Introducción a la Programación

Conceptos Básicos

Programación I

Licenciatura en Ciencias de la Computación

Algoritmos y Programas

Nuestro objetivo es aprender a **programar**, esto es, a **diseñar algoritmos** y expresarlos como **programas** escritos en un **lenguaje de programación** para poder ejecutarlos en una **computadora**.

Un **algoritmo** es, sencillamente, una secuencia de pasos finita orientada a la consecución de un objetivo.

Para **"diseñar algoritmos"** primero vamos a describir de una manera rigurosa el problema que queremos resolver (especificación) para luego mediante pasos metódicos finamenente obtener el algoritmo (derivación).

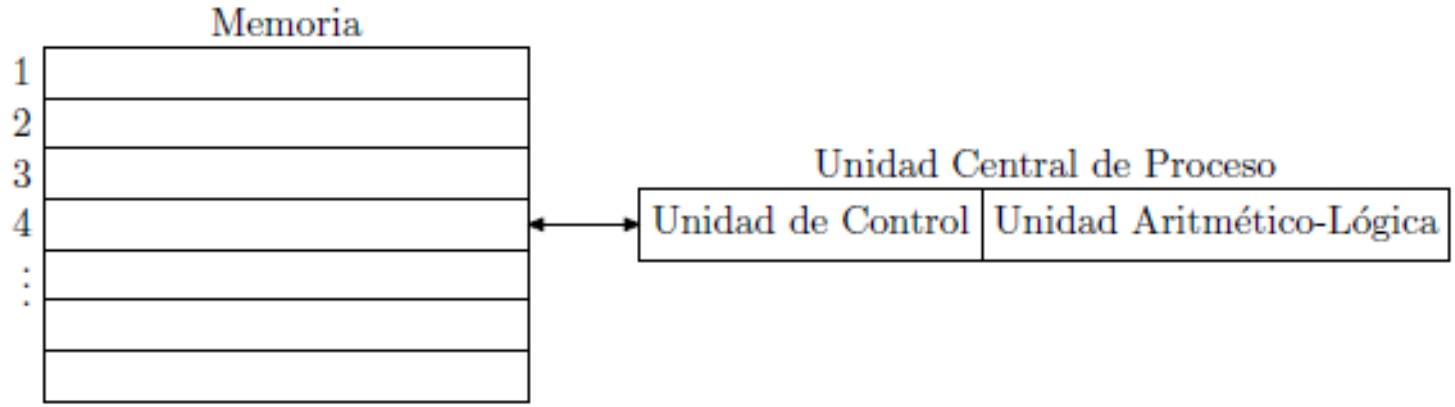
Computadoras

El diccionario de la Real Academia define computador electrónico como: “Máquina electrónica, analógica o digital, dotada de una **memoria** de gran capacidad y de métodos de tratamiento de la información, **capaz de resolver problemas matemáticos y lógicos** mediante la utilización automática de programas informáticos’.

La propia definición nos da indicaciones acerca de algunos elementos básicos del computadora:

- la memoria,
- y algún dispositivo capaz de efectuar cálculos matemáticos y lógicos.

Computadoras



- En **la memoria** es un gran almacén de información.
- Tenemos un dispositivo encargado de realizar operaciones matemático lógicas, que recibe el nombre de **Unidad Aritmético lógica (UAL)** (o ALU en inglés).
- Otro dispositivo se encarga de transportar la información de la memoria a la UAL, y de controlarla para que haga las operaciones correspondientes, **la unidad de control**.

Codificación de la información

- Cada uno de los "cajones" que conforman la memoria recibe el nombre de **celda** (de memoria) y el número que lo identifica es su **posición** o **dirección**.
- Cada posición de memoria permite almacenar una secuencia de unos y ceros de tamaño fijo, o sea un numero **binario**.
- Se usan ceros y unos porque es fácil representarlos con la tecnología actual (hay corriente o no) (encendido / apagado).
- A una variable que sólo puede tomar 2 valores se la llama **bit** y una secuencia de 8 bits se conoce como **byte**.

Programas

- La CPU, el cerebro del ordenador, es capaz de ejecutar acciones especificadas mediante secuencias de instrucciones. Una **instrucción** describe una acción muy simple.

- Las instrucciones se representan mediante combinaciones particulares de unos y ceros (valores binarios) y, por tanto, se pueden almacenar en la memoria.

- Combinando inteligentemente las instrucciones en una secuencia podemos hacer que la CPU ejecute cálculos más complejos. Una secuencia de instrucciones es un **programa**.

Lenguajes

- Las secuencias de instrucciones que el ordenador puede ejecutar reciben el nombre de **programas en código de máquina**, porque el lenguaje de programación en el que están expresadas recibe el nombre de código de máquina.

- Un **lenguaje de programación** es cualquier sistema de notación que permite expresar programas.

Código máquina

- El código de máquina codifica las secuencias de instrucciones como sucesiones de unos y ceros que siguen ciertas reglas.

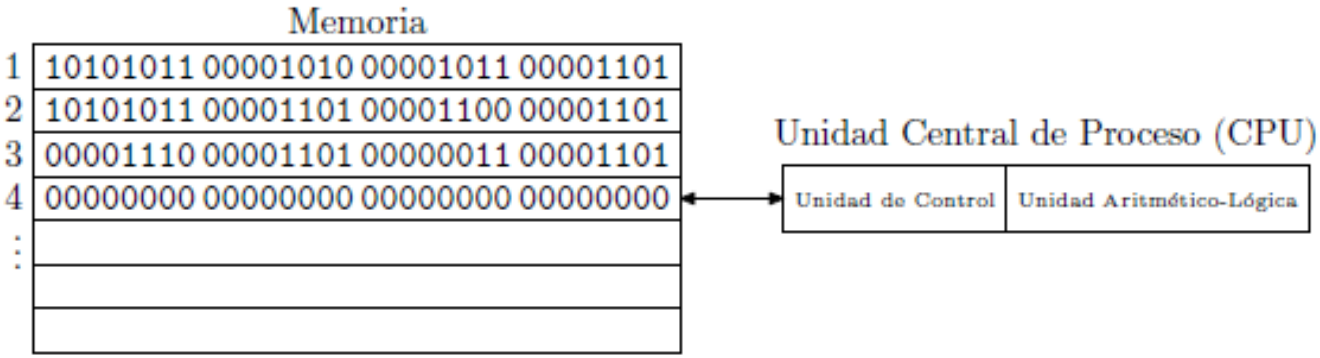
- Un programa que, por ejemplo, calcula la media de tres números almacenados en las posiciones de memoria 10, 11 y 12, respectivamente, y deja el resultado en la posición de memoria 13, podría tener el siguiente aspecto expresado de forma comprensible para nosotros

Memoria

1	`Sumar contenido de direcciones 10 y 11 y dejar resultado en dirección 13`
2	`Sumar contenido de direcciones 13 y 12 y dejar resultado en dirección 13`
3	`Dividir contenido de dirección 13 por 3 y dejar resultado en dirección 13`
4	`Detener`

Código máquina

- En realidad, el contenido de cada dirección estaría codificado como una serie de unos y ceros . . .



- La CPU es un ingenioso sistema de circuitos electrónicos capaz de interpretar el significado de cada una de esas secuencias de bits y llevar a cabo las acciones que codifican.

Lenguaje ensamblador

- En los primeros tiempos de la informática los programas se introducían en el ordenador directamente en código de máquina, indicando uno por uno el valor de los bits de cada una de las posiciones de memoria. Engorroso!

- El **ensamblador** es un programa traductor que lee el contenido de las direcciones de memoria en las que hemos almacenado códigos mnemotécnicos y escribe en otras posiciones de memoria sus instrucciones asociadas en código de máquina.

- El repertorio de códigos mnemotécnicos traducible a código de máquina y las reglas que permiten combinarlos, expresar direcciones, codificar valores numéricos, etc., recibe el nombre de **lenguaje ensamblador**.

Lenguaje ensamblador

- Nuestro programa que calcula la media de tres números almacenados en las posiciones de memoria 10, 11 y 12, respectivamente, y deja el resultado en la posición de memoria 13, ahora se vería algo así:

```
SUM #10, #11, #13
SUM #13, #12, #13
DIV #13 3, #13
FIN
```

¿Un programa diferente para cada ordenador?

- Cada CPU tiene su propio juego de instrucciones y, en consecuencia, un código de maquina y uno o más lenguajes ensambladores propios.

- Si queremos que un programa se ejecute en más de un tipo de ordenador, ¿habrá que escribirlo de nuevo para cada CPU particular?

¿Un programa diferente para cada ordenador?

¡Hola, mundo!

Nos gustaría mostrarte el aspecto de los programas escritos en lenguajes ensambladores reales con un par de ejemplos. Es una tradición ilustrar los diferentes lenguajes de programación con un programa sencillo que se limita a mostrar por pantalla el mensaje «Hello, World!» («¡Hola, mundo!»), así que la seguiremos. He aquí ese programa escrito en los lenguajes ensambladores de dos CPU distintas: a mano izquierda, el de los procesadores 80x86 de Intel (cuyo último representante por el momento es el Pentium 4) y a mano derecha, el de los procesadores de la familia Motorola 68000 (que es el procesador de los primeros ordenadores Apple Macintosh).

```
    .data
msg:
    .string "Hello, World!\n"
len:
    .long . - msg
.text
.globl _start
_start:
    push $len
    push $msg
    push $1
    movl $0x4, %eax
    call _syscall
    addl $12, %esp
    push $0
    movl $0x1, %eax
    call _syscall
_syscall:
    int  $0x80
    ret
```

```
start:
  move.l #msg,-(a7)
  move.w #9,-(a7)
  trap #1
  addq.l #6,a7
  move.w #1,-(a7)
  trap #1
  addq.l  #2,a7
  clr -(a7)
  trap #1
  msg: dc.b "Hello, World!",10,13,0
```

Como puedes ver, ambos programas presentan un aspecto muy diferente. Por otra parte, los dos son bastante largos (entre 10 y 20 líneas) y de difícil comprensión.

Lenguajes de programación de alto nivel

- Por esto surgen los **lenguajes de alto nivel** con tienen una alta abstracción de los detalles de la computadora, además están "más cerca" del lenguaje natural. En contraposición los lenguajes anteriores son llamados **lenguajes de bajo nivel**.(Finales de 1950)

```
a = 5
b = 10
c = 6
media = (a + b + c)/3
```

- Para cada lenguaje de alto nivel y para cada CPU se puede escribir un programa que se encargue de traducir las instrucciones del lenguaje de alto nivel a instrucciones de código de máquina,

Lenguajes de Programación - Composición

- **Sintaxis:** Es forma visible de un lenguaje de programación, los simbolos que forman el lenguaje y las reglas para combinarlos. Suele describirse mediante una **gramática**.

- **Semántica:** Las reglas que determina el significado de los programas constituyen la semántica de los lenguajes de programación.

La definición precisa de cada una de estas partes define a un lenguaje de programación.

Compiladores e intérpretes

Hemos dicho que los lenguajes de alto nivel se traducen automáticamente a código de máquina, pero existen dos tipos diferentes de traductores.

Un compilador lee completamente un programa en un lenguaje de alto nivel y lo traduce en su integridad a un programa de código de máquina equivalente.

Un intérprete actúa de un modo distinto: lee un programa escrito en un lenguaje de alto nivel instrucción a instrucción y, para cada una de ellas, efectúa una traducción a las instrucciones de código de máquina equivalentes y las ejecuta inmediatamente.

Compiladores e intérpretes

- Por regla general, los intérpretes ejecutarán los programas más lentamente, pues al tiempo de ejecución del código de máquina se suma el que consume la traducción simultánea. Además, un compilador puede examinar el programa de alto nivel abarcando más de una instrucción cada vez, por lo que es capaz de producir mejores traducciones.

- Pero, por regla general, los interpretes permiten una mayor flexibilidad que los compiladores y ciertos lenguajes de programación de alto nivel han sido diseñados para explotar esa mayor flexibilidad.

Compiladores e intérpretes

- Aunque nada impide que compilemos o interpretemos cualquier lenguaje de programación, ciertos lenguajes se consideran apropiados para que la traducción se lleve a cabo con un compilador y otros no. Es más apropiado hablar, pues, de lenguajes de programación típicamente interpretados y lenguajes de programación típicamente compilados.

- Entre los primeros podemos citar Python, Perl, Tcl, Ruby, **Bash**, PHP o Lisp.

- Entre los segundos, C, Pascal, C++ o Fortran.

Etapas en el desarrollo del software

Para crear correctamente software es necesario pasar por las distintas **etapas en el desarrollo del software**. Las cuales son:

Especificación Definición precisa del problema.

Diseño Elección de una solución y división del problema en partes.

Implementación Escritura de la solución en un lenguaje de programación

Validación Prueba de que el programa es correcto respecto a la especificación.

Mantenimiento Corrección de errores y adaptación a nuevos requisitos.

Especificación

- Consiste en definir el problema inicial de forma clara y precisa. Suele utilizarse un lenguaje formal(por ejemplo, el lenguaje de la lógica matemática).
- Puede considerarse como un contrato que debe cumplir el programa para ser solución del problema.
- Para escribir una especificación debemos evitar pensar en una solución para el problema y limitarnos a describir cual es el problema a resolver. Esta debe proveernos una respuesta a la pregunta, ¿qué hace el programa? y no a la pregunta ¿cómo lo hace?
- Generalmente existen distintas formas de crear un programa que cumpla con la especificación.

Diseño

- En esta etapa se decide como dividir el software en partes, que porción del problema resuelve cada una y como interactúan las partes.

- Existen muchos estilos a seguir, los cuales proveen descripciones de tipos de elementos y relaciones, junto con restricciones sobre como deben usarse.

Implementación

- En la etapa de implementación se debe pensar en la solución al problema y escribir la misma en un lenguaje de programación.
- Un algoritmo es un conjunto finito ordenado de pasos que especifican la secuencia de operaciones que se han de realizar para resolver un determinado problema.
- Un algoritmo debe ser de fácil lectura e interpretación e independiente del lenguaje de programación.
- Para representar los algoritmos se suelen usar pseudocódigos (mezcla de lenguaje natural y expresiones matemáticas que permiten describir de un modo preciso un programa).

Validación

- Puede ocurrir que el programa no resuelva correctamente el problema. En la etapa de validación se intenta asegurar que un programa cumple con su especificación.

- Existen dos formas de llevar a cabo dicha tarea, aplicando la técnica testing o realizando una verificación formal.

Testing

Testing: consiste en preparar un conjunto de datos de entrada, lo mas amplio posible, probar el programa con estos datos y ver si el resultado es el esperado.

- En general los datos de entrada con que se tienen que probar los programas son infinitos.
- El testing NO garantiza que el programa no tenga errores.
- Existen muchas técnicas para hacer testing.

Validación Formal

Verificación formal: consiste en demostrar matemáticamente que un programa cumple su especificación.

- Tal demostración puede cubrir infinitos valores de entrada.
- Permite probar que un programa es correcto (respecto a su especificación).
- Es mas difícil de realizar que el testing.

Mantenimiento

Es común que se encuentren errores un tiempo después de que el software comenzó a ser utilizado, o que se cambien los requerimientos del mismo.

Algunos tipos de errores que pueden encontrarse son:

Errores de especificación: fallos cometidos en la fase de especificación. Su corrección es muy compleja ya que hay que modificar el trabajo realizado durante las fases siguientes.

Errores logicos: se detectan al ejecutar el programa con cierta entrada y comprobar que se obtuvieron datos incorrectos como salida. En este caso el programa no cumple con su especificación.

Documentación

- Es importante que la documentación se lleve a cabo en cada una de las etapas de desarrollo del software.

- Tiene como objetivo que cualquier persona externa al desarrollo pueda modificar el software.

- Por ejemplo, en la etapa de implementación se puede proveer:

  Documentación interna: viene contenida en el propio programa fuente.

  Documentación externa: esta constituida por una serie de archivos que acompañan al programa. Entre ellos pueden estar: descripción de las versiones,descripción de archivos y estructuras de datos, descripción del programa principal y subprogramas, manual de mantenimiento.

Referencias


Andrés Marzal, Isabel Garcia *Introducción a la programación con Python*, Capítulo 1, 2003.


C.Mancino, M Notti, G Amadio. *Introducción a la programación*, apunte 2009.