Compiladores
Proyecto

Prof. José Fiestas
UTEC

Instrucciones para desarrollar el proyecto:

- formen grupos de 2 o 3 estudiantes
- la entrega será: el proyecto programado en lex/yacc, flex/bison, y un informe en LaTeX
- El informe debe tener capítulos correspondientes a: **Introducción** (con la base teórica del problema), **Método y Desarrollo**, **Resultados y Conclusiones**
- escoja uno de los 5 ejercicios. La calificación será sobre 20
- La nota de proyecto será igual a $\mathbf{P = 0.5 \cdot D + 0.4 \cdot Pe + 0.1 \cdot Po}$, donde **D** es el desarrollo de códigos de programación, **Pe** es la presentación escrita , y **Po** es la presentación oral
- Entrega: viernes 11 de Diciembre

1 Cifras en alemán

Programe en C++ la gramática de números cardinales en alemán en forma normal de Chomsky:
$\mathbf{R_1 : Z_2 \rightarrow Z_3 + zehn}$
$\mathbf{R_2 : Z_7 \rightarrow Z_4 + zig}$
$\mathbf{R_3 : Z_7 \rightarrow drei + ssig}$
$\mathbf{R_4 : Z_8 \rightarrow U + Z_7}$
$\mathbf{R_5 : Z_9 \rightarrow Z_1 + Z_5}$
$\mathbf{R_6 : Z_{10} \rightarrow Z_2 + Z_5}$
$\mathbf{R_7 : Z_{11} \rightarrow Z_{5,9} + Z_{1,2,7,8}}$
$\mathbf{R_8 : Z_{12} \rightarrow Z_{10} + Z_{1,2,7,8}}$
$\mathbf{R_9 : Z_{13} \rightarrow Z_{1,2,7,8,9,11} + Z_6}$
$\mathbf{R_{10} : Z_{14} \rightarrow Z_{6,13} + Z_{1,2,7,8,9,11}}$
$\mathbf{R_{11} : U \rightarrow Z_1 + und}$

Donde:
Z_1 = {ein, zwei, drei, vier, fünf, sechs, sieben, acht, neun}
Z_2 = {zehn, elf, zwölf }
Z_3 = {drei, vier, fünf, sech, sieb, acht, neun }
Z_4 = {zwan, vier, fünf, sech, sieb, acht, neun }
Z_5 = {hundert}
Z_6 = {tausend}
drei = {drei}
zehn = {zehn}
zig = {zig}
ssig = {ssig}
und = {und}

El proyecto, consistirá de las siguientes partes:

1. Dado un número entero, programar el **scanner** correspondiente (i.e. que determina expresiones v'alidas pero no ciertas o falsas)
2. Programar el **parser** correspondiente con un estado de aceptación y rechazo
3. Implementar además una regla de identificación de errores con un mensaje de error
4. Implementar las salidas del análisis (aceptado o rechazado)

Probar el compilador con los siguientes números:

- **fünftausendzweihundertneunundfünfzig**
- **zweitausendneunhundertsechsundsiebzig**
- **zweihundertzweiundzwanzigtausendvierhundertsiebzehn**

2 Árbol ancestral

Construya un compilador para traducir términos ancestrales.
Utilice el lenguaje ancestral
A_1={ mother, father, grandmother, grandfather, greatgrandmother, greatgrandfather, greatgreatgrandmother, ..., greatgreatgreatgreatgrandfather, ...}
y construya una gramática que lo reproduzca.
Traduzca asimismo las expresiones de entrada en el número de generaciones que existen entre los ancestros y la persona. I.e. **mother** y **father** : 1. generación, **grandmother** y **grandfather**: 2. generación, **greatgrandmother** y **greatgrandfather**: 3. generación, etc..
Traduzca las expresiones a un lenguaje intermedio, como:
mother, mo()
father, fa()
grandmother, g(mo())
grandfather, g(fa())
greatgrandmother, g(g(mo()))
greatgrandfather, g(g(fa()))
...
greatgreatgreatgrandmother , g(g(g(g(mo()))))
greatgreatgreatgrandfather, g(g(g(g(fa()))))
...

Almacene las expresiones en una lista de tokens y reconstrúyalas al alemán según el lenguaje
A_2 ={mutter, vater, grossmutter, grossvater, urgrossmutter, urgrossvater, ururgrossmutter, ...} .
E.g. la palabra **greatgreatgrandfather** de A_1 debe traducirse en la lista **string["ur", "ur", "gross", "vater"]**.

El compilador debe también traducir una expresión del lenguaje $\mathbf{A_3}$ en uno del lenguaje $\mathbf{A_4}$, según la siguiente tabla

A_3	A_4
The mother of Mary	Die mutter von Maria
The father of Mary	Der vater von Maria
The mother of John	Die mutter von Johann
The father of John	Der vater von Johann
The mother of the mother of Mary	Eine grossmutter von Maria
The mother of the father of Mary	Eine grossmutter von Maria
The father of the mother of Mary	Ein grossvater von Maria
The father of the father of Mary	Ein grossvater von Maria
The mother of the mother of John	Eine grossmutter von Johann
The mother of the father of John	Eine grossmutter von Johann
The father of the mother of John	Ein grossvater von Johann
The father of the father of John	Ein grossvater von Johann
...	...
The mother of the mother of the father of the mother of John	Eine urgrossmutter von Johann

El proyecto, consistirá entonces de las siguientes partes:

1. Construcción de la **gramática**
2. Programar el **parser** correspondiente
3. Construir una **representación intermedia** de las expresiones
4. Implementar atributos en el **parser**, que hagan la traducción
5. Implementar las salidas de traducción

3 Calculadora científica

Construya un compilador que desarrolle una calculadora, como la construída en clase, con las siguientes funciones:

- **- expr** : negación de la expresión
- **++ var**: incremento de la variable en uno
- **- - var**: decremento de la variable en uno
- **expr + expr** : suma de dos expresiones
- **expr - expr** : diferencia de dos expresiones
- **expr * expr** : producto de dos expresiones
- **expr / expr** : cociente de dos expresiones
- **expr + expr** : módulo de dos expresiones
- **expr** $\wedge$ **expr** : primera expresión elevada a la segunda (debe ser entero)
- **(expr)** : redefinir reglas de precedencia con el paréntesis
- **var= expr** : el valor de expresión se asigna a la variable

Además, deberá evaluar comparaciones:

- **expr1** $<$ **expr2** : retorna 1 si expr1 es menor a expr2
- **expr1** $\leq$ **expr2** : retorna 1 si expr1 es menor o igual a expr2
- **expr1** $>$ **expr2** : retorna 1 si expr1 es mayor a expr2
- **expr1** $\geq$ **expr2** : retorna 1 si expr1 es mayor o igual a expr2
- **expr1** == **expr2** : retorna 1 si expr1 es igual a expr2
- **expr1** ! = **expr2** : retorna 1 si expr1 no es igual a expr2

Expresiones booleanas:

- ! **expr** : retorna 1 si expr1 es 0
- **expr1** && **expr2** : retorna 1 si ambas son no-cero
- **expr1** || **expr2** : retorna 1 si alguna de las expresiones es no-cero

Implementando las reglas de precedencia:

- **operador** ||, asociativo por la izquierda

- **operador** &&, asociativo por la izquierda
- **operador** !, no asociativo
- operadores de asignación, asociativo por la derecha
- **operador** $+, -$, asociativo por la izquierda
- **operador** $*, /, \%$, asociativo por la izquierda
- **operador** $\wedge$, asociativo por la derecha

4 Fabrica textil

Una fabrica textil está implementando un sistema automatizado de diseño de flujos de producción optimizados. Este sistema cuenta con reglas que deben ser cumplidas para garantizar un ahorro adecuado de tiempo y recursos. Se propone construir un compilador, que maneje el lenguaje de producción y determine rutas de materia prima adecuadas en una secuencia de operaciones. El flujo de producción textil aceptado está definido por la siguiente gramática:

$\mathbf{A} \rightarrow \mathbf{Sh_1} \mid \mathbf{Sh_2} \mid \mathbf{Sh_3} \mid \mathbf{Sh_4} \mid \mathbf{Sh_5}$
$\mathbf{Sh_1} \rightarrow \mathbf{Sv_1} \mid \mathbf{Sv_2} \mid \mathbf{Sv_3}$
$\mathbf{Sh_2} \rightarrow \mathbf{Sv_1} \mid \mathbf{Sv_2}$
$\mathbf{Sh_3} \rightarrow \mathbf{Sv_1} \mid \mathbf{Sv_2} \mid \mathbf{Sv_3}$
$\mathbf{Sh_4} \rightarrow \mathbf{Sv_3} \mid \mathbf{Sv_4} \mid \mathbf{Sv_5}$
$\mathbf{Sh_5} \rightarrow \mathbf{Sv_3} \mid \mathbf{Sv_5}$
$\mathbf{Sv_1} \rightarrow \mathbf{N_1} \mid \mathbf{N_2} \mid \mathbf{N_3}$
$\mathbf{Sv_2} \rightarrow \mathbf{N_1} \mid \mathbf{N_2} \mid \mathbf{N_3}$
$\mathbf{Sv_3} \rightarrow \mathbf{N_4} \mid \mathbf{N_5}$
$\mathbf{Sv_4} \rightarrow \mathbf{N_4} \mid \mathbf{N_5}$
$\mathbf{Sv_5} \rightarrow \mathbf{N_5}$
$\mathbf{N_1} \rightarrow \mathbf{Ac_1} \mid \mathbf{Ac_2}$
$\mathbf{N_2} \rightarrow \mathbf{Ac_1} \mid \mathbf{Ac_2} \mid \mathbf{Ac_3}$
$\mathbf{N_3} \rightarrow \mathbf{Ac_2} \mid \mathbf{Ac_3} \mid \mathbf{Ac_4}$
$\mathbf{N_4} \rightarrow \mathbf{Ac_2} \mid \mathbf{Ac_3} \mid \mathbf{Ac_4}$
$\mathbf{N_5} \rightarrow \mathbf{Ac_4}$
$\mathbf{Ac_1} \rightarrow \mathbf{C_1} \mid \mathbf{C_2} \mid \mathbf{C_3}$
$\mathbf{Ac_2} \rightarrow \mathbf{C_1} \mid \mathbf{C_2} \mid \mathbf{C_3}$
$\mathbf{Ac_3} \rightarrow \mathbf{C_1} \mid \mathbf{C_2} \mid \mathbf{C_3}$
$\mathbf{Ac_4} \rightarrow \mathbf{C_4}$
$\mathbf{C_1} \rightarrow \mathbf{C_2}$
$\mathbf{C_2} \rightarrow \mathbf{C_4}$
$\mathbf{C_3} \rightarrow \mathbf{A}$
$\mathbf{C_4} \rightarrow \mathbf{A}$

Donde:
A: Almacén, **Sh:** Tejido horizontal, **Sv:** Tejido vertical, **N:** Teñido, **Ac:** Acabado, **C:** Confección. Y los subíndices denotan variaciones de la operación a la que se refieren. I.e. hay distintos tipos de tejido vertical (específicamente 5), hay 4 tipos de acabado ($\mathbf{Ac_1}, \mathbf{Ac_2}, \mathbf{Ac_3}, \mathbf{Ac_4}$), etc.
Para optimizar procesos, se requiere que cada artículo atraviese como máximo 5 etapas. ¿Garantiza esto la gramática planteada?
Elabore un código que permita definir flujos de operaciones permitidas, ingresando expresiones como: $\mathbf{ASh_1Sv_3N_5Ac_4C_4A}$
Implemente reglas de error en caso las operaciones no sean permitidas, y realice las correcciones adecuadas para proponer una alternativa optima al flujo propuesto.

5 Lenguaje CEs (C en Español)

Implemente el lenguaje CEs en Yacc/Lex y genere un código que introduzca dos enteros, calcule su máximo común divisor y lo imprima

5.1 Lexicografía en CEs

1. Palabras reservadas son: **entero retorno sin_tipo mientras si sino** (deben estar escritas en minúscula)
2. Símbolos especiales: + − ∗ / < <= > >= == ! = = ; , ( ) [ ] { } /* */
3. **ID** y **NUM** son token definidos como:
   ID = letra letra*
   NUM = digito digito*
   letra = $a|...|z|A|...|Z$
   digito = $0|...|9$
4. Los espacios en blanco se componen de blancos, retornos de Iínea y tabulaciones. El espacio en blanco es ignorado, excepto cuando deba separar ID, NUM y palabras reservadas.
5. Los comentarios están encerrados como en el lenguaje C /*...*/. Los comentarios se pueden colocar en cualquier lugar donde pueda aparecer un espacio en blanco (es decir, los comentarios no pueden ser colocados dentro de los token) y pueden incluir más de una línea. Los comentarios no pueden estar anidados.

5.2 Gramática en CEs

La gramática está dada por:

1. **programa** → **lista_declaracion**
2. **lista_declaracion** → **lista_declaracion declaracion** | **declaracion**
3. **declaracion** → **var_declaracion** | **fun_declaracion**

Un **programa** se compone de una lista (o secuencia) de declaraciones (**lista_declaracion**), las cuales pueden ser declaraciones de variable o función, en cualquier orden. Debe haber al menos una **declaración**. Las restricciones semánticas son como sigue (distinto a C). Todas las variables y funciones deben ser declaradas antes de utilizarlas (esto evita las referencias de retroajuste). La última declaración en un programa debe ser una declaración de función con el nombre **main**. CEs no hace una distinción entre declaraciones y definiciones (como en el lenguaje C).

4. **var_declaracion → tipo ID ; | tipo ID [NUM]**

5. **tipo → int | void**

Una declaración de variable declara una variable simple de tipo entero o una variable de arreglo cuyo tipo base es entero, y cuyos índices abarcan desde 0 ... NUM-1. Observe que en CEs los únicos tipos básicos son entero y vacío (**void**). En una declaración de variable sólo se puede utilizar el especificador de tipo **int**. **void** es para declaraciones de función. Advierta también que sólo se puede declarar una variable por cada declaración.

6. **fun_declaracion → tipo ID (params) sent_compuesta**

7. **params → lista_params | void**

8. **lista_params → lista_params, param | param**

9. **param → tipo ID | tipo ID [ ]**

Una declaración de función consta de un especificador de tipo de retorno, un identificador y una lista de parámetros separados por comas dentro de paréntesis, seguida por una sentencia compuesta con el código para la función. Si el tipo de retorno de la función es **void**, entonces la función no devuelve valor alguno (es decir, es un procedimiento). Los parámetros de una función pueden ser **void** (es decir, sin parámetros) o una lista que representa los parámetros de la función. Los parámetros seguidos por corchetes son parámetros de arreglo cuyo tamaño puede variar. Los parámetros enteros simples son pasados por valor. Los parámetros de arreglo son pasados por referencia (es decir, como apuntadores) y deben ser igualados mediante una variable de arreglo durante una llamada. Advierta que no hay parámetros de tipo "función". Los parámetros de una función tienen un ámbito igual a la sentencia compuesta de la declaración de función, y cada invocación de una función tiene un conjunto separado de parámetros. Las funciones pueden ser recursivas.

10. **sent_compuesta → { declaracion_local lista_sentencias }**

Una sentencia compuesta se compone de llaves que encierran un conjunto de declaraciones y sentencias. Una sentencia compuesta se realiza al ejecutar Ia secuencia de sentencias en el orden dado. Las declaraciones locales tienen un ámbito igual al de la lista de sentencias de la sentencia compuesta y reemplazan cualquier declaración global.

11. **declaracion_local → declaracion_local var_declaracion | vacio**

12. **lista_sentencias → lista_sentencias sentencia | vacio**

Advierta que tanto la lista de declaraciones como la lista de sentencias pueden estas vacías. (El no terminal **vacio** representa Ia cadena vacía, que se describe en ocasiones como ϵ)

13. **sentencia** → **sentencia_expresion** | **sentencia_compuesta** | **sentencia_seleccion** | **sentencia_iteracion** | **sentencia_retorno**

14. **sentencia_compuesta** → **expresion ;** | **;**

Una sentencia de expresión tiene una expresión opcional seguida por un signo de punto y coma. Tales expresiones por lo regular son evaluadas por sus efectos colaterales. Por consiguiente, esta sentencia se utiliza para asignaciones y llamadas de función.

15. **sentencia_seleccion** → **si (expresion) sentencia** | **si (expresion) sentencia sino sentencia**

La sentencia **si** tiene la semántica habitual: la expresión es evaluada; un valor distinto de cero provoca la ejecución de la primera sentencia; un valor de cero ocasiona la ejecución de la segunda sentencia, si es que existe. Esta regla produce la ambigüedad clásica del **sino** (else) ambigüo, la cual se resuelve de la manera estándar: la parte else siempre se analiza sintácticamente de manera inmediata como una subestructura del **si** actual (la regla de eliminación de ambigüedad "de anidación más cercana")

16. **sentencia_iteracion** → **mientras (expresion) sentencia**

La sentencia **mientras** es la única sentencia de iteración en el lenguaje CEs. Se ejecuta al evaluar de manera repetida la expresión y al ejecutar entonces la sentencia si la expresión evalúa un valor distinto de cero, finalizando cuando la expresión se evalúa a 0.

17. **sentencia_retorno** → **retorno ;** | **retorno expresion ;**

Una sentencia de retorno puede o no devolver un valor. Las funciones no declaradas como **void** deben devolver valores. Las funciones declaradas **void** no deben devolver valores. Un retorno provoca la transferencia del control de regreso al elemento que llama (o la teminación del programa si está dentro de **main**).

18. **expresion** → **var=expresion** | **expresion_simple**

19. **var** → **ID** | **ID [expresion]**

Una expresión es una referencia de variable seguida por un símbolo de asignación (signo de igualdad) y una expresión, o solamente una expresión simple. La asignación tiene la semántica de almacenamiento habitual: se encuentra la localidad de !a variable representada por **var**. Luego se evalúa la subexpresión a la derecha de la asignación, y se almacena el valor de la subexpresión en la localidad dada. Este valor también es devuelto como el valor de la expresión completa. Una **var** es una variable (entera) simple o bien una variable de arreglo subindizada. Un subíndice negativo provoca que el programa se detenga (a diferencia de C). Sin embargo, no se verifican los Iímites superiores de los subíndices. Las variables representen una restricción en CEs respecto a C. En C

el objetivo de una asignación debe ser un **valor l**, y los valores l son direcciones que pueden ser obtenidas mediante muchas operaciones. En CEs los únicos valores l son aquellos dados por Ia sintaxis de **var**, y así esta categoría es verificada sintácticamente, en vez de hacerlo durante la verificación de tipo como en C. Por consiguiente, en CEs está prohibida la aritmética de apuntadores.

20. **expresion_simple** → **expresion_aditiva relop expresion_aditiva** | **expresion_aditiva**

21. **relop** → < | <= | > | >= | == | ! =

Una expresión simple se compone de operadores relacionales que no se asocian (es decir, una expresión sin paréntesis puede tener solamente un operador relacional). El valor de una expresión simple es el valor de su expresión aditiva si no contiene operadores relacionales, o bien, 1 si el operador relacional se evalúa como verdadero. o 0 si se evalúa como falso

22. **expresion_aditiva** → **expresion_aditiva addop term** | **term**

23. **addop** → + −

24. **term** → **term mulop factor** | **factor**

25. **mulop** → ∗ | /

Los términos y expresiones aditivas representan la asociatividad y precedencia típicas de los operadores aritméticos. El símbolo / representa la división entera

26. **factor** → **(expresion)** | **var** | **call** | **NUM**

Un factor es una expresión encerrada entre paréntesis, una variable. que evalúa el valor de su variable; una llamada de una función, que evalúa el valor devuelto de la función; o un **NUM**, cuyo valor es calculado por el analizador léxico. Una variable de arreglo debe estar subindizada, excepto en el caso de una expresión compuesta por una **ID** simple y empleada en una llamada de función con un parámetro de arreglo.

27. **call** → **ID (args)**

28. **args** → **lista_arg** | **vacio**

29. **lista_arg** → **lista_arg , expresion** | **expresion**

Una llamada de función consta de un **ID** (el nombre de la función), seguido por sus argumentos encerrados entre paréntesis. Los argumentos pueden estar vacíos o estar compuestos por una lista de expresiones separadas mediante comas, que representan los valores que se asignarán a los parámetros durante una llamada. Las funciones deben ser declaradas antes de llamarlas, y el número de parámetros en una declaración debe ser igual al número de argumentos en una llamada. Un parámetro de arreglo en una declaración de función debe coincidir con una expresión compuesta de un identificador simple que representa

una variable de arreglo. Finalmente. las reglas anteriores no proporcionan sentencia de entrada o salida. Debemos incluir tales funciones en la definición de CEs, puesto que a diferencia del lenguaje C, CEs no tiene facilidades de ligado o compilación por separado. Por lo tanto, consideraremos dos funciones por ser predefinidas en el ambiente global, corno si tuvieran las declaraciones indicadas.

int input(void) { ... }
void output (int x) { ... }

La función **input** no tiene parámetros y devuelve un valor entero desde el dispositivo de entrada estándar (por lo regular el teclado). La función **output** toma un parámetro entero, cuyo valor imprime a la salida estándar (por lo regular la pantalla), junto con un retorno de línea.