Proyecto Especial de Diseño de Compiladores COVID19 AD20: En Pareja

Lenguaje Par++

A continuación, se describen las características generales del lenguaje imperativo que se deberá desarrollar durante el curso.

Hay 2 opciones para quienes decidan trabajar en parejas.
Opción 1 → Un lenguaje de input gráfico, donde un joven pueda plasmar su algoritmo a través de Diagramas de Flujo. El lenguaje debe proveer un ambiente de "dibujo de objetos" (tipo DRAW), hacer una "traducción interna" para generar el archivo que será finalmente compliado.
Opción 2 → Generar un ambien que permita crear, editar, compilar y ejecutar programas en un dispositivo móvil. Ya sea bajo iOS ó Android.

En ambos casos, la gramática con la que se trabajará sea, más o menos la siguiente:

La estructura general de un programa escrito en Par++ es:

Program Nombre_prog ;
<Declaración de Variables Globales>
<Definición de Funciones> **%%** Sólo hay funciones

%% Procedimiento Principal .... comentario
main()
{
<Estatutos>
}

*** Las secciones en itálicas son opcionales (pudiera o no venir).**
*** Las palabras y símbolos en bold son Reservadas y el %% indica comentario.**

Para la <u>Declaración de Variables</u>: *(hay globales y locales)*
sintaxis:

var %%Palabra reservada
tipo **:** lista_ids**;**
<*tipo : lista_ids;* **>** *etc...*

donde
tipo =(solo tiene) **int, float y char**.
lista_ids = identificadores separados por comas,
pudieran tener un máximo dos dimensiones [N] [M] de 0 a N-1 y 0 a M-1.
Ej: **int** : id1, id2[4], id3[2][3];
con lo que se define una variable entera (id1), un vector de 4 enteros (id2[0]... id[3]) y una matriz de 6 enteros (id3[0][0] ... id3[1][2]). %%la declaración de la dimensión siempre es una CTE-Entera

Para la <u>Declaración de Funciones</u>: *(se pueden definir 0 ó más funciones)*
sintaxis:
<tipo-retorno> **module** nombre_módulo **(** *<Parámetros>* **)** ;
<Declaración de Variables Locales>
{
<Estatutos> ***%% El lenguaje soporta llamadas recursivas*.**
}
Los parámetros siguen la sintaxis de la declaración de <u>variables simples</u> y únicamente son de entrada.
tipo-retorno puede ser de cualquier tipo soportado o bien void (si no regresa valor)

Para los <u>Estatutos</u>:

La sintaxis básica de cada uno de los estatutos en el lenguaje Par++ es:

ASIGNACION

Id<*dimensiones*> **=** Expresión**;**

A un identificador (que pudiera ser simple o ser una casilla de un elemento dimensionado) se le asigna el valor de una expresión. Cabe aclarar que siempre, a excepción de en la declaración, las Dimensiones son Expresiones aritméticas.

Id<*dimensiones*> **=** Nombre_Módulo**((**<*param1*>*,* (<*param2*>*,…*)**);** %%siempre los parámetros actuales son Expresiones

A un identificador, se le asigna el valor que regresa una función.

O bien, pudiera ser algo como: Id<*dimensiones*> = Nombre_Módulo(<*param1*>,..) + Id<*dimensiones*> – cte

A un identificador se le puede asignar el resultado de una expresión en donde se invoca a una función.

LLAMADA A UNA FUNCIÓN VOID

Nombre_Módulo (<*param1*>,..**);**

Se manda llamar una función que no regresa valor (caso de funciones *void*).

RETORNO DE UNA FUNCIÓN

return(exp **)** %%Este estatuto va dentro de las funciones e indica el valor de retorno (si no es void)

LECTURA

read (id<*dimensiones*> *, id*<*dimensiones*> >....**);**

Se puede leer uno ó más identificadores (con o sin dimensiones) separados por comas.

ESCRITURA

write ("letrero" ó expresión<*, "letrero" ó expresión*>....**);**

Se pueden escribir letreros y/ó resultados de expresiones separadas por comas.

ESTATUTO DE DECISION (puede o no venir un “sino”)

if (expresión**) then** **%%** típica decisión doble
{ <*Estatutos*>*;* **}**
<***else***
{ <*Estatutos*>*; }*>

ESTATUTOS DE REPETICION

CONDICIONAL

while (expresión**) do** **%%** Repite los estatutos mientras la expresión sea verdadera
{ <*Estatutos*>*;* **}**

NO-CONDICIONAL

for Id<*dimensiones*>**=** exp **to** exp **do**
{ <*Estatutos*>*;* **}** **%%** Repite desde N hasta M brincando de 1 en 1

EXPRESIONES

Las expresiones en Par++ son las tradicionales (como en C y en Java). Existen los operadores aritméticos, lógicos y relacionales: **+, - , *, / , &**(and) **, |** (or)**, < , >, ==,** etc**.** Se manejan las prioridades tradicionales, se pueden emplear paréntesis para alterarla.

En Par++ existen identificadores, palabras reservadas, constantes enteras, constantes flotantes, constantes char y constantes string (letreros).

%%Se anexa ejemplo (de la parte de la Gramática del luenguaje)

```
program Ejemplo;
var
   int i, j, p;
   int  Arreglo[10] ;
   float  valor;
   int  Matriz[3][8] ;

 int module fact (int j)
var int i;
   { i= j + (p - j*2+j) ;
  if ( j == 1) then
       { return ( j ); }
   else
       { return ( j * fact( j-1); }
 }

 void module inicia (int y)
var int x;
 { x= 0;
  while ( x < 11) do
     {Arreglo[x] = y * x;
      x = x+1;}
 }

main ( )
{   read (p) ;  j =p *2;
    inicia ( p * j  -  5) ;
    for i=0 to 9 do
            { Arreglo [ i ] = Arreglo [ i ] * fact (Arreglo [ i ] - p) ; }
    for j=0 to 2 do
            for k= 0 to 7 do
              { Matriz[ j , k ] = Arreglo[ j+k- fact(p) + p*k ] * p + j; }  %%Probablemente NO funcione, es solo para ejemplificar :)
    while ( i >= 0) do
            {  write ("resultado" , Arreglo [ i ] ,   fact ( i +2) * valor ) ;
                i = i - 1;
            }
}
```