**Variables e instrucións**

**Datos**

* Os datos son elementos que serven de base para o exacto coñecemento dunha cousa.
* Son os valores iniciais para a solución dun problema mediante un algoritmo, e tamén os resultados do procesamento realizado.
* Cada dato está almacenado nun enderezo na memoria (ex: 0x0022FF24) que se asocia co nome da variable.

**Constante** (#define identificador valor)

* Valor que non pode cambiar durante a execución do programa
* O valor pode ser unha constante ou unha expresión constante (n\*10)

**Tipo de datos**

* O tipo dun dato determina:
  + O rango de valores que pode tomar
  + A súa representación interna
  + O conxunto de operacións que se poden realizar con eles
* A asignación de tipos aos datos ten varios obxectivos:
  + Indicar a representación axeitada na memoria
  + Detectar erros de operacións
  + Determinar como executar as operacións
* **Tipos de datos:**
  + Estáticos:
    - Simples: int, float, double, char
    - Compostos: vector, matriz, estrutura
  + Dinámicos: punteiros

**Int**

* Valores numéricos positivos e negativos (enteiros)
* Limitados a un rango simétrico en torno a 0 que depende dos bits de memoria. (2^(n-1) onde n son os bits)
  + n habitualmente é 16 ou 32. Pódese coñecer cal é mediante sizeof(int).
* En binario, pódese representar como ‘signo x magnitude’ (o primeiro bit indica signo’, pero en realidade empreganse variantes como o ‘complemento a 2’ (o negativo é o contrario do positivo e súmase 1)
* Operacións asociadas: +, -, \*, /, %(resto). O valor do resultado sempre é enteiro.

**float, double, long double** (reais)

* Valores reais positivos e negativos, precisión limitada (double posúe o doble)
  + O rango e a precisión dependen do ordenador
  + Debido á imprecisión, pode ocorrer que (a/b)\*b != a
* Require aproximacións para numeros irracionais
* Representación (en 64 bits): Signo x mantisa x Base^exponente
  + Signo: 1bit
  + Exponente: 11 bits (codificación en exceso, o valor réstase a unha constante)
  + Mantisa: 53 bits
* Operacións: +, -, \*, /

**char**

* Calqueira carácter disponible no teclado dun ordenador, alfabético / numérico / puntuación / control
* A maioría dos ordenadores adoptan a táboa ASCII
  + Asociados cun valor binario de 1 byte (-128,127 se signed ou 0,255 se unsigned)
  + Se definimos char c = ‘a’ e realizamos printf("%d, %c", c, c); imprime ‘97, a’.

**Booleano**

* Poden ser TRUE ou FALSE
* Non existen en C, o resultado de operacións lóxicas é 0 ou 1.
* **Operadores lóxicos:** Permiten combinar operaci´pns relacionais dando lugar a predicados máis complexos.
  + &&(AND), ||(OR), !(NOT)
  + Se ao empregar un && a primeira condición é falsa, non se evalúa a segunda.

**Variable**

* Espazo na memoria asociado a un nome que permite almacenar, ler e modificar un certo dato.
  + Modifícanse mediante ‘sentencias de asignación’ (variable = expresión)
* Requiren un identificador e un **tipo** de dato.
* O espazo de memoria noméase con & e o nome da variable. Para leer este enderezo, úsase scanf(%tipo, &variable);

**Compatibilidade de tipos**

* Ao empregar un dos operandos que se aplican a máis dun tipo de variable (+, \*, etc), se se intenta operar entre variables de distintos tipos, ocorrerá unha conversión:
  + Se algún operando é un long double, o outro convértese a long double
  + Se non, se algún é un double, o outro convértese a double
  + Se non, se algún é un float, o outro convértese a float
  + Se non se cumple ningunha, ambos se toman como enteiros. Logo, realízase a mesma cadea de prioridade con unsigned long int > long int > unsigned int > int.

**Iniciacións**

* Ao iniciar unha variable declárase obligatoriamente o seu nome e tipo.
* Opcionalmente, pódese aportar tamén o seu valor inicial: int a=0.
* Pódese empregar tamén int b=a. Para isto a debe ter un valor asignado.

**Instrucións de entrada/saída**

* **Saída:** Procedemento printf
  + Permite imprimir texto xunto con variables mediante a especificación de formato %c, %d, %u(unsigned int), %f, %e(double ou float en formato científico), %g(double ou float no formato máis apropiado), %s
  + Tamén permite caracteres de escape: \n, \t(tab), \r(retroceso)
  + Para escribir os caracteres “ ou \, escribense con \” e \\.
* **Entrada:** Procedemento scanf
  + Funciona igual ca o printf.
    - A especificación de formato é distinta no caso dos float, que se leen con %f, %e ou %g. Os double lense con %lf, %le e %lg.

**Cadeas de carácteres**

* En C decláranse na mesma forma ca outro vector, pero de caracteres.
* **char** nome\_cadea [tamaño].
* Os strings rematan sempre cun carácter nulo, \0, correspondente á posicion 0 da táboa ASCII.
  + Para declarar unha string por declaración débese incluír este caracter na declaración: **char** cidade[] = {‘V’, ‘i’, ‘g’, ‘o’, ‘\0’};
  + Non se pode asignar unha string enteira fóra da súa declaración. É preciso asignar cada caracter por separado, por exemplo, mediante un bucle for.
* Ao empregar **scanf** co especificador %s, só se le a primeira palabra, pois scanf remata ao ler un espazo.
  + Para solucionar isto, scanf("%[^’\n’]", cidade) le ata o \n.
    - nótese que non se emprega &, ao ser un vector.
  + Tamén é posible empregar o procedemento **gets:** gets(cidade)