Salarium

Trabajo práctico especial

Autómatas, Teoria de Lenguaje y Compiladores

53774 López, Noelia Belén

53852 Mounier, Agustín

54037 Vera, Juan Sebastián

Índice

Introducción

Nuestro trabajo consiste en un lenguaje orientado a calcular los salarios de los empleados de una empresa teniendo en cuenta los cálculos más comunes que por lo general se realizan como por ejemplo las deducciones debido a impuestos, jubilación, etc. entre otros.

El lenguaje “Salarium” fue pensado para usuarios con poca experiencia en la programación. La sintaxis es intuitiva y orientada al rubro empresarial al que pertenece. Esto resulta en una fácil adopción por parte de los usuarios de dicho rubro con una curva de aprendizaje menos pronunciada que con otros lenguajes.

Desarrollo

Salarium fue desarrollado utilizando YACC y LEX. Para ello se realizaron dos archivos: salarium.y el cual contiene las reglas gramaticales del lenguaje para su utilización con YACC y salarium.l para la declaracion de los patrones de las palabras reservadas y variables para su identificación con LEX.

Al ejecutar el make, se genera un archivo llamado “parser” el cual se encargará de traducir del lenguaje Salarium a C. Para compilar código escrito en Salarium, debemos ejecutar el archivo “compile” seguido del nombre del archivo a compilar. Este último, traducirá utilizado el parser el código a C, agregara librerías del lenguaje y por último compilara el archivo en C obteniendo el ejecutable.

Implementaciones Adicionales

Cómo indica el enunciado del trabajo práctico, se realizaron 5 programas de ejemplo:

- GeneralExample.um: Muestra de manera breve y consisa, la mayoría de las implementaciones de Salarium.
- Balance.um: Se encarga de mostrar cuánto cobraría un empleado en un plazo fijo (de no ser modificado su sueldo), y cuánto sería su sueldo sin reducciones impositivas.
- RandomHeadcount.um: Crea una planta de 50 empleados azarosamente.
- RaiseSalary.um: Se encarga de aumentarles el sueldo a todos los empleados, como bono navideño, y de aumentarle la categoría a ciertos empleados.
- Salaries.um: Muestra lo que, como empleador, se gasta en los sueldos de un grupo de empleados.

Sintaxis

Variables

Dado que Salarium es un lenguaje tipado, una variable está formada por un tipo, un nombre y un valor. Esta puede ser inicializada en la declaración o luego en el código. Salarium posee un tipo especial para la representación de empleados, Employee, el cual debe ser inicializado cuando se declara.

Empleado

Un empleado es un objeto que está compuesto por:

- Nombre
- Apellido
- Categoría
- ID
- Antiguedad
- Salario

Ejemplo de creación de un empleado:

Employee nombre_variable_empleado = name: nombre_empleado, lastname: apellido_empleado, category: categoria_empleado, id: numero_empleado, antiquity: antiguedad_empleado, salary: salario_empleado;

Se deben reemplazar los textos en violeta por los valores correspondientes de cada empleado. Un ejemplo podría ser:

Employee e5 = name: Pedro, lastname: Gimenez, category: Junior, id: 53857, antiquity: 1, salary: 1500;

Arreglos

Un arreglo es un colección de elementos. La declaracion de los mismos es igual que en el lenguaje C a excepción de los arreglos de Employee's. Para la declaracion de estos últimos se requiere especificar la cantidad de elementos entre corchetes junto con los elementos (Employee's) separados por comas. A continuación un ejemplo de declaración.

EJEMPLO:
Employee employees[5] = e1, e2, e3, e4, e5;

Built In Functions

salaryFor

SalaryFor es la función que calcula el salario bruto que va a ganar un empleado en determinado lapso de tiempo que puede estar dado por semanas, meses y/o años.

EJEMPLO:
int salary = e1 salaryFor 3 month;

EXPLICACION:
Se guarda en la variable salary el importe del salario que ganará el empleado e1 los siguientes tres meses.

salaryFor MinusDeductions

SalaryFor minusDeductions es la función que calcula el salario neto que va a ganar un empleado. Se utiliza en conjunto con la funcion salaryFor explicada anteriormente.

EJEMPLO:
```
int salaryMinusDeductions = e1 salaryFor 4 month minusDeductions {
        tax1 = 100,
        retirement = (e1.salary * 0.03)
};
```

EXPLICACIÓN:
Se guarda en la variable salaryMinusDeductions el salario neto que recibirá el empleado e1 los siguientes 4 meses restándole 100 de impuestos y el 3% del salario que se debe a los aportes de jubilación.

showEmployee

Esta función imprime por consola la información de determinado empleado.

EJEMPLO:
showEmployee e1;

RESULTADO:
Empleado: [53853] Gimenez, Pedro.
Trabaja hace 1 meses, cobrando $1500.
Categoria: Junior

getEmployee

La función getEmployee imprime la información de determinado empleado contenido en un arreglo. Para ello, se especifica al empleado a través de su ID.

EJEMPLO:
getEmployee 53853 employees;

printAll

La función printAll imprime la información de todos los empleados de un arreglo.
EJEMPLO:
printAll employees;

raise10

La función raise10 aumenta el sueldo del empleado en un 10%.

EJEMPLO:

raise 10 empleado;

raise20

Análogo al raise10 con la diferencia de que aumenta un 20% el sueldo.

raiseCategory

La función raiseCategory aumenta de categoría al empleado.

EJEMPLO:

raise category empleado;

EXPLICACIÓN

Aumenta la categoría del empleado, de Junior a Senior y de Senior a CEO y la imprime. Si la categoría del empleado ya era CEO, informa que ya se encuentra en la máxima categoría, si la categoría del empleado es otra, asume que está mal definida y lo pasa a Junior.

RESULTADO

- El empleado ahora es Senior
- El empleado ya se encuentra en su máxima categoría
- La categoría del empleado estaba mal definida
  El empleado ahora es Junior

iterate

La función iterate, a la que se llama con el comando forEach, simula un iterador del arreglo de empleados.

EJEMPLO:

forEach in employees do {raise 20;};

EXPLICACIÓN:

Cuando se traduce, la función forEach llama a iterate y le pasa como parámetros el arreglo (employees) un puntero a una función que reciba un Employee (raise20), y el tamaño del arreglo, que se encuentra 'hardcodeado' dentro de la traducción. Ejecuta la función pasada por parámetro a cada empleado del arreglo (que es pasado por parámetro de dicha función)

Gramática

REFERENCIAS

- No terminales
- Terminales Variables
- Terminales
- %empty

Program	->	Functions Main
	|	Main
CallFunction	->	VAR Parameters
Functions	->	Functions Function
	|	Function
Function	->	TYPE VAR Parameters CodeBlock
	|	VOID VAR Parameters CodeBlock
	|	EMPLOYEE [ ] VAR Parameters CodeBlock
	|	EMPLOYEE VAR Parameters CodeBlock
Parameters	->	(FirstVariable)
Firstvariable	->	Variable CommaVariable
	|	%empty
Commavariable	->	CommaVariable , Variable

| %empty

Variable -> TYPE VAR

| DigOrVar

| EMPLOYEE VAR

| EMPLOYEE [] VAR

CodeBlock -> [Statement]

Statement -> Variable ; Statement

| Exp ; Statement

| SpecialFunction ; Statement

| Variable = SpecialFunction ; Statement

| Variable = Exp ; Statement

| VAR = Exp ; Statement

| VAR DigOrVar = Exp ; Statement

| EMPLOYEE VAR = SpecialFunction ; Statement

| EMPLOYEE VAR [DigOrVar] = EmpValues ; Statement

| EMPLOYEE VAR [DigOrVar] = CallFunction ; Statement

| EMPLOYEE VAR = CallFunction ; Statement

| EMPLOYEE VAR [DigOrVar] ; Statement

| WHILE (Exp) CodeBlock Statement

| IF (Exp) CodeBlock Statement

| IF (Exp) CodeBlock Statement ELSE CodeBlock

| DO CodeBlock WHILE (Exp) ; Statement

| FOREACH IN VAR DO [FunctionOverEmployee] ; Statement

| RETURN Exp

```
                |       BREAK ;
                |       %empty
                |       GET_NAME VarOrFunc ; Statement
                |       RAISE TEN VarOrFunc ; Statement
                |       RAISE TWENTY VarOrFunc ; Statement
                |       RAISE CATEGORY_VAR VarOrFunc ; Statement

SpecialFunction ->      VAR SALARYFOR DIGITO TimeLapse
                |       VAR SALARYFOR DIGITO TimeLapse MINUSDEDUCTIONS
                        [ Deductions ]
                |       SHOW_EMPLOYEE VAR
                |       PRINT_ALL VarOrFunc
                |       ReturnEmployee

VarOrFunc       ->      VAR
                |       CallFunction

ReturnEmployee  ->      NAME VAR , LASTNAME VAR , CATEGORY VAR , ID DigOrVar ,
                        ANTIQUITY DigOrVar , SALARY DigOrVar
                |       GET_EMPLOYEE DigOrVar VAR

DigOrVar        ->      DIGITO
                |       VAR

FunctionOver    ->      GET_NAME ;
Employee        |       RAISE TEN
                |       RAISE TWENTY
```

| RAISE CATEGORY_VAR ;

| SHOW_EMPLOYEE VAR

Deductions -> VAR = AddExp , Deductions

| VAR = AddExp

EmpValues -> VAR , EmpValues

| VAR

TimeLapse -> WEEK

| MONTH

| YEAR

Exp -> CondOrExp

MultExp -> Term

| MultExp * Term

| MultExp / Term

| MultExp % Term

AddExp -> MultExp

| AddExp + MultExp

| AddExp - MultExp

RelExp -> AddExp

```
                |       RelExp < AddExp
                |       RelExp > AddExp
                |       RelExp <= AddExp
                |       RelExp >= AddExp

EqExp           ->      RelExp
                |       EqExp == RelExp
                |       EqExp != RelExp

CondAndExp      ->      EqExp
                |       CondAndExp && EqExp

CondOrExp       ->      CondAndExp
                |       CondOrExp || CondAndExp

FirstVal        ->      Exp CommaVal
                |       %empty

CommaVal        ->      CommaVal , Exp
                |       %empty

Term            ->      VAR
                |       DIGITO
                |       TRUE
                |       FALSE_T
                |       STRING
```

| NULL_T

| VAR . VAR

| VAR [VAR] . VAR

| VAR [DIGITO] . VAR

| VAR (FirstVal)

| (Exp)

Main -> TYPE MAIN (VOID) CodeBlock

A futuro

Salarium posee una infinidad de posibles implementaciones ya que su rubro es muy extenso y lleno de cálculos matemáticos complejos que se podrian agregar al lenguaje de forma muy fácil.

Un posible agregado al lenguaje sería el soporte para el cálculo total de impuestos, salarios etc. que invierte la empresa. Si bien esto se puede realizar con las herramientas que posee el lenguaje actualmente, al ser un cálculo común en una empresa no estaria de mas que este integrado al lenguaje en sí.

A su vez, pensamos en que sería útil poder incorporar alguna manera de guardar variables globales para tener mejor y más rápido acceso a la información, como por ejemplo el arreglo de empleados o incluso la cantidad de empleados con los que se cuenta. Esto no sólo facilitaría el uso de las funciones ya implementadas, sino que permitiría a alguien trabajar con información de más de una empresa, haciendo el lenguaje más que interesante para contadores y/o escribanos.

Mismo, incluso, las funciones implementadas de manera manual en los ejercicios de muestra, se podrían incorporar a la librería de funciones del lenguaje.

Bibiliografía

- http://epaperpress.com/lexandyacc/download/LexAndYaccTutorial.pdf
- http://ds9a.nl/lex-yacc/cvs/lex-yacc-howto.html