Introdución á Programación

Conceptos Básicos

Todas as linguaxes imperativas comparten os seguintes elementos:

- identificadores
- constantes
- variables
- expresións aritméticas
- expresións lóxicas
- sentencias de control: Selección e Iteración
- funcions / procedimientos
- librerías de funcións

Identificadores

Un identificador é un texto que se utiliza para identificar algo. Nas linguaxes de programación existen unhas regras de creación de identificadores. O máis estándar é:

- Poden estar formadas por números e letras do alfabeto inglés

- Non poden comezar por números

De todos modos, estas normas dependen da linguaxe de programación elixida.

E moi importante elixir identificadores significativos, existindo un compromiso entre a lonxitude do identificador e a súa claridade. A idoneidade do identificador a utilizar dependerá do problema concreto a resolver, por exemplo si nos queremos referir a idade dunha persoa, ***a*** é un mal identificador, e ***idade*** un bo identificador.

Constantes

Unha constante é un valor que non varía. Por exemplo **9**, **23** ou **11**.

Unha distinción moi importante é entre constantes numéricas e constantes de texto. As constantes numéricas se refiren a valores numéricos e se soen poñer sen comiñas, como **23**. As constantes alfabéticas se refiren a texto, e se poñen entre comiñas como **“Texto”** (*si puxeramos Texto, sen comiñas, sería realmente un identificador*).

E moi importante caer na conta que a constante “10” non é un número, se non un texto cos caracteres (letras) ‘1’ e ‘0’. Mentras que 10, sí e un número. Polo tanto con 10 podemos facer operacións matemáticas, pero con “10”, non.

As constantes significativas moitas veces se lles asigna un identificador, para facilitar a comprensión do algoritmo e facilitar a modificación. Por exemplo, podemos definir a constante PI

como o valor 3.141592. No algoritmo utilizaríamos o identificador PI, en lugar do número, e si quixeramos variar a precisión dos cálculos, so precisaríamos variar a definición da constante PI.

Variables

Unha variable e un identificador que representa un espazo de almacenamento na memoria onde podemos gardar un valor e modificalo cando consideremos oportuno.

E importante distinguir entre un identificador dunha variable, e un identificador dunha constante. Cando un identificador representa unha constante o uso do identificador e completamente igual ao uso directo da constante representada. *A constante non pode variar o seu valor durante a execución do algoritmo.*

Para almacenar información na dirección de memoria identificada pola variable, se utilizan as sentencias de asignación que empregan o signo = , ou ← :

idade = 12

idade ← 12

Segundo o modo de traballar coas variables as linguaxes se poden clasificar en linguaxes con tipo ou linguaxes sen tipo. Dentro das linguaxes con tipo podemos distinguir entre tipos fortes e tipos febles. Esta clasificación ten certa controversia, e a maior parte da literatura fala simplemente de linguaxes sen tipo ou tipo dinámico, e de linguaxes con tipo ou de tipo estático.

Linguaxes sen tipo

Unha linguaxe sen tipo non precisa especificar o tipo de información que se vai a almacenar na posición de memoria identificada, xa que o tipo necesario se determina no momento de executar o programa (*tipos dinámicos*) a partir da información que almacenamos. Isto permite almacenar valores de distinto tipo nunha mesma variable en distintas partes do programa. Exemplos de linguaxes sen tipo son *PHP* ou *Javascript*.

Por exemplo:

```
$variable=12   ; Almacenamos un valor numérico na posición de memoria identificada por variable
$variable=”HOLA”   ; na mesma variable, almacenamos un valor textual
```

Aínda que non ter a necesidade de definir os tipos de variables pode parecer moi cómodo e unha vantaxe, fai que os programas sexan máis difíciles de seguir e se facilitan os erros de programación. Por outra banda, o desenvolvemento é máis rápido.

Linguaxes con Tipo

As linguaxes con tipo precisan da definición da variable antes do seu uso, indicando o tipo de información que se vai a almacenar na posición de memoria identificada. Polo tanto, o tipo está predeterminado no momento da codificación do programa (*tipos estáticos*), e nunha variable so se poderá almacenar a información do tipo indicado.

Dentro das linguaxes con tipo, podemos distinguir entre linguaxes con “tipos febles” e linguaxes con “tipos fortes ”. Nas linguaxes con tipos febles, a definición so indica realmente a cantidade de

espazo de memoria a utilizar e a linguaxe permite realmente almacenar calquera tipo de información nel baixo a responsabilidade do programador e normalmente indicando unha conversión forzada (*casting*).

Por exemplo, na linguaxe 'C', o seguinte é correcto:

```
char x='A';      // Almacenamos o caracter 'A' na variable x
int y=x+5;       // lle sumamos ao caracter 'A' o número 5 (o caracter 'A' se garda como o código ASCII da letra 'A', que e 65, polo que en y teríamos o valor 70
```

O exemplo anterior en "PASCAL", que é unha linguaxe con tipos fortes, non está permitido xa que **x** e de tipo "caracter", polo que non se poden facer operacións matemáticas con él nin almacenar o seu contido en unha variable que non sexa de tipo "caracter".

En lugar de tipos fortes e tipos febles, que é confuso, poderíamos falar de ***tipos ríxidos*** e ***tipos flexibles***.

Tipos comúns empregados nas linguaxes son ***byte, short, int, long, float, double, boolean, char*** ...

Expresións Aritméticas

As linguaxes de programación permiten o uso de *expresións aritméticas*, que se levan a cabo facendo uso de *operadores aritméticos*. Non todas as linguaxes dispoñen da mesma cantidade de operadores aritméticos, pero os típicos son:

- **Asignación (=) :** Permite almacenar un valor ou resultado da avaliación dunha expresión nunha variable. A variable donde se vai a almacenar o valor se pon a esquerda. *NON SE DEBE CONFUNDIR CUNHA COMPARACIÓN*

Exemplo:

```
peso = 75;          // Almacena o valor 75 na zona da memoria identificada por peso
```

- **Cálculo :** Permiten realizar cálculos aritméticos seguindo as regras habituais de precedencia de operacións matemáticas, permitindo o uso de paréntese para alterala. Os operadores de cálculo habituais son **suma (+), resta (-), multiplicación (*), división (/) e resto (%)**. O operador de división realizará divisións enteiras ou decimais dependendo dos operandos. Si os operandos son enteiros, a división é enteira, noutro caso a operación é decimal.

  Outros operadores aritméticos menos habituais son os operadores a nivel de bit, que operan cos valores binarios efectuando as operacións **and (&), or (|), xor (^) e not (~)**

Exemplo:

```
// Multiplica o contido da posición de memoria identificada por velocidad por 5 e almacena o
// resultado na posición de memoria identificada por aceleracion
aceleracion = velocidad * 5;
```

Expresións Lóxicas

As expresións lóxicas se levan a cabo empregando operadores lóxicos, dando como resultado un valor booleano: certo (true) ou falso (false). Este resultado se emprega normalmente nas sentencias de selección e iteración, aínda que é posible tamén almacenalos en variables.

Os operadores de comparación son ***igual (==), maior (>), menor (<) e distinto (!=)*** . Para construir expresións lóxicas complexas se utiliza o paréntese e os operadores lóxicos **y (&&), ou (||) e non (!)**

Exemplo:

```
// Almacena na posición de memoria identificada por emaior  o resultado de comprobar si o
// contido da posición de memoria identificada por idade é maior de 18 e maior de 12
emaior = ( idade > 18 )  && ( idade > 12 )
```

Sentencias de Control: Selección e Iteración

Os programas se executan en secuencia, unha instrucción tras outra dende a primeira ata chegar ao final en orden. Si desexamos alterar a orde de execución podemos utilizar as estructuras de selección e de iteración.

- **Estructura de Selección:** A estructura de selección permite executar ou non unha instrucción ou conxunto de instruccións dependendo do resultado dunha operación lóxica (avaliación dunha condición).

Exemplo:

```
if  (( idade < 18 ) && (idade > 12)) {
        System.out.println("Es Adolescente\n");
        if (idade > 16) System.out.println("Pronto serás maior de edade\n");
} else {
        if ( (idade > 18) || (idade == 18) ) System.out.println("Es adulto");
        else                                 System.out.println("Es un neno");
}
```

- **Estructura de Iteración:** Nos permite repetir un conxunto de instruccións dependendo do resultado da avaliación dunha condición. As linguaxes de programación modernas dispoñen de varias estructuras de iteración, pero son todas equivalentes. A máis importante e a **'while'** (mentras).

Exemplo:

```
System.out.println("Os 100 primeiros números pares son: ");
num = 2
conta = 0
while ( conta < 100 ) {
        if (num != 2) System.out.print(",");
        System.out.print (num);
        conta = conta + 1;
        num = num + 2;
}
```