Desenvolvemento de Algoritmos

Algoritmo

Un **algoritmo** é un conxunto de pasos ben definidos, finitos e ordenados, que permiten resolver un problema ou realizar unha tarefa.

Características principais:

- **Claridade**: cada paso debe ser preciso e sen ambigüidades.
- **Finitude**: o proceso debe rematar despois dun número determinado de pasos.
- **Eficiencia**: idealmente, o algoritmo debería empregar o menor tempo e recursos posibles.

Exemplos:

Algoritmo para facer un té:

- Quentar Auga
- Poñer unha bolsa de té na cunca.
- Botar a auga quente.
- Agardar uns minutos.
- Retirar a bolsa e servir.

Algoritmo para calcular a media de tres números:

- Sumar os tres números
- Dividir o resultado da suma por 3.
- O resultado é a media.

Programa

Un programa é a expresión dun algoritmo nunha linguaxe de programación. É o código fonte que un ordenador pode executar. Mentres o algoritmo é a idea ou a lóxica, o programa é a súa tradución a unha linguaxe que a máquina entende.

Programa para calcular a media de tres números en Python:

```
def calcular_media(a, b, c):
    media = (a + b + c) / 3
    return media
```

Compiladores e Intérpretes. A compilación e enlazado (Linking)
quí explícanse as dúas formas principais de executar código.

- **Intérprete:** Traduce e executa o código fonte liña por liña. A vantaxe é que podes ver os resultados inmediatamente. As linguaxes interpretadas máis comúns son Python, JavaScript ou PHP.

- **Compilador:** Traduce todo o código fonte a código máquina nun proceso chamado **compilación**. Este proceso crea un ficheiro executable que pode ser executado directamente pola máquina. As linguaxes compiladas máis comúns son C, C++ ou Java.

As linguaxes de programación teñen unha cantidade moi limitada de acción, limitándose normalmente a:

- Definición de variables
- Operacións matemáticas e lóxicas básicas mediante operadores matemáticos e lóxicos
- Sentencias de control (Selección e Iteracón)
- Creación de funcións e paso de parámetros

Ademais proporcionan unha serie de algoritmos (funcións) para realizar accións máis complexas que podemos incorporar mediante a biblioteca estándar de funcións da linguaxe.

A Compilación e o Enlazado (Linking)

A **compilación** é o proceso de converter o código fonte (o que escribe o programador, composto de sentencias nunha linguaxe de programación) a código obxecto (o que executa o ordenador composto por códigos numéricos). Por exemplo, un compilador de C converterá un ficheiro `.c` (código fonte) a un ficheiro `.o` (código obxecto).

E posible agrupar varios algoritmos para resolver problemas concretos (pasar un texto a maiúsculas, calcular a mediana dun conxunto de valores, desprazar un gráfico na pantalla....) nun único ficheiro denominado ***libraría*** ou ***biblioteca*** que xeito que poden ser reutilizados facilmente en novos programas.

As linguaxes de programación unicamente teñen sentencias e operacións matemático-lóxicas básicas, pero incorporan numerosas bibliotecas con multitude de módulos en código obxecto para realizar unha gran cantidade de accións complexas en múltiples ámbitos (librarías estándar do sistema). Habitualmente esas librarías se irán ampliando con módulos de terceiros ou de deseño propio.

O **enlazado** ou **linking** é o proceso de combinar o código obxecto cos módulos das librarías que o teu programa necesita. Por exemplo, se usas unha función matemática, o **enlazador** unirá o teu código coas funcións necesarias desa libraría para crear o programa final e executable.

Variables

Unha **variable** é un espazo de memoria que almacena datos durante a execución do programa. Ten as seguintes características:

•**Nome**: Identificador único (ex: idade).
•**Tipo**: Define a natureza dos datos (ex: enteiro, carácter, booleano).
•**Valor**: Dato almacenado.

Exemplos:

```
idade = 25;      // Variable enteira
altura = 1.75;  // Variable decimal
letra = 'A';     // Variable carácter
texto = “Cadea de texto”';     // Variable tipo texto
area = base * altura;  // Variable decimal. Expresión aritmética e asignación
```

Como podemos ver, as principais operacións con variables son a asignación (=) e o uso do valor especificando o nome da variable

Os identificadores *son nomes que eliximos para distintos elementos do programa como variables, funcións, clases, ... etc. Cada linguaxe ten as súas propias restricións para formar eses nomes, pero en xeral non poden empezar por número, levar espazos en branco nin símbolos especiais salvo o _ (suliñado baixo). Tamén habitualmente se consideran as letras do alfabeto non inglés como “símbolos especiais”.*

As variables residen nunha zona da memoria do ordenador. A memoria de ordenador está dividida en conxuntos de 8 bits denominados bytes, e cada byte se identifica cun valor numérico coñecido como ***dirección de memoria***. Polo tanto, ***as variables comezan nunha dirección de memoria*** que é a dirección de memoria onde reside a información almacenada.

O número de bytes que ocupa a variable depende do tipo de información que se almacene na variable

Segundo o tratamento das variables, as linguaxes de programación se poden dividir en:

- L***inguaxes sen tipado ou tipado débil***: As variables son so nomes para os valores, e o compilador non realiza comprobacións.
- **Linguaxes con tipado forte:** Non é posible facer operacións con datos de tipos incompatibles sen facer conversións explicitamente.
- **Linguaxes con tipado estático:** E necesario indicar o tipo de datos que pode almacenar unha variable no momento de creala. Polo tanto, se coñece o tamaño que ocupan na memoria no instante da súa creación, e ese tamaño non varía.
- **Linguaxes de tipado dinámico:** O tipo de datos da variable se establece no momento da asignación, e pode variar segundo o dato almacenado. Polo tanto non é necesario especificar o tipo de variable.

Ainda que o tipado dinámico parece “maís cómodo” realmente produce problemas en tempo de execución xa que non é posible comprobar que os datos que se pasan ou reciben das funcións son do tipo correcto. Debido a isto, as linguaxes de tipado dinámico están incorporando extensións de tipado (t***ype hinting***) que permiten especificar o tipo de datos que vai a almacenar a variable e comprobar a corrección dos datos.

Expresións Aritmético-Lóxicas

- Unha expresión aritmética da como resultado un valor numérico
- Unha expresión lóxica da como resultado un valor lóxico (verdadeiro ou falso)

As distintas linguaxes de programación proporcionan unha serie de operadores aritméticos que permiten realizar as operacións aritméticas máis comúns (suma, resta, multiplicación, división, módulo -calculo do resto-, desprazamentos de bits, operacións matemáticas AND, OR, XOR e NOT a nivel binario....) e outra serie de operadores lóxicos que permiten realizar operacións lóxicas (comparar valores, maior, menor, distinto, OR lóxico, AND lóxico, NOT lóxico, maior ou igual, menor ou igual).

Unha expresión aritmética emprega operadores aritméticos, mentres que unha expresión lóxica emprega operadores lóxicos.

Tanto nas expresións aritméticas como lóxicas é posible agrupar e alterar a precedencia das operación mediante o uso de parénteses e corchetes.

As expresións lóxicas se utilizan habitualmente nas **sentencias de control** para indicar baixo qué circunstancias se debe executar un código e durante canto tempo, mentres que as **sentencias aritméticas** normalmente se empregan no cálculo de datos.

Exemplos:

- c = (10 - 4) * valor // sentencia aritmética. parénteses alteran a orde
- d = (5 > 3) OR (2 > valor) // Sentencia lóxica

Funcións

Un programa complexo normalmente está composto de moitas accións complexas. Por exemplo, para facer unha tortilla:

- Pelamos e cortamos patacas e cebola de determinado xeito
- Collemos unha tixola a e botamos unha cantidade determinada de aceite de oliva
- Esperamos a que o aceite quente ata o punto axeitado
- Botamos as patacas e a cebola
- Preparamos a cantidade de ovos batidos necesaria
- Esperamos a que as patacas e a cebola estean no punto axeitado
- Escorremos as patacas e a cebola e a misturamos co ovo
- Baleiramos a tixola de aceita e botamos a mistura
- Cociñamos ata que estea a tortilla

Este sería o algoritmo para elaborar unha tortilla. Algunhas das accións necesarias son simples, como botar as patacas e cebola na tixola, mentres que outras accións son complexas e requiren de accións máis simples. Por exemplo “Preparar a cantidade de ovos batidos necesaria” require:

- Coller a cantidade de ovos especificada
- Cascar cada un dos ovos e vertelos nun bol
- Cun garfo ou batedor bater os ovos ata que teñamos unha mezcla homoxénea

Unha función é un conxunto de accións para realizar unha acción concreta agrupadas baixo un nome e que se pode utilizar unha ou varias veces nun programa ou incluso en múltiples programas (si a separamos a un módulo distinto que logo enlazamos)

As funcións poden recibir ***parámetros.*** Un parámetro é un conxunto de datos que se subministra a función para que poda realizar o seu cometido, e se almacenan en variables:

```
funcion bater ovos ( recibe numero de ovos)
        - Coller numero de ovos
        - Cascar cada un dos ovos e vertelos nun bol
        - Cun garfo ou batedor bater os ovos ata que teñamos unha mezcla homoxénea
```

Deste xeito poderíamos dicir ***bater ovos (5);*** para unha tortilla de 5 ovos ou ***bater ovos (12)*** para unha tortilla de 12 ovos...

As linguaxes de programación proporcionan na súa libraría de funcións da linguaxe, entre outras, as funcións necesarias para visualizar información (print, printf....) e para pedir datos (read, readLine, scanf....)

Sentencias de Control

Chamamos ***fluxo de execución*** a orde na que se van levando a cabo as instrucións do programa. Normalmente o fluxo é secuencial, e dicir, se executan da primeira ata a última segundo a orde na que se especifican.

As sentencias de control das linguaxes de programación ***permiten alterar o fluxo de execución*** do programa.

Para realizar o control da execución é necesario **agrupar as instrucións do programa en *bloques***. Normalmente cada bloque terá as súas propias variables (as que se definan dentro do bloque) e poderá acceder tamén as variables dos bloques superiores.
O xeito de agrupar as instrucións varía segundo a linguaxe de programación, sendo o habitual pechar o bloque de instrucións entre chaves { }

Selección

As sentencias de selección permiten seleccionar qué codigo queremos levar a cabo dependendo do cumprimento dunha condición (resultado dunha expresión lóxica)

```
print(“Introduce a tua idade: “)
idade = input()

if idade > 18:
   print("Eres maior de idade")
else:                # O else so se especifica en caso necesario
   print("Eres menor de idade")
```

Para o caso de seleccionar un código dependendo do valor almacenado nunha variable temos unha forma abreviada:

```
print(“Introduce o dia da semana de 1 a 7”)
dia=input()
switch (dia) {
        case 1:
           System.out.println("Luns");
           break;
        case 2:
           System.out.println("Martes");
           break;
```

```
        case 3:
           System.out.println("Mércores");
           break;
        case 4:
           System.out.println("Xoves");
           break;
        case 5:
           System.out.println("Venres");
           break;
        case 6:
           System.out.println("Sábado");
           break;
        case 7:
           System.out.println("Domingo");
           break;
        default:
           System.out.println("Número de día non válido");
}
```

Iteración

As sentencias de iteración permiten repetir un conxunto de operacións mentres sexa certa unha expresión condicional. Existen dúas formas de iterar:

1.- Avaliando a condición ANTES de comezar a execución do bloque. Si a condición non se cumpre o bloque non se executa.

```
print(“Os números divisibles por 3 menores que 10 son: “)
numero=1
while(numero<10) {
  if  (numero % 3)!=0
         print(numero+” “);
 numero=numero+1
}
print(“Rematei, o valor de numern e “+numero)
```

2.- Avaliando a condición DESPOIS de comezar a execución do bloque. Sempre se levan a cabo as instrucións do bloque como mínimo unha vez.

```
print(“Os números divisibles por 3 menores que 10 son: “)
numero=1
do {
  if  (numero % 3)!=0
         print(numero+” “)
 numero=numero+1
} while(numero<10);
print(“Rematei, o valor de numern e “+numero)
```

Existe tamén habitualmente unha versión abreviada de iteración pensada principalmente para levar a cabo contas secuenciais (como nos casos anteriores). Esta instrución se divide en 3 partes separadas por ; que son:

- ***Inicialización*** – Se leva a cabo unha única vez antes de comezar a execución do bucle
- ***Condición*** – Se avalía antes de entrar no bucle e logo cada vez que se remata a execución do bloque de instrucións. As instrucións se levarán a cabo mentres esta expresión sexa *True*
- ***Post-Condición*** – E unha instrución que se leva a cabo sempre que se remata a execución do bloque de instrucións.

```
print(“Os números divisibles por 3 menores que 10 son: “)
for (numero=1;numero<10;numero=numero+1) {
  if  (numero % 3)!=0
          print(numero+” “)
 numero=numero+1
}
print(“Rematei, o valor de numern e “+numero)
```

As tres formas de iterar son equivalentes. Calquera algoritmo pode facerse empregando unicamente unha das tres.

Programación Estruturada

Calquera algoritmo pode ser construído utilizando unicamente tres estruturas básicas de control:

1. **Secuencia:** execución de instrucións unha tras outra.
2. **Selección (ou condicional):** elixir entre dúas ou máis opcións en función dunha condición (`if`, `switch`).
3. **Iteración (ou bucle):** repetición dun conxunto de instrucións mentres se cumpre unha condición (`while`, `for`).

As Funcións: Paso de Parámetros e Ámbito das Variables

Deseño Descendente

O deseño descendente, tamén chamado “divide e vencerás” consiste en resolver os problemas en varios pasos sucesivos. No primeiro paso, non nos preocupamos dos detalles, se non que propoñemos unicamente os pasos xerais para resolver o problema:

Tortilla de Patacas
1.- Preparamos a cebola, as patacas e os ovos
2.- As pochamos ata o punto axeitado
3.- Retiramos o contido a un bol e retiramos o aceite
4.- Misturamos co ovo
5.- Cociñamos ata que estea a tortilla

Posteriormente collemos cada un de estes pasos complexos e o descompoñemos en problemas máis simples aplicando o mesmo proceso ata considerar que as solucións xa son suficientemente simples. Por exemplo, o punto 1:

a) Pelar e partir as patacas
b) Pelar e cortar a cebola

c) Escachar e bater os ovos nun bol

Cada paso pode dar lugar a unha función si se da unha ou varias das seguintes circunstancias:

- A acción que estamos desenvolvendo é útil para outros problemas
- A acción que estamos desenvolvendo é demasiado complexa e compre separala e utilizala dun xeito simple
- A acción que estamos desenvolvendo posiblemente a teremos que repetir en varios puntos distintos do programa

A información que se lle pasa a unha función se recibe en variables locais a función (so teñen existencia dentro da función) denominadas ***Parámetros***

Por exemplo, no punto anterior se poderían facilitar como parámetros a cantidade de patacas e cebola e o número de ovos:

Funcion Preparar_Ingredientes(npatacas, ncebola, numovos)

1. Pelar e Cortar Patacas (npatacas)
2. Pelar e Cortar Cebola (ncebola)
3. Escachar e Bater ovos (numovos)

A súa vez, por exemplo a acción 3 da lugar a un conxunto de accions complexas:

Escachar e Bater ovos (numovos)

1. ovos_procesados=1
2. Mentres ovos_procesados < = numovos,
    1. escachar o ovo, vertelo nun bol
    2. ovos_procesados=ovos_procesados+1
3. Bater o contido do bol.

Podería utilizar a funcóin así: Preparar_ingredientes(4,0.5, 6)

As variables definidas nun bloque de código unicamente teñen existencia e son accesibles dende dentro dese bloque ou os bloques anidados no seu interior. Esas variables se destrúen cando a execución sae do bloque. Son ***locais ao bloque***, e son ***globais a todos os bloques internos***.

Normalmente tamén se poden definir variables globais a todo o programa, aínda que non é en xeral unha boa práctica (calquera modificación da variable podería afectar a partes inesperadas do programa)

As Librarías ou Bibliotecas de Funcións: Enlace Estático e Enlace Dinámico

As linguaxes de programación dispoñen dun conxunto de funcións de uso común xa programadas e dispoñibles en módulos constituíndo o que se denomina ***libraría ou biblioteca do sistema***. Ademais e posible utilizar outras bibliotecas externas ou crear un conxunto propio.

Como xa comentamos, as bibliotecas utilizadas se unen ao código obxecto da nosa aplicación mediante o enlazado producindo o executable.

Existen dúas maneiras de facelo:

1. Enlace estático: Os módulos se unen producindo un executable que contén todo o código do programa. Isto produce executables grandes que son independentes do que teñamos instalado no noso sistema.
2. Enlace dinámico. A libraría non se inclúe no executable, se non que simplemente se rexistra unha referencia ao módulo que se cargará cando se precise. O tamaño dos executables son moito máis pequenos, pero precisa que o sistema dispoña das librarías utilizadas. Poden actualizarse as librarías sen necesidade de modificar o programa.