limpiar el buffer de entrada estándar (`stdin`) hasta encontrar un salto de línea (\n) para no mantener datos “basura” y evitar obtener datos erróneos en subsiguientes usos de funciones que utilicen la entrada estándar.

Esta función se utiliza cuando no se lee por completo el buffer de entrada estándar, por ejemplo, utilizando `getchar` o `fgets`

src/**main.c**

Tipo de datos:

`linealidad_t`

enum utilizado como valor de retorno por `verificarLinealidad`. Se definen las constantes enumeradas:

NO_LINEAL	-1
LINEAL_A_IZQUIERDA	0
LINEAL_A_DERECHA	1
NO_APLICA	2

Funciones

`int main()`

función principal de toda la implementación.

`void cargarNoTerminales(gramatica_t *);`

solicita al usuario cargar los no terminales por consola, y en caso de ser válidos, los almacena en la gramática pasa como parámetro.

`void cargarTerminales(gramatica_t *);`

solicita al usuario cargar los terminales por consola, y en caso de ser válidos, los almacena en la gramática pasa como parámetro.

`void cargarProducciones(gramatica_t *);`

solicita al usuario cargar las producciones por consola, y en caso de ser válidos, los almacena en la gramática pasa como parámetro.

`void cargarAxioma(gramatica_t *);`

solicita al usuario cargar el axioma, y en caso de ser válido, lo almacena en la gramática pasa como parámetro.

`void ingresarGramatica(gramatica_t *);`

función utilitaria que solicita al usuario cargar los diferentes componentes de la gramática pasa por parámetros, utilizando las funciones previamente definidas y luego imprime resultado.

`linealidad_t verificarLinealidad(char*, char*, char*);`

verifica la linealidad de una producción, analizando el lado derecho de la produccion (primera parámetro) y según una lista de terminales (segundo parámetro) y una lista de no terminales (tercer parámetro).

Para verificar la linealidad se analiza la estructura de una una de la producciones, se determina si esta es lineal a la izquierda, lineal a la derecha, no lineal, o no aplica y devuelve el tipo de linealidad de la producción.

Utilizada por la función `esGramaticaRegular.`

`bool_t esGramaticaRegular(gramatica_t *);`

Verifica si la gramática pasa por parámetro es regular. devuelve 1 (`true`) en caso de serlo, de lo contrario devuelve 0 (`false`).

Se verifica que todas las producciones tengan en su lado izquierdo un No terminal que se encuentre en la lista de no terminales de la gramática; determina si las producciones son lineales (a zquierda o a derecha) y si son consistentes en su linealidad utilizando la función `verificarLinealidad`.

`void generarPalabras(gramatica_t *);`

genera una palabra aleatoria utilizando la gramática pasada por parámetro.

Acá se desarrolla el “corazón” del código. En este se define la función main en la cual se establece los pasos a seguir del programa:

1. inicializar una nueva gramática,
2. solicitar al usuario ingresar una gramática,
3. verificar si la gramática ingresada es regular y
4. generar palabras aleatorias si la gramática es formal.

También se definen las funciones de carga de axioma, terminales, no terminales y producciones, las cuales solicitan al usuario que ingrese las definiciones, verifican si son válidas y las almacena dentro de la gramática.

Siguiendo con esto, está la generación de palabras aleatorias, implementada en la función `generarPalabras`, que inicializa el generador de números aleatorios, genera palabras aleatorias comenzando desde el axioma, muestra el proceso de derivación paso a paso y permite al usuario generar múltiples palabras hasta que decida detenerse.