Universidad Nacional Autónoma de México

Facultad de ingeniería

ESTRUCTURA DE DATOS Y ALGORITMOS I

Actividad 1

Acordeón del Lenguaje C y Ruby

Sánchez García Rocío

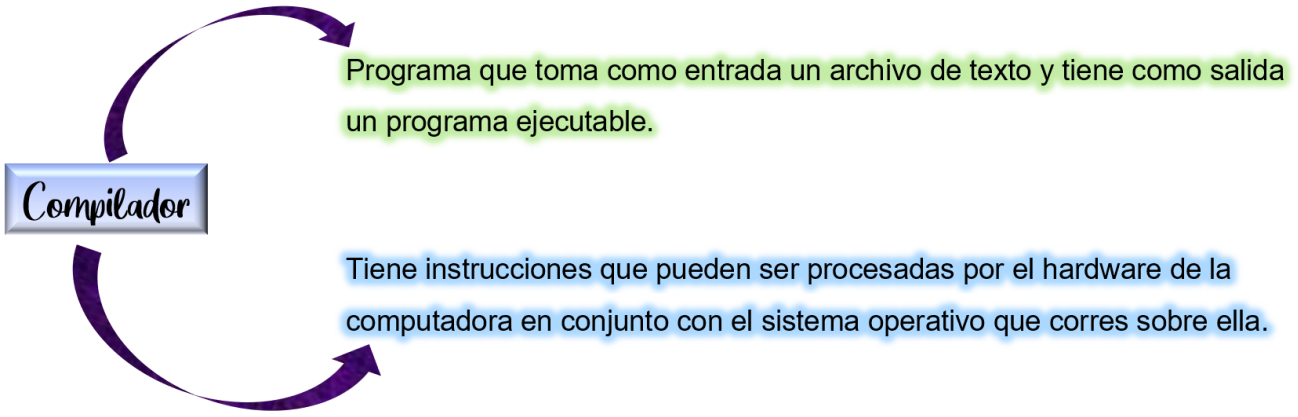
26/02/2021

Actividad. Realizar un acordeón del Lenguaje C con lo visto en las prácticas de Laboratorio de Fundamentos de Programación

Lenguaje C

Las instrucciones de lenguaje C son fáciles de entender para las personas. Para elaborar un programa en C se necesita escribir las instrucciones en un archivo de texto para posteriormente sean procesadas por un compilador.

Editores de C



Programa en C:

- Escrito en un editor de texto
- Generar un programa ejecutable en la computadora por medio de un compilador.

Editor de texto	Procesador de texto
▪ Edita un texto plano que ▪ Puede tener muchas utilidades como guardar la configuración y tener escrito un programa. Se interpreta cuando se hace la lectura de este.	▪ Permite dar formato al texto, a la hoja donde es escrito. ▪ Se puede incrustar imágenes. ▪ Su salida puede ser un archivo de texto plano que contiene etiquetas que señalan el formato que se le dio al texto.

Compiladores

Programa que produce un archivo ejecutable que depende totalmente del hardware de la computadora y del sistema operativo que corres sobre ella.

Ejecución

Una vez que se a compilado el programa es posible distribuirlo para equipos que ejecuten el mismo sistema operativo y tengan la misma plataforma de hardware. Los pasos para realizar la ejecución dependen del sistema operativo y del entorno.

Codificación

Se puede realizar en cualquier lenguaje de programación estructurada. Dentro del ciclo de vida del software, la implementación de un algoritmo se encuentra dentro de la etapa de codificación de problema.

El proceso del desarrollo del lenguaje C se originó con la creación de un lenguaje llamada BCPL que fu desarrollado por Marín Richards.

Ventajas del lenguaje C:

- Economía de expresión
- Conjunto de operadores
- Estructura de datos
- Control de flujo

Etapas principales para para la creación de un programa en C:

Edición: descripción del código fuente en lenguaje C desde un editor de textos.

Compilación: A partir del código fuente se genera el archivo en lenguaje máquina.

Ejecución: El archivo en lenguaje maquina se puede ejecutar en la arquitectura correspondiente.

La función principal debe llamarse *main ()*

Comentarios

Se utilizan para documentar el programa

Tipos de comentarios:

- **Comentario por línea:** se inicia con los símbolos '//' y se termina con el salto de línea.
- **Comentario por bloque:** se inicia cuando se insertan los símbolos '/*' y se termina cuando se insertan los símbolos '*/'. Este comentario puede abarcar varios renglones.

Declaración de variables

Sintaxis:

```
[modificadores] tipoDeDato identificador [= valor];
```

- Declaración con varios identificadores

```
tipoDeDato identificador1[= valor], identificador2[= valor];
```

Tipos de datos

Los tipos de datos básicos en C son:

- Caracteres: codificación definida por maquina
- Enteros: números sin punto decimal
- Flotantes: números reales de precisión normal
- Dobles: números reales de doble precisión

Identificador

Se trata del nombre en el que se va a almacenar en memoria un tipo de dato.

Reglas para los identificadores.

- Debe iniciar con una letra [a-z].
- Puede contener letras [A-Z, a-z], números [0-9] y el carácter guion bajo (_)

La biblioteca *'stdio.h'* contiene diversas funciones tanto para imprimir en la salida estándar como para leer en la entrada estándar.

printf

se tiene que especificar entre comillas el tipo de dato que se quiere imprimir, se puede combinar con la impresión de un texto determinado.

```
printf(El valor de la variable real es: %lf”, varReal);
```

scanf

Función que sirve para leer los datos de la entrada estándar

```
scanf(“%i”, &varEntera);
```

secuencias de caracteres de escape:

\a carácter de alarma

\b retroceso

\f anvance de hoja

\n salto de línea

\r regreso de carro

\t tabulador horizontal

\v tabulador vertical

‘\0’ carácter nulo

Expresiones lógicas

Están constituidas por números, caracteres, constantes o variables que están relacionados entre sí por operadores lógicos. Permiten formular condiciones complejas a partir de condiciones simples.

Estructuras de selección

También se les llama condicionales y permiten realizar una u otra acción en base a una expresión lógica.

- if else: se evalúa una expresión lógica y si la condición es verdadera se ejecutan las instrucciones del bloque, de lo contario se ejecutara el bloque de código que se encuentra después de la palabra reservada ‘else’.

  Sintaxis:

```
if (expresión_lógica)
{
    //Bloque de código a ejecutar
    //si la condición es verdadera
}
else
{
    //Bloque de código a ejecutar
    //si la condición es falsa
}
```

- switch case: evalu la variable despues de la palabra resrevada switch y la compara con los valores constantes que posee cada clase
  tipos de datos que puede evaluar la estructura:
  - enteros
  - caracteres
  - enumeraciones

  Sintaxis:

```
switch (opción_a_evaluar)
{
    case valor1:
        /*Código a ejecutar*/
    break;
    case valor2:
        /*Código a ejecutar*/
    break;
    …
    case valorN:
        /*Código a ejecutar*/
    break;
    default:
    /*Código a ejecutar*/
}
```

Estructuras de repetición

También llamadas estructuras cíclicas, permiten ejecutar un conjunto de instrucciones de manera repetida mientras que la expresión a evaluar sea verdadera. Existen tres estructuras de repetición:

- while: primero valida la expresion lógica y de ser verdadera ejecuta el bloque de instrucciones de la estructura. Si la condición es falsa se continua con el flujo normal del programa.

  Sintaxis:

```
while (expresion_lógica)
{
//Bloque de código a repetir
//mientras que la expresion
//lógica sea verdadera.
}
```

- do-while: ejecuta el bloque de código que se encuentra en la estructura y después s encarga de validar la condición.

  Sintaxis:

```
do
{
    /*
    Bloque de código que se ejecuta
    por lo menos una vez y se repite
    mientras la expresión lógica sea
    verdadera.
    */
} while (expresión_lógica);
```

- for: permite realizar repeticiones cuando se conoce el número de elementos a repetir.

Sintaxis:

```
for(inicialización ; expresión_lógica ; operaciones por iteración)
{
```

```
    /*
        Bloque de código a ejecutar
    */
}
```

Define

Permite definir constantes o literales. Se les nombra también como constantes simbólicas.

Sintaxis:

```
#define<nombre><valor>
```

Depuración de programas

Depurar un programa significa someterlo a un ambiente de ejecución controlado por medio de herramientas especializadas para ello.

Arreglos

Conjunto de datos del mismo tipo asociados por el nombre de una variable. Reservan la memoria a utilizar a través de una inicialización o reservación de memoria.

Sintaxis:

```
tipoDeDato nombre [tamaño]
```

Lenguaje Ruby

Fue creado por Yukihiro Matsumoto (aka Matz) en 1993 en Japón. Se trata de un código abierto. Parra una programación orientada a objetos, tiene la posibilidad de realizar directamente llamadas al sistema operativo y retroalimentación inmediata mediante el proceso de desarrollo.

No se necesita hacer las declaraciones de variable por lo que su sintaxis es simple

- Es capaz de desarrollar:
  - Procesamiento de datos de Backend
  - Aplicaciones de red
  - Aplicaciones de servicio web.

Este lenguaje no es compilado, debe haber un intérprete que evalué el código y lo traduzca en lenguaje de maquina entendible por un ordenador. Es un software libre y multiplataforma

Operadores básicos

- Aritméticos: + - / * ** %
- Relacionales: == != =
- Lógicos: and or !

Impresión

- puts

  Realiza un salto de línea

  `puts “a”,“b” #=>`$\frac{a}{b}$
- print

  `print “a”, “b” #=> ab`

Operador !

Este lenguaje permite realizar las funciones sobre los objetos sin guardar cambios sobre si mismo.

Cadenas

- “Comilla doble”
  Permiten la presencia de caracteres de escape procedidos por un backslash y la expresión de evaluación #{}

```
                    a
puts “a\nb\nc” #=> b
                    c
```

- ‘comilla sencilla’
  `puts ‘a\nb\nc’ #=> a\nb\nc`

Concatenación

Es posible llevarlo a cabo con el carácter +:

```
Puts “Ruby” + “on Rails” #=>Ruby on Rails
```

Clases

Su estructura guarda cierta similitud con Python, nos permite declarar atributos y métodos a una clase de manera intuitiva.

- Herencia
- Encapsulamiento
- Polimorfismo

Rangos

Sintaxis:

```
a = (1..10.to_a #=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

a = (1..10.to_a #=> [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Estructuras de control

- *while*: posee cuatro palabras reservadas para hacer operaciones en cada ciclo.

- **break:** termina el ciclo
- **next:** termina la ejecución del bucle.
- **redo:** Reinicia la iteración actual
- **return**

- *case:* es usada para comprobar un valor:

sintaxis:

```
valor = 30
case valor
   when 30, (1..10)
       puts “1-10” + “, o puede ser 30”
 when 11..20
     puts “11 - 20”
 end
```

for-in

En colecciones

```
numeros = [1,2,3,4,5]
for numero in numeros
  #Comandos
End
```

En rangos

```
for i in (0..4) #=> (0,1,2,3,4)
  #Comandos
end
```

Tipos de variables

Permite declarar distintas clases de variables, las cuales se especifican de la siguiente manera:

• Variable global: $*[a-z]+[a-zA-Z0-9]**
• Variable de instancia: *@[a-z]+[a-zA-Z0-9]**
• Variable local: [a-z]+[a-zA-Z0-9]*
• Constante: *[A-Z]+[a-zA-Z0-9]**

Bibliografía

Cerón L. P, Dajer C. A. (2017). *Tutorial de Ruby.* Lenguajes de programacion. Universidad Nacional de Colombia.

Manual de prácticas del Laboratorio de Fundamentos de Programación, Facultad de Ingeniería UNAM, recuperada el 14 de octubre, en http://lcp02.fi-b.unam.mx/