Cpanish++

Teoría de Lenguajes y Automatas

Trabajo Práctico Especial

Integrantes

- Bautista Blacker …….57129
- Martin Craviotto …..……..57756
- Ignacio Grasso …...…….57610
- Felipe Oliver ….……..58439

Índice

Idea subyacente y objetivo del lenguaje.

Entre el mundo de posibilidades que este trabajo contraia, decidimos darle una extensión a un lenguaje que nos pareció sencillo, práctico e inclusivo. Sencillo dado que la sintaxis del mismo no es difícil de aprender, practico ya que es de muy fácil comprensión tanto para el que implementa una porcion de codigo como para el que lo lee y por último inclusivo, dado que cualquier persona de habla hispana puede aprenderlo y no se necesita en lo más mínimo aprender inglés, idioma en el cual está basado la gran mayoría de los lenguajes.

Pero este lenguaje tiene un defecto, que es muy limitado. Las operaciones que son posible es Cpanish no alcanzan para llevar al máximo la experiencia de programar, por lo que nosotros decidimos dar el primer paso hacia lo ilimitado, y ese primer paso es Cpanish++.

Cpanish++ no solo incluye todas la operaciones posibles en Cpanish, sino que también incluye estructuras tales como Mapas, Listas doblemente encadenadas, Stacks y Queues y también tipos de datos nuevos como Bool y Double, creando infinitas posibilidades.

Consideraciones realizadas

Dado que nuestro punto de partida fue Cpanish y luego de leer detenidamente toda consideración que dicho lenguaje implica, consideramos que el desarrollo de Cpanish++ sea también en C, ya que además de ser un lenguaje en el cual los integrantes de este grupo pueden asegurar implementaciones eficientes y prolijas, es un lenguaje para el cual existen compiladores para la mayoría de las arquitecturas.

Descripción del desarrollo del TP

Lo primero que se hizo en los comienzos del desarrollo de este trabajo práctico fue entender la gramática del lenguaje Cpanish, ya que a partir de la misma se iba a construir Cpanish++, y, una vez comprendida, se debatieron maneras de mejorar este lenguaje. De manera casi inmediata surgió la idea de implementar tipo de datos nuevos, y posteriormente, se nos ocurrió la ambiciosa idea de agregarle estructuras, y así es como surgieron las estructuras de datos Queue, Stack, Listas y Mapas.

LEX y *YACC* fueron dos herramientas constantemente utilizadas dado que sin las mismas el trabajo práctico no sería posible. Gracias a estas dos herramientas se pudo mejorar el compilador ya implementado.

Descripción de la gramática

La gramática es muy similar a la de Cpanish ya que tiene la misma de base. De igual manera, enumeramos sus características resaltando algunos de nuestros aportes a la misma en **negrita.**

1. Se parte de un símbolo distinguido PROGRAMA
2. Del mismo derivan los PROTOTIPOS, que se definen al principio, así como el alcance global de variables NUEVO_ALCANCE (representa un ALCANCE que en este caso no tienen padre), seguido de la función de entrada PRINCIPAL, y posteriormente una sucesión opcional de FUNCIONES.
3. Los PROTOTIPOS y variables globales se definen con prefijos que se ilustran en los ejemplos, y pretenden facilitar la comprensión a primera vista del lenguaje.
4. Las FUNCIONES degeneran en no terminales FUNCIÓN, cada una de las cuales devuelve un TIPO de datos, tiene un nombre, y recibe una lista de argumentos con nombre y TIPO propios. Las reglas de nomenclatura para funciones y variables son las mismas: conjunto de números, letras, y guiones precedido siempre de una letra.
5. Una FUNCIÓN contiene un conjunto de LÍNEAS.
6. Un no terminal LINEA degenera en un BLOQUE o en una LÍNEA y más LÍNEAS.
7. Una LÍNEA produce una INSTRUCCIÓN
8. Una INSTRUCCIÓN puede ser una DECLARACIÓN o REASIGNACIÓN de una variable, así como una EXPRESIÓN (operaciones entre variables), una INCREMENTACIÓN o DECREMENTACIÓN de una variable numérica, una FUNCIÓN BUILTIN (en este momento solo Mostrar), **una acción como AGREGAR, OBTENER o BORRAR un elemento de un conjunto (Listas, Mapas, y demás estructuras ya nombradas anteriormente). Validar a través de PRESENTE si una variable es NULL o no. Y también a través de SINO, poder agregar casos al ya existente Si.**
9. Las variables, ya sea al definirse o redefinirse, pueden adquirir valor el valor de una variable, de una expresión, del retorno de una función, o de un dato del tipo del que fueron definidas o al que pueda ser promovida.
10. Un TIPO puede ser entero, cadena de texto, **un doble, un booleano, una lista doblemente encadenada, un stack, una queue o un mapa**.
11. Las operaciones que se definen son las de aritmética básica para números, y la concatenación de dos cadenas de texto o de una cadena y un entero, que nos

parece que es algo que falta en C y que facilita la lectura del código evitando hacer uso de funciones que requieran conocimiento de punteros.

Dificultades encontradas en el desarrollo del TP

El desconocimiento tanto de las herramientas LEX y YACC como de la gramática de Cpanish fueron un gran problema cuando se inició el desarrollo de este trabajo, ya que se necesitaron gran cantidad de horas para finalmente entender cómo funcionaban. Lo que llevó al atraso en su implementación y en muchos casos, la resolución de conflictos.

Otra complicación con la que nos encontramos, fue que Cpanish no fue pensado y diseñado para en un futuro agregar nuevos tipos, funciones o estructuras. Por lo que generó que cualquier implementación nueva resulte un poco más difícil.

En cuanto a la implementación de estructuras, tuvimos que implementar funcionalidades extras a modo de validación, por ejemplo la validación de si un elemento es NULL a traves de presente, y asi como tambien, agregarle el SINO al condicional SI para evitar redundancia.

Futuras extensiones

En el futuro se espera:

- Involucrar más estructuras de datos tales como árboles, grafos, Circle Linked List, entre otras.
- Independizar aún más Cpanish++, implementado nuestras propias versiones de las librerías de las cuales depende el lenguaje (strings.h, stdlib.h, stdio.h).
- Que el código sea aún más eficiente.
- Incluir los tipos de datos float, char, long int, long double.
- Que las estructuras soporten más tipos de datos y no solo enteros en el caso de las Listas, Stacks y Queues o cadenas en el caso del Mapa.

Referencias

Las siguiente fuentes fueron consultadas:

- http://es.tldp.org/Manuales-LuCAS/FLEX/flex-es-2.5.html
- http://www.stackoverflow.com
- http://www.man7.org/
- Manual de la terminal

Repositorio Cpanish: https://github.com/atharos1/Cpanish