Proyecto Especial COVID para Diseño de Compiladores FJ21: Parejas

Lenguaje Team++

A continuación se describen las características generales del lenguaje que se deberá desarrollar. Es un lenguaje básico que soporta definición y manipulación de clases (estilo C++).

La estructura general de un programa escrito en **Team++** es:

Programa Nombre_prog ;
<Declaración de Clases>
<Declaración de Variables Globales>
<Definición de Funciones>

%% Procedimiento Principal .... comentario
principal()
{
<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 Clases</u>:
sintaxis:

Clase id **<** ***hereda*** *id* **>** **;** %%Nombre de la clase y, posible herencia
{ ***atributos*** **<** *declaración de atributos* **>** *%%sigue la sintaxis de una declaración de variables*
metodos **<** *declaración de métodos* **>** *%% sigue la sintaxis de una declaración de funciones.*
};
***** tanto los atributos, como los métodos tendrán un nivel de accesibilidad PÚBLICA.***

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

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

donde
tipo = puede ser **entero, flotante, char** y id (donde éste es el nombre de la clase).
lista_ids = identificadores separados por comas, pudiendo definir UNA o DOS dimensiones
(id[N, M] donde N y M son números enteros indexable de 0 a N-1 y 0 a M-1).
Ej: id1, id2, id3[6,2] : **entero**; %%con lo que se definen dos variables y una matriz de 6x2 enteros.
Id4, id5: **persona**; %%define 2 objetos de una clase PERSONA.

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

Para los Estatutos:

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

ASIGNACION

variable **=** Expresión**;**

****** donde ésta puede ser id ó id[exp, exp] **ó** Id.id

A un identificador (o a una casilla o a un atributo) se le asigna el valor de una expresión.

variable **=** Nombre_Función**(**(<*param1*>, (<*param2*>,…**);** %%siempre los parámetros actuales son Expresiones
ó = Nombre_Objeto.Nombre_Método**(**(<*param1*>, (<*param2*>,…**);**

A un identificador (o a una casilla o a un atributo), se le asigna el valor que regresa una función o método.

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

A un identificador (o a una casilla o a un atributo) se le puede asignar el resultado de una expresión en donde se invoca a una función libre o a un método.

LLAMADA A UNA FUNCIÓN VOID

Nombre_Función (<*param1*>,..**);**

Nombre_Objeto. Nombre_Función (<*param1*>,..**);**
Se manda llamar una función o método que no regresa valor (caso de funciones *void*).

RETORNO DE UNA FUNCIÓN

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

LECTURA

lee (variable, *variable*....**);**
Se puede leer uno ó más identificadores (o a una casilla o a un atributo) separados por comas.

ESCRITURA

escribe ("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")

si (expresión**) entonces** **%%** típica decisión doble
{ <*Estatutos*>*;* **}**
<***sino***
{ <*Estatutos*>*;* ***}***>

ESTATUTOS DE REPETICION

CONDICIONAL

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

NO-CONDICIONAL

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

EXPRESIONES

Las expresiones en **Team++** 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 **Team++** existen identificadores, palabras reservadas, constantes enteras, constantes flotantes, constantes char y constantes string (letreros).

%% Se anexa ejemplo

```
programa Team_pp;
Clase Persona
{ atributos
       edad : entero;
      nombre[30] : char;
  metodos
   entero funcion UNO ( x : entero)
     { regresa ( edad - x); }
}
variables
      i, j, p : entero;
     estudiante: Persona;

entero funcion fact (j : entero)
variables  i: entero;
   { i= j + (p - j*2+j) ;
si ( j == 1) entonces
       { regresa ( j ); }
  sino
       { regresa ( j * fact( j-1); }
 }

void funcion  pelos (y: entero)
variables x: entero;
 { leer(estudiante.edad);
   x = y;
  mientras ( x < 10) hacer
     {escribir (estudiante.UNO(x) );
       x = x + 1;
 }

principal ( )
 {   leer (p) ;  j =p *2;
    i =  fact ( p) ;
    leer( estudiante. nombre);
    pelos (p);
   desde i = 1 hasta 10
         {  escribir ("HelloWorld", alumno.nombre,  fact(alumno.edad)) ;
              }
 }
```