PRIMERA ENTREGA DEL PROYECTO DE AULA

INTEGRANTES

ANDRÉS FELIPE ARISTIZABAL VELASQUEZ

JUAN DAVID AGUIRRE CÓRDOBA

PROFESOR

GABRIEL TABORDA BLANDON

INSTITUTO TECNOLOGICO METROPOLITANO

FACULTAD DE INGENIERIA

INGENIERIA DE SISTEMAS

MEDELLIN

2021

Tabla de contenido.

¿Qué es un lenguaje de programación? 4

Tipos básicos de lenguajes de programación por generaciones. 5

Nivel Alto 5

Nivel Intermedio 5

Nivel Bajo 6

Componentes básicos de un lenguaje de programación. 7

Definición. 7

Tipos y estructuras de datos. 7

Instrucciones. 7

Control de flujo. 7

Funciones y objetos. 7

Fases de la compilación. 8

DEFINICIÓN DE UN LENGUAJE DE PROGRAMACIÓN PROPIO 9

Introducción o descripción del lenguaje 9

Tipos básicos de datos o variables 9

Especificación para nombre identificadores 9

Declaración de variables 10

Declaración de Métodos 10

Parámetros en un método. 11

Retorno de valores en un método. 11

Operadores 12

Constantes de cadena o cadena de caracteres 12

Comentarios 13

Instrucción de asignación 13

Instrucciones de entrada y salida 14

Instrucción de entrada 14

Instrucción de salida 14

Instrucciones de decisión 14

Instrucciones de repetición o ciclos 16

Delimitadores de programa 16

Línea de instrucciones 16

Bloque de instrucciones 16

Clases y métodos. 16

3 ejercicios o programas sencillos con su lenguaje de programación (escrito): una para operaciones aritméticas, otro para instrucciones de decisión y uno para ciclo. 17

CONCLUSIONES PERSONALES 19

Referencias. 20

TEORÍA

¿Qué es un lenguaje de programación?

El lenguaje de programación es un sistema estructurado de comunicación, el cual está conformado por conjuntos de símbolos, palabras claves, reglas semánticas y sintácticas que permiten el entendimiento entre un programador y una máquina.

Al proceso por el cual se escribe, se prueba, se depura, se compila y se mantiene el código fuente de un programa informático se le llama programación.

Es un lenguaje formal y artificial que, mediante una serie de instrucciones, le permite a un programador escribir un conjunto de órdenes, acciones consecutivas, datos y algoritmos para, de esa forma, crear programas que controlen el comportamiento físico y lógico de una máquina.

Mediante este lenguaje se comunican el programador y la máquina, permitiendo especificar, de forma precisa, aspectos como:

- Cuáles datos debe operar un software específico.
- Cómo deben ser almacenados o transmitidos esos datos.
- Las acciones que debe tomar el software dependiendo de las circunstancias variables.

Existe un error común que trata por sinónimos los términos 'lenguaje de programación' y 'lenguaje informático'. Los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como por ejemplo el HTML. (Lenguaje para el marcado de páginas web que no es propiamente un lenguaje de programación sino un conjunto de instrucciones que permiten diseñar el contenido y el texto de los documentos).

Tipos básicos de lenguajes de programación por generaciones.

Nivel Alto

Tienen como objetivo facilitar el trabajo del programador, ya que utilizan unas instrucciones más fáciles de entender.

Además, el lenguaje de alto nivel permite escribir códigos mediante idiomas que conocemos (español, inglés, etc.) y luego, para ser ejecutados, se traduce al lenguaje de máquina mediante traductores o compiladores.

1. Traductor

Traducen programas escritos en un lenguaje de programación al lenguaje máquina de la computadora y a medida que va siendo traducida, se ejecuta.

2. Compilador

Permite traducir todo un programa de una sola vez, haciendo una ejecución más rápida y puede almacenarse para usarse luego sin volver a hacer la traducción. (rockcontent, n.d.)

Nivel Intermedio

En la programación de computadoras, cuando un compilador analiza el código fuente legible por humanos, el compilador a menudo traduce los comandos fuente en una secuencia de instrucciones que no son del código de la máquina nativa pero que el compilador u otro software pueden procesar, optimizar o compilar aún más. herramientas. Los archivos que se producen contienen instrucciones que se dice que están en un idioma intermedio, porque el lenguaje utilizado por el compilador no es significativo para el sistema operativo más grande ni el lenguaje utilizado por el programador para escribir el código en primer lugar, sino es un lenguaje que actúa como un puente entre la escritura y la ejecución del programa.

A veces se usa un lenguaje intermedio para permitir que un compilador realice optimizaciones muy precisas para que el programa se ejecute de manera más eficiente, pero también se puede usar para producir archivos de salida que son portables entre diferentes sistemas incompatibles. La sintaxis real del lenguaje puede parecerse al código de máquina u otros tipos de códigos de bytes legibles no humanos, o el lenguaje puede ser solo un lenguaje de programación de computadora multiplataforma existente.

Cuando se utiliza para la optimización del compilador, un compilador de lenguaje toma cada declaración en un archivo de código fuente y divide el comando en el idioma

intermedio. Una sola declaración de programación legible por humanos puede descomponerse en docenas de instrucciones de lenguaje de máquina, por lo que el lenguaje intermedio crea un nivel de abstracción que el compilador puede usar para identificar áreas del código donde se pueden realizar optimizaciones sin tener que vincular primero el código a implementaciones nativas o bibliotecas. Una vez completado, el archivo de código intermediario puede compilarse más para crear un archivo binario nativo, o puede ejecutarse a través de otro programa, conocido como intérprete, que ejecutará el código compilándolo en instrucciones nativas según sea necesario. (Netinbag, n.d.)

Ejemplo: C y C++.

Nivel Bajo

1. Lenguaje máquina

Es el más primitivo de los lenguajes y es una colección de dígitos binarios o bits (0 y 1) que la computadora lee e interpreta y son los únicos idiomas que las computadoras entienden.

Ejemplo: 10110000 01100001

No entendemos muy bien lo que dice ¿verdad? Por eso, el lenguaje ensamblador nos permite entender mejor a qué se refiere este código.

2. Lenguaje ensamblador

El lenguaje ensamblador es el primer intento de sustitución del lenguaje de máquina por uno más cercano al utilizado por los humanos.

Un programa escrito en este lenguaje es almacenado como texto (tal como programas de alto nivel) y consiste en una serie de instrucciones que corresponden al flujo de órdenes ejecutables por un microprocesador.

Sin embargo, dichas máquinas no comprenden el lenguaje ensamblador, por lo que se debe convertir a lenguaje máquina mediante un programa llamado Ensamblador.

Este genera códigos compactos, rápidos y eficientes creados por el programador que tiene el control total de la máquina. (rockcontent, n.d.)

Ejemplo: **MOV AL, 61h** (asigna el valor hexadecimal 61 al registro “AL”)

Componentes básicos de un lenguaje de programación.

Definición.

Los lenguajes de programación no han dejado de ser un conjunto de símbolos con una estructura gramatical, reglas semánticas y de sintaxis. En este orden, los lenguajes de alto nivel han facilitado su uso al implementar un lenguaje parecido al inglés, más reducido y formal, para establecer condiciones como *if-then-else*, indicar el tipo de dato que se va a manejar, como *integer*, real, *double*, o señalar eventos como *print*. De igual manera, hay signos y operadores que ayudan a estructurar operaciones matemáticas o lógicas, como suma, resta, multiplicación (+, -, *, /), etcétera.

Tipos y estructuras de datos.

Las estructuras de datos son elementos de los lenguajes de programación que permiten manipular de forma más eficiente variables diversas: numéricas o tipo texto (cadenas de caracteres), y otras más complejas, como vectores, matrices y apuntadores, etcétera.

Instrucciones.

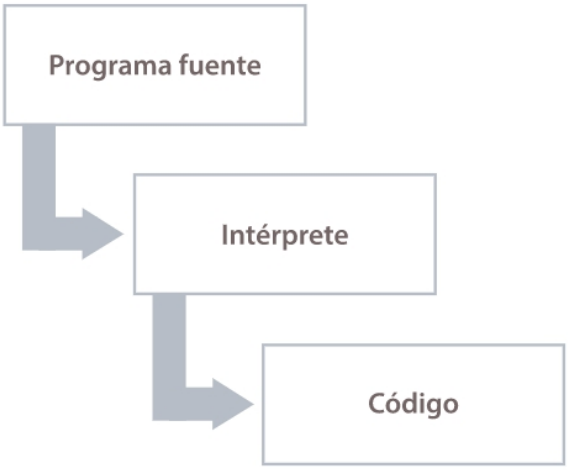
Son estructuras gramaticales predefinidas, muy parecidas al lenguaje humano, para generar secuencias de acciones que conformen un programa. Van desde los operadores aritméticos y lógicos básicos (sumas, restas, *and*, *or*) hasta instrucciones más especializadas para realizar diversas acciones dentro del programa, como guardado de archivos, volcado de pantalla de un texto, etcétera.

Control de flujo.

Se refiere a la secuencia de acciones de un programa. En ocasiones, dentro de la secuencia de instrucciones, hay puntos donde el programa debe tomar decisiones con base en el valor de una variable o el cumplimiento de una cierta condición. El tipo de instrucciones que posibilitan dichas acciones son, precisamente, las de control de flujo: condicionales (*if-then-else*), de bucle (*for o while*) o selección (*case*).

Funciones y objetos.

Con la aparición de la programación estructurada también surge el empleo de funciones: una serie de instrucciones localizadas fuera del cuerpo principal del programa que realizan una tarea específica y regresan un resultado; pueden ser empleadas a lo largo de un programa una o varias veces. Los lenguajes de alto nivel, además de las funciones predefinidas por el propio lenguaje, permiten al programador diseñar y construir sus propias funciones.



Fases de la compilación.

La compilación permite crear un programa de computadora que puede ser ejecutado por ésta y comprende tres pasos:

(UNAM, n.d.)

DEFINICIÓN DE UN LENGUAJE DE PROGRAMACIÓN PROPIO

Introducción o descripción del lenguaje

JACAF es un lenguaje de programación bastante tipado, tiene bases del lenguaje de programación Typescript y algunas adaptaciones para un mejor aprendizaje y legibilidad del código.

Tipos básicos de datos o variables

Tipos básicos de datos:

Numéricos: Pueden ser *integer* o *double*, dependiendo si es entero o real respectivamente

Alfanuméricos o **cadena de caracteres**: *string*

Lógicos: *boolean*

Especificación para nombre identificadores

Para este caso en particular la lista de las palabras reservadas no posibles a usar para la creación en nombres de clases, variables y métodos son las siguientes:

class
integer
double
string
var
for
while
do
if
else
abstract
private
public
return
static
void
switch
case

break
default
print
read

Para JACAF sólo será permitido el guion bajo como carácter especial para la creación de variables, métodos y clases. Por lo tanto, ningún otro carácter especial es permitido.

No pueden llevar espacios.

Pueden contener números, pero no al inicio del nombre de la variable.

Declaración de variables

Para la declaración de variables vamos a declarar primero el identificador único *var* que será requerido cada vez que se desee crear cualquier variable, luego viene el nombre de la variable y al final 2 puntos (:) que delimitan con el tipo de dato. Después, la asignación utilizando el signo igual (=) y para finalizar se asigna el valor y se cierra con punto y coma (;).

Ejemplo 1: var nombreVariable: string = “Juana”;

Ejemplo 2: var nombreVariable: integer = 10;

Declaración de Métodos

Para la declaración de métodos, es necesario utilizar la palabra reservada *method*, luego vendrá el nombre del método con apertura de paréntesis, luego los parámetros y cierra paréntesis, seguido a esto 2 puntos para determinar el tipo de retorno del método. En el siguiente ejemplo, se ilustra mejor la sintaxis descrita.

Todos los métodos son públicos, no es necesario agregarle la palabra reservada *public* al inicio. Caso contrario, si se quiere tener un método privado, si es necesario agregarle la palabra reservada *private*.

Ejemplo 1:

```
method printCurrentName(): void {

  var name: string = “Juan”;

  print(`Current name ${name}`);

}
```

Ejemplo 2:

```
private method printCurrentName(): void {

  var name: string = “Juan”;

  print(`Current name ${name}`);

}
```

Parámetros en un método.

Para definirle parámetros a un método, basta con poner el nombre del argumento y el tipo, tal y como se puede evidenciar a continuación:

Ejemplo:

```
method printCurrentName(name: string): void {

  print(`Current name ${name}`);

}
```

Retorno de valores en un método.

Se utilizará la palabra reservada *return*, para retornar el valor calculado o no calculado en un método. También, de sólo necesitar salir del método, se usará *return* seguido de punto y coma. Este *return* implica que retorna el tipo vacío o *void*.

Ejemplo:

```
method printCurrentName(name: string): string {

  return `Current name ${name}`;

}
```

Operadores

Para el lenguaje se van a utilizar los operadores comunes de la siguiente forma:

Operadores aritméticos:

Suma: +, Resta: - , Multiplicación: * , División: /

Operadores de asignación:

Para la relación vamos a utilizar el signo igual (=)

Operadores relacionales:

Símbolos:

menor que: <, mayor qué: >, mayor e igual: >=, menor e igual: <=,

exactamente igual: ==, diferente: !=

Operadores lógicos:

Para este caso se usará lo siguiente:

and: Referente a la proposición Y

or: Referente a la proposición O

not: Referente a la proposición de negación. Para el caso del *not* que tenga que negar una expresión se debe usar la siguiente sintaxis: *not*(variable)

Constantes de cadena o cadena de caracteres

Las cadenas de caracteres estarán definidas solamente entre comillas dobles, tal y como se evidencia en el siguiente ejemplo:

Ejemplo 1: var nombreVariable: string = “Maria”; (**Bien**)

Ejemplo 2: var nombreVariable: string = ‘Maria’; (**Error**) No se admite comilla simple.

También es posible utilizar la tilde atrasada o más conocida como backtick y con esta es posible concatenar variables de la siguiente forma.

Ejemplo:

var miTexto: string = “Hola”;

print(`Este es mi texto ${miTexto}`);

Notar que para incluir una variable dentro de una cadena de caracteres es necesario utilizar un signo pesos y 2 llaves que la contengan.

Comentarios

Los comentarios del lenguaje serán siempre (para una o varias líneas de código) en el tipo C, es decir, **/*** abre el comentario, mientras que ***/** cierra el comentario.

En caso de no cerrarse un comentario con su sintaxis (**/***) este se extenderá hasta el final de la página, partiendo desde donde se declaró (**/***), y en caso de solo encontrar el final de un comentario (***/**) sin su respectiva apertura (**/***), el compilador no lo interpretará como una secuencia valida de su sintaxis, por lo que arrojará un error.

Ejemplo:

/* Este ciclo busca la factura con mayor precio ***/**: **Comentario válido.**

// Buscamos el número de facturas: **Comentario invalido**, esta sintaxis no se usará.

Instrucción de asignación

En este caso utilizaremos la común asignación, que consiste en tomar el valor de la derecha del operador de asignación (=) y almacenarlo en el elemento de la izquierda. Se conserva el útil y simple símbolo de igualdad **=**

Ejemplo 1:

var nombreVariable: integer = 0;

nombreVariable= nombreVariable + 1;

Instrucciones de entrada y salida

Instrucción de entrada

La sintaxis general de la instrucción de entrada es:

Ejemplo: var nombreVariable: string = read();

Dentro de *read*(“Mensaje”) se puede enviar un parámetro adicional para mostrar un mensaje por consola al pedir por información. Por defecto, todo valor leído por la terminal es un valor de tipo *string.*

Se tomaría desde el teclado así: El valor a la derecha del signo igual debe ser una variable declarada con el mismo tipo de dato que retorna la función read().

Instrucción de salida

En nuestro lenguaje usaremos el método print(miVariableString) donde la cadena es lo que se imprimirá en consola así:

Ejemplo: print(“Hola mundo”);

// Se imprimirá Hola mundo en la consola.

Instrucciones de decisión

Para las instrucciones de decisión se usará:

Para la comparación: *if*, *else*. El *if* puede estar sin un *else*.

Sí: *if* (/*Expresión lógica a evaluar*/) { /*Expresión en caso de que sea verdad*/ }

Si no: *if* (/*Expresión lógica a evaluar*/) { /*Expresión en caso de que sea verdad*/ }
else { /*Expresión en caso de que no sea verdad*/ }

En caso de:

```
switch (variable) {
        case valor1:
        break;
        case valorN:
        break;
        default:
        break;
}
```

Ejemplo 1:

```
var nombreVariable: integer = 20;
if (nombreVariable >= 20) {
   print(“Soy mayor que 20”)
}
```

Ejemplo 2:

```
var nombreVariable: integer = 20;
if (nombreVariable >= 20) {
   print(“Soy mayor que 20”);
} else {
  print(“Soy menor que 20”);
}
```

Instrucciones de repetición o ciclos

Para las instrucciones de repetición se usará lo siguiente:

Ciclo Para: for(var indice: integer=0; indice<cantidad; índice+=1) { /* código a repetir*/ }

Ciclo Mientras: while(Expresión lógica a evaluar) { /* código a repetir */}

Ciclo Hacer Mientras:

do { /*código a repetir*/ } while(Expresión lógica a evaluar);

Delimitadores de programa

Básicamente, y como en general lo estamos haciendo, usaremos los delimitadores de los lenguajes tipo C, ya que son los más conocidos universalmente y fáciles de aplicar.

Llaves: Con estas enceramos bloques de código para ciclos, condicionales, entre otros como **variables de tipo objeto.**

Línea de instrucciones

Punto y coma (;): Se utilizará para decirle al compilador que una instrucción termina ahí.

Bloque de instrucciones

Llaves: Con estas encapsulamos bloques de código.

Clases y métodos.

Llaves: Con estas encapsulamos las clases y métodos.

3 ejercicios o programas sencillos con su lenguaje de programación (escrito): una para operaciones aritméticas, otro para instrucciones de decisión y uno para ciclo.

```
/* Ejercicio 1 - Operaciones aritméticas Suma: +, Resta: - , Multiplicación: * ,
División: /, Potenciación: ** */
public class EjemploOperadores {
  var variableA: integer = 3;
  var variableB: integer = 2;

  method suma(): void {
    print(variableA + variableB); /* Resultado = 5 */
  }

  method resta(): void {
    print(variableA - variableB); /* Resultado = 1 */
  }

  method multiplicacion(): void {
    print(variableA * variableB); /* Resultado = 6 */
  }

  method multiplicacion(): void {
    print(variableA / variableB); /* Resultado = 1.5 */
  }
}
```

```
/* Ejercicio 2 - Instrucción de decisión*/
public class InstruccionesDecision {
  var edad: integer = 20;

  method esMayorA20(): boolean {
    if (edad ≥ 20) {
      return true;
    }
    return false;
  }

  method validar(): string {
    if (edad ≥ 20) {
      return "Soy mayor que 20";
    } else {
      return "Soy menor que 20";
    }
  }
}
```

```
/* Ejercicio 3 - Instrucción de repetición */
public class InstruccionesRepeticion {
  var variableC: integer = 10;

  method repetir(): void {
    for(var indice: integer=0 ;indice<variableC ;indice+=1) {
      print(`Valor del índice: ${indice}`);
    }

    while(variableC ⩾ 11) {
      print(`Valor variableC: ${variableC}`);
    }

    do {
      print(`Valor variableC: ${variableC}`);
    } while(variableC ⩾ 11);
  }
}
```

CONCLUSIONES PERSONALES

- Comprender la funcionalidad, estructura y sintaxis de los lenguajes de programación.
- Identificar los diferentes tipos de lenguaje de programación y sus niveles
- Identificar los componentes de los lenguajes de programación para aplicarlos mediante de un código.
- Emplear las funciones en la programación para reducir la operatividad y la cantidad de líneas de código.
- Definir las variables y combinarlas con los operadores de forma correcta para que el programa ejecute y no reviente errores.
- Conocer los ciclos que se pueden emplear al momento de realizar una repetición para que las variables a analizar tengan sentido.

Referencias.

Netinbag. (s.f.). Recuperado el 09 de 10 de 2021, de https://www.netinbag.com/es/internet/what-is-an-intermediate-language.html

rockcontent. (s.f.). Recuperado el 09 de 10 de 2021, de https://rockcontent.com/es/blog/que-es-un-lenguaje-de-programacion/

UNAM. (s.f.). *Lenguajes de Programación*. Recuperado el 09 de 10 de 2021, de https://programas.cuaed.unam.mx/repositorio/moodle/pluginfile.php/1023/mod_resource/content/1/contenido/index.html