Curso de Programación

Orientado a videojuegos

Empezamos en minutos...

Maestro Jedi: Daniel Delgado
Duración: al menos 10 horas.
El resto depende de ti...

Pseudocódigo

Es un algoritmo estructurado de tal forma que sea más fácil llevarlo a algún lenguaje de programación luego.
[Para mi] es solo la mejor forma de practicar y desarrollar nuestra lógica, y habilidades de programación, antes de pasar a algún lenguaje en concreto.

Se comienza con un Begin y nombre del programa, aunque el nombre no es necesario.

Se termina con un End y nombre del programa, aunque el nombre del programa no es necesario.

Pseudocódigo - Reglas y convenciones.

- Cada instrucción debe llevar un punto y coma ; al final.
- Las instrucciones deben ser siempre escritas con la primera letra en mayúscula Let, Print, Begin, End.
- Dos slashes // abren un comentario. Esto es usualmente una descripción de lo que hace esa línea en el programa. Los comentarios son desechados en el momento de la compilación o interpretación de nuestro código.
- La instrucción **Print** imprime el mensaje entre comillas en la pantalla. Print “Hola Mundo”;
- Las instrucciones **Let** y **Var** inicializan una variable. Var contador = contador + 1;

Ejemplo:

Begin demo
    Let a = 10; //inicializa la variable en un número entero igual a 10
    Print “El número es ” + a; //Imprime en pantalla el mensaje “El número es ” y el valor de la variable a
End demo

Pseudocódigo - Reglas y convenciones.

- Por convención entre programadores, los nombres pueden ser tan largos como se necesite para describir exactamente lo que hace la variable. Esto facilita la lectura del código. Por ejemplo, si se tiene un contador de vida, y un contador de turbo, es más fácil leer contadorVida y contadorTurbo que si se llamaran contador1 y contador2.

- También los nombres de las variables deben tener relación con el valor u objeto que van a guardar. Por ejemplo aunque fácilmente pueden crear una variable llamada email = 100; en sentido práctico no servirá, y se podría prestar a confusión por ti mismo y otros programadores de tu equipo.

- Así mismo todas las variables deberían comenzar en minúsculas, y si tienen más palabras, la inicial de cada palabra siguiente en mayúsculas. Ejemplo: timeRespawnEnemy, orbCooldownDuration. Este estilo de escritura de nombres, se llama lowerCamelCase. El caso de todas las iniciales en mayúsculas se llama UpperCamelCase y se usa para definir nombres de clases, lo cual veremos más adelanto.

- Por regla general, ninguna variable puede ser una instrucción o palabra reservada del código, así una variable Var Print = 0; daría error, ya que el nombre de la variable no puede ser la palabra Print por ser una instrucción de programación. Esto aplicará para todas las instrucciones y palabras reservadas que iremos viendo en este curso, y las que puedan descubrir en lenguajes de programación en el futuro.

- Las variables no pueden tener espacios. Si se necesita usar un espacio se usa un separador como underscore (_) Ejemplo: enemy_life, kills_counter, objective_time son nombres de variables correctos. No se pueden usar separados como punto (.) o coma (,) punto y coma (;) corchetes ([ ]), llaves ({}) y ningún operador matemático (+, /, *, -,%,mod,etc) o simbolos como <, >, ?, &,^,%,$,#,@. Solo usen underscore para separar palabras o mejor no separen palabras en el nombre de variables.

- Dado que no todos los lenguajes lo aceptan, por convención ninguna variable puede comenzar por carácter especial, o número, salvo underscore (_)

Pseudocódigo - Reglas y convenciones.

- Mantener el idioma español, inglés o cualquier otro al momento de programar, durante todo el programa. Si tus variables, clases y funciones están en español, debes mantenerlo así para todo. No usar combinaciones de idiomas.
- Así mismo, no debemos usar caracteres en español, como la ñ, o letras con acentos (á, é, etc) o dieresis (ü), no muchos lenguajes los aceptan, y por regla no se usan.
- Cada bloque de código debe ser indentado una serie de espacio o un tab a la derecha. Esto no es obligatorio, pero facilita la lectura de un código extenso.
  Por ejemplo, el pseudocódigo indentado de un programa que imprima la tabla de multiplicar del 7:

```
Begin sample //inicio del programa sample
➜       Var mainNumber = 7;                     //el número del que se quiere imprimir la tabla de multiplicar
➜       Print “Tabla de multiplicación del 7”; //título en pantalla de la tabla
➜       For multipNumber = 1 to 10 do           //Ciclo For del 1 al 10
        ➜       Print mainNumber + “ x ” + multipNumber + “ = ” + mainNumber * multipNumber;
➜       EndFor                                  //fin del ciclo for
End sample                                      //fin del programa sample
```

Tipos de datos

Definir a la computadora qué tipo de contenido tiene una variable o una constante.

Por ahora, vamos a ver variables de tipo Number, String, y Booleans. Hay otros tipos de variables, float para números decimales, integer para números enteros, date para fecha, etc. Por ahora haremos ejercicios con estos 3 tipos.

Number: variables que van a guardar cualquier tipo de número, o cualquier resultado de operación matemática.
Ej: Var Number: vida = 100;

String: variables que van a contener una cadena de caracteres, oraciones y mensajes en el programa.
Ej: Var String: msgMoira10 = “Surrender to my will!”;

Booleans: variables lógicas que van a tener dos valores solamente, 1 o 0, o true o false.
Ej: Var Boolean: zoneOneActive = false; //nota que false es una palabra reservada del lenguaje y no puede ser usada como nombre de funciones, variables, clases, etc.

Print, Input

Print sirve para saber cuando un programa debe mostrar algo por el hardware de salida definido para el proyecto.
Print “Hola, mundo!”; //imprime Hola, mundo! En pantalla.

Si se usa con el símbolo +, te permite concatenar, unir valores de variables o constantes con el contenido de la cadena.

```
Begin
        Var String: mensaje1 = “- Hola, mundo”;
        Var String: mensaje2 = “- Saluda!”;
        Print mensaje2 + “ ” + mensaje1; //Imprime -Saluda! -Hola, Mundo

End
```

Input sirve para decirle a la computadora que pida un dato al usuario, para procesarlos.

```
Begin
        Var Number: edad = 0;
        Print “Escribe tu edad”; //escribe un mensaje en pantalla para decirle al usuario lo que se necesita.
        Input edad; //Lee el valor escrito y lo almacena en la variable edad.
        Print “No importa lo que te digan, para mi siempre serás un bebé!”; //mensaje positivo...
End
```

Operadores Matemáticos

Suma +

Resta -

Multiplicación *

División /

Entero de la división \ (Div)

Potencia ^

Módulo Mod

Raíz Cuadrada SQRT()

Operadores de Comparación y Lógicos

Igual que = ==

Menor que <

Mayor que >

Menor o igual que <=

Mayor o igual que >=

Distinto que <> !=

No se cumple que (Not) !

Y se cumple que (And) &&

O se cumple que (Or) ||

Prioridad de operadores

Igual que en matemática aritmética hay prioridad de símbolos a resolver primero.

En programación no se usan corchetes [ ] o llaves { } para indicar prioridad de operaciones, sólo se usan paréntesis.
Ej:
Var Number: operacion = 10 * ((2 + 1) / (100 / 50) * 3);

La computadora resolverá primero (2 + 1) = 3, luego (100 / 50) = 2, luego 3 / 2 * 3, primero la división ya que está a la izquierda, 3 / 2 = 1.5 * 3 = 4.5, y por último 10 * 4.5 = 45 con lo que la variable operación será igual a 45.

La multiplicación siempre deberá ser indicada por el asterisco *, los lenguajes no saben qué hacer con una expresión como A B y se necesita escribir siempre el símbolo de la multiplicación: A * B.

Igual que en matemáticas se resuelven primero la operación de más peso, a la izquierda. Así A / B * C resolverá primero A entre B y luego multiplicará por C.

La prioridad de operadores es:

1. **Paréntesis ( )**
2. **Potencias ^**
3. **Multiplicación, División * /**
4. **Sumas, Restas + -**
5. **Resto, Cocientes \ (Div) Mod**
6. **Operadores de comparación = < > <= >= <>**
7. **No ! (Not)**
8. **Y && (And)**
9. **O || (Or)**

Realicemos un algoritmo lineal complejo...

De un número de 4 cifras sacar la cifra que representa la unidad del número, la decena, la centena y la unidad de mil, o unidad de millar.
Dato de entrada: un número de 4 cifras.
Procesos: calcular unidad de mil, centenas, decenas y la unidad.
Datos de salida: unidad de mil, centenas, decenas y la unidad del número dado.

Tenemos el número 1.043. Fácilmente, con nuestros ojos podemos decir que la unidad de mil está representada en el 1. Luego pasamos la vista a las centenas que son el 0. Las decenas están representadas en el 4 y por último la unidad es el 3.
Ahora con lógica de aritmética de programación hagamos el algoritmo que “vea” al número y lo divida en esas partes, sin error.
Lo que el 1 quiere decir es que está multiplicando al 1000
Lo que el 0 quiere decir es que está multiplicando al 100
Lo que el 4 quiere decir es que está multiplicando al 10
Y lo que el 3 quiere decir es que está multiplicando al 1
El número 1043 se puede escribir como 1043 = 1 x 1000 + 0 x 100 + 4 x 10 + 3 x 1
El número 6258 se puede escribir como 6258 = 6 x 1000 + 2 x 100 + 5 x 10 + 8 y así todos los números de 4 cifras.

Saquemos primero la unidad de mil del número. Debería dar 1. **UnidadMillar = 1043 \ 1000. (\ nos da el entero de una división) = 1**

Para sacar las centenas debemos deshacernos de la unidad de mil. Este número puede ser un número entre 1 y 9. Por lo cual debemos entonces multiplicar el resultado anterior por 1000. En nuestro ejemplo la unidad de mil es 1. 1 x 1000 es 1000. Y luego restar de nuestro número completo, estos 1000. Y para tener las centenas simplemente dividir este número de la resta anterior entre 100. **Centenas = (1043 - (1 * 1000)) \ 100**

Hagamos el ejemplo: (1043 - 1000) \ 100 = 43 \ 100 = 0.43 (el \ toma solo el 0)
Hagamos las decenas siguiendo el mismo pensamiento lógico anterior.
Necesitamos deshacernos de la unidad de mil y de las centenas. Para esto igual multiplicamos nuestra unidad de mil por 1000. Las centenas que nos dio en el paso anterior multiplicadas por 100. Y lo sumamos. El resultado debe ser restado de nuestro número. **Decenas = (1043 - (1 * 1000 + 0 * 100)) \ 10 = 4.3 (El \ toma solo el 4)**

Por último la unidad. Necesitamos como ya hemos visto, deshacernos de la unidad de mil, de las centenas, y de las decenas. Para esto igual multiplicamos nuestra unidad de mil por 1000. Las centenas que nos dio en el paso anterior multiplicadas por 100. Las decenas del paso anterior por 10. Y sumamos esos 3 resultados. Luego lo restamos de nuestro número y tendremos las unidades. **Unidades = (1043 - (1 * 1000 + 0 * 100 + 4 * 10)) = 3**