C MARX MARX

Adrián Gómez Lamuedra
Miguel Manzano Rodríguez

1. Explicación y requerimientos

Explicación del lenguaje

Nuestro lenguaje, “C Marx Marx”, es un lenguaje de programación imperativo de propósito general inspirado en la obra de Karl Marx y, ante todo, en los lenguajes de programación C/C++.

Contexto histórico:

- **Lenin (lenInt):** Líder revolucionario ruso y fundador del Partido Comunista, principal arquitecto de la Revolución de Octubre de 1917.
- **Trotski (trotskIf):** Revolucionario y teórico marxista ruso, Comandante del Ejército Rojo durante la Revolución Rusa, y rival de Stalin en la lucha por el poder.
- **Stalin (stalinTruct):** Dictador soviético, sucesor de Lenin, liderando la Unión Soviética durante gran parte del siglo XX.
- **Bolcheviques (boolShevik):** Facción radical del Partido Obrero Socialdemócrata Ruso liderada por Lenin, crucial en la Revolución Rusa de 1917.
- **Che Guevara (guevArray):** Guerrillero y revolucionario argentino-cubano, ícono de la lucha revolucionaria en América Latina.
- **Fidel Castro (fidElseCastro):** Líder revolucionario cubano, comandante en jefe de la Revolución Cubana de 1959.
- **Proletariado (forLetariat):** Clase social compuesta por trabajadores asalariados, según la teoría marxista
- **Rojos (red):** Término coloquial utilizado para referirse a simpatizantes de movimientos políticos socialistas, comunistas o progresistas.

Características principales de C Marx Marx:

- **Tipado estático:** Cada variable tiene un tipo de dato específico, como entero, booleano o puntero.
- **Control de flujo condicional:** Se pueden usar instrucciones como trotskIf, fidElseCastro, forLetariat y while para controlar el flujo del programa
- **Funciones:** Se pueden definir funciones para reutilizar código y mejorar la modularidad del programa.
- **Estructuras de datos:** Se pueden usar estructuras de datos como arrays y punteros para almacenar y organizar datos de forma eficiente.
- **Operadores aritméticos y lógicos:** Se pueden usar operadores como +, -, *, /, && y || para realizar operaciones matemáticas y lógicas.

Requerimientos del lenguaje

1. Identificadores y ámbitos de definición

- Declaración de variables simples (int, double, string, bool) **int, bool**
- Arrays de cualquiera de los tipos anteriores, incluidos otros arrays (o bien, permitir arrays de varias dimensiones). **array**
- Bloques anidados (que requerirá trabajar con tabla de símbolos para bloques anidados).
- Funciones (paso de parámetros por valor o por referencia).
  hacemos también punteros y registros

Opcionales: Punteros, registros, clases (sin ningún tipo de herencia), módulos, cláusulas de importación.

2. Tipos Mínimos

- Declaración explícita del tipo de las variables.
- Tipos básicos predefinidos enteros y booleanos.
- Operadores infijos, con distintas prioridades y asociatividades para estos tipos.
- Tipo array.
- Comprobación de tipos.

Opcionales: Definición de tipos de usuario.

3. Conjunto de instrucciones del lenguaje

- Instrucción de asignación incluyendo elementos de arrays, condicional con una y dos ramas, y algún tipo de bucle.
- Expresiones formadas por constantes, identificadores con y sin subíndices (para acceso a arrays), operadores infijos y llamadas a función.

Opcionales: Expresiones con punteros y nombres cualificados (notación "." en presencia de clases o registros). Instrucción case o similar con salto a cada rama en tiempo constante. , o/y métodos de clase. Instrucciones de reserva de memoria dinámica (en presencia de punteros).

4. Gestión de errores

- Indicación del tipo de error, fila y columna.
- Parar la compilación.
- Recuperación de errores (tratar de proseguir la compilación tras un error, a fin de detectar más errores)

2. Nivel Léxico

- **Identificadores de variable:** Conjunto de letras, dígitos o _, que empieza por una letra o un subrayado (_)
- **Literales enteros:** Conjunto de dígitos del 0 al 9.
- **Símbolos de operadores aritméticos:** + - * / %
- **Símbolos de operadores booleanos:** == != < <= > >= && ||
- **Símbolos unitarios:** (Negación, referencia, valor,...): + - ! # *
- **Simobolos de accesos:** (Arrays, punteros, campos de struct...): [ ] . ->
- **Símbolos estructurales:** (Paréntesis, final de línea...): ( ) { } ; := ,
- **Símbolo para función principal del programa:** main
- **Palabras reservadas para variables y estructuras:** var lenInt boolShevik guevArray pointer stalinTruct
- **Palabras reservadas para constantes:** const true false null
- **Palabras reservadas para instrucciones:** fun return trotskIf fidElseCastro while forLetariat red write call new delete

3. Sintaxis

Identificadores y declaraciones

- Inicio del programa: cualquier código escrito en C Marx Marx tendrá una función especial “main” que contendrá el código del programa. Se podrán declarar antes de ella cualquier función, variable, struct o constante, que si se declaran fuera de cualquier función serán globales. La función main tendrá el siguiente formato:
    - **fun lenInt main(arg1, arg2, ...) {**
        ... cuerpoBloque ...
        return expresionArit;
      }

- Declaración de variables: se utiliza la palabra reservada *var* junto al tipo de la variable y su nombre.
    - **var tipo name;**

- Inicialización de variables: se especificará una expresión que, al evaluarse, tomará el valor de la variable. La variable puede estar inicializada previamente o no.
    - **var tipo name := expresion;**
    - **name := expresion;**

- Declaración de funciones: se utilizará la palabra reservada *fun* junto al tipo de la función y su nombre, para después especificar los argumentos de la función y, finalmente, el tipo de la expresión devuelta por la función.
    - **fun tipo nombre(arg1, arg2, ...) {**
        ... cuerpoBloque ...
        return expresion;
      }
    - Argumentos pasados por valor arg1, arg2, ...: **tipo name**
    - Argumentos pasados por referencia arg1, arg2, ...: **tipo # name**
    - Las funciones no pueden llamar a otras funciones pero sí puede haber anidación de la propia función; es decir, puede haber funciones recursivas.

- Declaración de funciones void: son un caso especial de la declaración de funciones, que no devolverán ninguna expresión.
    - **fun void nombre(arg1, arg2, ...) {**
        ... cuerpoBloque ...
      }

- Declaración de registros (structs): dentro de las llaves aparecerán las declaraciones de variables del registro.
    - **stalinTruct name { otras declaraciones de variables }**

Tipos de variables

- Enteros: **lenInt**
    - Números enteros
    - Ej. Declaración: **var lenInt x := 3;**
    - Ej. Uso: **x := x + 3**

- Booleano: **boolShevik**
    - Valores verdadero o falso
    - Ej. Declaración: **var boolShevik b := false**
    - Ej. Uso: **if b {x := x + 3}**

- Vectores: **guevArray tipo[tamaño]**
    - Conjunto de elementos de un mismo tipo con un tamaño predeterminado. El tamaño se especifica al declarar el array como una expresión aritmética.
    - Ej. Declaración: **guevArray lenInt v[x+5];**
    - Ej. Uso: **v[i] := x**

- Puntero: **pointer tipo**
    - Direcciones de memoria de otras variables.
    - Ej. Declaración: **var pointer lenInt x := #y;** (pasamos la referencia de **y**)
    - Ej. Uso: **z := *x;** (pasamos el valor al que apunta el valor **x**)

Instrucciones

- Instrucción asignación: **objeto := expresion**
    - El “objeto” ha de ser una variable, un acceso a un array o un campo de struct. Por otro lado, la expresión dará como resultado un entero, booleano o puntero.{
    - Ej. Uso: **var lenInt x := 3; v[i] := false; tFecha.cuenta := 100;**

- Instrucción if: **trotskIf (expresiónBooleana) { Bloque }**
    - Ej. Uso: **if (x > 3 && comuna == true) {**
      x := x - 1;
      comuna := false;

```
}
```

- Instrucción if-else: **trotskIf (expresiónBooleana) { Bloque }**
  fidElseCastro (expresión Booleana) { Bloque }
  - Ej. Uso:
```
if (x > 3 && comuna == true) {
    x := x - 1;
    comuna := false;
}
else { x:= 0}
```

- Instrucción while: **while (expresiónBooleana) { Bloque }**
  - Ej. Uso:
```
while (x > 3 && comuna == true) {
    x := x - 1;
}
```

- Instrucción for: **forLetariat (var lenInt nombre := expresion; expresiónBooleana; nombre := expresion) { Bloque }**
  - Ej. Uso:
```
forLetariat (var lenInt i := 0; i < N; i := i + 1) {
    v[i] := i + 3;
}
```

- Instrucción write (para salida de datos por consola): **write (expresiónAritmetica)**
  - Ej. Uso: **var lenInt x; red x; x := x + 3; write (x * 3 - 5)**

- Instrucción delete (para liberar memoria): **delete expresion**
  - Siempre se ha de usar delete después de reservar memoria para el objeto con new.
  - Ej. Uso: **var lenInt *p = new lenInt; delete p;**

- Instrucción call (función tipo void): **call identificadorFuncion(parametros);**
  - El identificador es el nombre con el que se ha declarado la función void previamente y los parámetros una lista de expresiones.
  - Ej. Uso: **call suma(x, y, resul);**

Expresiones

En cuanto a las expresiones, encontramos la posibilidad de realizar uniones de dichas expresiones, utilizando los operadores ya definidos. En general, se especifican los tipos de expresiones y su prioridad. En lenguaje encontraremos:

- Expresiones básicas: Enteros, booleanos, identificadores y **null**

- Expresiones compuestas: Tendremos expresiones aritméticas (**exArit**) que se tratarán de la misma forma que un tipo entero, y expresiones booleanas (**exBool**), que devolverán **true** o **false**. Estas expresiones se construyen de acuerdo con las siguientes prioridades y asociatividades de los operadores contemplados (0 es el nivel de menor prioridad).

Nivel 0

- Conjunción: **exBool || exBool**

Nivel 1

- Disyunción: **exBool && exBool**

Nivel 2

- Igual: **exArit == exArit**
- Distinto: **exArit != exArit**
- Menor o igual: **exArit <= exArit**
- Mayor o igual: **exArit >= exArit**
- Menor: **exArit < exArit**
- Mayor: **exArit > exArit**

Nivel 3

- Suma: **exArit + exArit**
- Resta: **exArit - exArit**

Nivel 4

- Multiplicación: **exArit * exArit**
- División entera: **exArit / exArit**
- Módulo: **exArit % exArit**

Nivel 5

- Entero positivo: **+ exArit**
- Entero negativo: **- exArit**
- Negación: **! exBool**
- Referencia: **# id**
- Valor: *** id**

Nivel 6

- Expresión new: **new tipo** Ej.Uso: **var lenInt *p = new lenInt;**
- Expresión read: **red tipo** Ej.Uso: **var lenInt p = red lenInt;**

Nivel 7

- Acceso a un array: **idArray[exArit]** Ej: **v[i+3]**
- Acceso a un campo de un struct: **idStruct.idCampo** Ej: **fecha.mes**

- Acceso a un campo de un struct apuntado por variable puntero: **idStruct->idCampo** Ej: **fecha->mes**

Nivel 8

- Expresión con paréntesis: **(expresion)**

Los niveles de prioridad de las expresiones son algo orientativo y serán reajustados a lo largo del proyecto.

Errores

En cuanto a este apartado, no encontramos ninguna especificación sintáctica, pues los errores aparecerán en el *output* en respuesta a cierto fallo en nuestro código.

4. Explicación y ejemplo de código

Ejemplo de código

```
// Declaración de tipos de datos
stalinTruct tFecha {
  var lenInt dia;
  var lenInt mes;
  var lenInt ano;
}

// Declaración de variables
var lenInt x := 3;
var boolShevik b := true;
var guevArray lenInt v[TAM_ARRAY];
var pointer lenInt p := #x; // Puntero a la variable x
var tFecha fecha := {1, 1, 2024};

// Declaración de funciones
fun lenInt suma(lenInt a, lenInt b) {
        return a + b;
}
```

```
fun lenInt max(lenInt a, lenInt b) {
// Ejemplo de trotskIf
        trotskIf (a > b) {
                return a;
        } fidElseCastro {
                return b;
        }
}

fun void imprimirFecha(tFecha #fecha) {
        write (fecha.dia);
        write(fecha.mes);
        write(fecha.ano);
}
// Asignación de valores al array
forLetariat (var lenInt i := 0; i < TAM_ARRAY; i := i + 1) {
        v[i] := i * 2;
}

// Bucle while que imprime los valores del array
while (x < TAM_ARRAY) {
        write(v[x]);
        x := x + 1;
}

// Llamada a la función suma e impresión del resultado
var lenInt resultado := suma(x, 5);
write(resultado);

// Modificación del valor de la variable a través del puntero
*p := 10;

// Impresión de la fecha actual
call imprimirFecha(fecha);

// Liberación de memoria del array
delete v;

// Fin del programa
```