Algunes nocions bàsiques sobre C

Marc Graells Ricardo

15 de març de 2017

- `tots`, `Windows`, `Linux`
- `>cd`, `pwd` Ens diu a quin directori estem.
- `>cd "subdirectori"` Ens envia a "subdirectori".
- `>cd..` Ens envia al directori superior.
- `>dir`, `ls` Ens ensenya quins directoris hi ha al directori actual.
- `>gcc -Wall -O3 -o nomarxiu nomarxiu.c`
  Ens compila un arxiu nomarxiu.c a nomarxiu.exe .
  - `gcc` El programa que realment fa la compilació (traducció de codi font a codi màquina).
  - `-Wall` Diem que volem saber tots (all) els avisos (Warnings).
  - `-O3` És el nivell d'optimització.
  - `-o nomarxiu` És el nom que volem donar al programa que es crea.
  - `nomarxiu.c` És el nom del fitxer a compilar.
- `>cls`, `clear` Neteja la consola.
- `>arxiu.exe`, `./axiu.exe` Executa un programa anomenat `arxiu.exe`. En Windows fem servir el *absolute path* i en Linux el *relative path*.
- `>nomarxiu.exe>resultat.txt` Crea un arxiu `resultat.txt` al subdirectori actual de tot el text o dades que ha imprès el programa `nomarxiu.exe` .
- `|` Serveix per indicar que la sortida d'un programa (esquerra del símbol) es l'entada per a un altre programa (dreta del símbol).
- `cat`, `type` Imprimeix un fitxer per pantalla . Exemple: `cat fitxer.txt | ./inversos`, `type fitxer.txt | inversos.exe` .
- `#include` Serveix per carregar les biblioteques següents.
- `<stdio.h>` Biblioteca que proporciona el nucli de les capacitats de entrada/sortida del llenguatge C (inclou la venerable funció printf).[Wikipedia].
- `<math.h>` Biblioteca que proporciona un conjunt de funcions per calcular les operacions i transformacions matemàtiques comuns.[1]

[1] `http://www.cplusplus.com/reference/cmath/`

- `<string.h>` Biblioteca que proporciona un conjunt de funcions per manipular cadenes de caràcters.
- `<stdlib.h>` Biblioteca que conté funcions per transformar cadenes en valors numèrics. Conté les `malloc`, `calloc` i més per a memòria dinàmica.
- `main(void)` o `main()` És la funció principal d'un programa.
- `void` Equivalen a el buit.
- `double x,y` Declarem les variables x i y com a tipus double. Aquesta és una especificació adequada per a treballar amb nombres reals. És el mateix `long float` que `double` utilitzen 8 bytes (64 bits). En canvi `float` només utilitza 4 bytes (32 bits).
- `unsigned int` Serveix per declarar alguna cosa com enter sense signe o positiu.
- `long long int` Serveix per declarar una variable amb memòria de 8 bytes (64 bits).(En el cas de Windows farem servir `%I64u` i en el cas de Linux `%lli` quan vulguem fer-les servir amb `printf()`).
- `scanf()`S'usa per llegir de pantalla (teclat). Aquesta funció està declarada al fitxer de capçalera `stdio.h`. Per exemple `scanf("%lf",&y);` llegeix el valor introduit com a 'long float' i el guarda a la memoria y.
- Quan ens referim a que és bon costum validar les dades de la funció `scanf` ens referim a que fiquem un `if` que si el valor introduït no és com volem mostri un missatge d'error. Exemple:
```
if(!scanf("%d",&grau)||grau<2||grau>GRAUMAX)
{printf("ERROR: grau il.legal!\")
;return 1;}.
```
- `%lf` Es refereix al format de lectura, que en aquest cas és de tipus double ('long float').[2]
- `%lli` Es refereix al format de lectura, que en aquest cas és de tipus double ('long long int')
- `%g` Es refereix al format d'escriptura genèric d'una variable real. La sortida serà de la forma més apropiada a la variable a escriure. Si volem que sigui un enter escriurem `%d`. `%u` es refereix a una variable `unsigned int`.
- `%s` Serveix per imprimir un `char` en la funció `printf()`.
- `printf("")` És una funció que s'usa per escriure per pantalla. El que s'escriu per pantalla és el que es troba entre cometes (""). En aquest cas `printf("El producte de %g per %g es %g \n",x,y,x*y)` escriu: El producte de `%g` per `%g` és `%g` `\n` ens adonem que en aquesta sentència hi trobem el format d'escriptura `%g` repetit tres vegades.Notem

[2]En la següent direcció hi ha totes les possibilitats per als *tags* que comencen per % i que guarden el lloc a les variables que es volen imprimir. En general, la forma és %[flags][width][.precision][length]specifier
`http://www.cplusplus.com/reference/cstdio/printf/`

també que darrera de la cadena a escriure hi ha tres variables, x, y i x*y (tantes com formats d'escriptura). En lloc del primer %g s'escriurà el contingut de la variable x, en lloc del segon %g el contingut de la variable y i en lloc del tercer %g el de x*y. Cal tenir sempre el mateix nombre de formats d'escriptura com de variables a escriure.

- `&x` S'usa per situar el valor llegit de pantalla a l'adreça de memòria de la variable x.
- `&x` Fa referencia a la direcció de memòria on esta guardada el valor de la variable x. Quan fem que una funció que definim com per exemple `eval(double *pdf){ *pdf=cos(y)-2; }` i després quan cridem la funció de `eval(&x)` estem passant el valor de x per referencia , en altres paraules em canviat el valor de x, sense crear noves variables.
- Si volem llegir el que conté una adreça de memòria tenim l'operador *.
- `int n,i,pot=1;`Declaració de les variables `n`,`i` i `pot` a tipus enter. La variable `pot` també es declara enter i la inicialitza en 1.
- `for(i=1;i<=n;i++)` El programa executarà repetidament les instruccions entre les claus que segueixen el `for`, variant els valors de `i` des de `1` (el valor inicial és el primer argument), mentre es compleixi la condició `i<=n` (el segon argument) i d'un en un (el tercer argument, `i++`, és el mateix que `i=i+1`).
- `while(x<1000)`$\{\cdots\}$ Repeteix una instrucció $\{\cdots\}$ mentre és compleix una condició (1º arg.) en aquest cas és repetira la instrucció mentre x sigui més petita que 1000. També podem ficar la instrucció sota de un `do` per que s'executi sempre una vegada i continuí si es compleixen les condicions ex: `do` $\{\cdots;\}$ `while(CONDICIONS);` .
- `if`$\{\cdots\};$ $\cdots$ `else`$\{\cdots\};$ Aquesta instrucció avalua una condició; si se satisfà, aleshores executa el que hi ha entre claus. La part `else` és opcional, i s'executarà en cas que no es satisfaci la condició del `if`. `if(x<0)`$\{\cdots\};$ $\cdots$`else`$\{\cdots\};$ En aquest cas, `else` s'executarà si el valor de x és positiu o zero. En el fons cada comparació retorna `1` o `0` depenent de si es compleix o no, i el que fa la instrucció `if` és mirar si el resultat de la comparació és diferent de zero. Podem afegir més d'una comparació amb les instruccions `&&`, `||` i `!` per demanar que es compleixin les dues avaluacions, una de les dues almenys, o per a negar-ne una, respectivament. També podem fer servir la funció `scanf()` per preguntar si es compleix o no una condició per exemple: `if(!scanf("%d",&grau)||grau<2||grau>GRAUMAX)`$\{\cdots;\}$ ens pregunta quin és el valor de la variable n i si es compleix s'executa la instrucció.
- `\n` Serveix per generar un salt de línia.
- `\t` Deixa un espai gran.
- `//` Serveix per fer comentaris d'una sola línia.
- `/*`$\cdots$`*/` Serveix per inserir comentaris de més d'una línia.

- `sqrt()` És una funció que calcula l'arrel quadrada d'un nombre `double`. Per a poder fer servir aquesta funció o qualsevol altre funció matemàtica cal fer el següent:
  - Al codi del programa s'ha d'incloure el fitxer de capçalera `math.h`.
  - Al compilar el programa cal afegir l'opció `-lm`: `gcc -Wall -o nomarxiu nomarxiu.c -lm` per carregar la biblioteca que conté les funcions matemàtiques bàsiques.
- `fabs()` La funció del valor absolut en C.
- `log()`,`log10()` i `log2()`: Funcions de logaritme neperià, logaritme decimal i logaritme binari respectivament.
- `pow()` Funció en C per a realitzar potencies. `pow(24,350)` és equivalent a 24^{350}.
- `<,>,<=,>=,==,!=` Així escrivim les condicions lògiques: menor que, major que, menor o igual que, major o igual que, igual que, diferent que.
- `double f(double x);` En aquesta sentència estem informant del *prototipus* de la funció que ens calcula $f(x)$ estem dient que a aquesta funció cal passar–li un paràmetre `double x` , i que retorna un valor `double` return y.
- `f(x)` Això és la crida a la funció des de dins del programa principal `main()`, i amb la qual li passem l'argument x.
- `return (sin(x)+10)/(x*x+1);` La funció ens retorna el valor $\frac{sin(x)+10}{x^2+1}$.
- `# define step 1.e-2, # define max 30, # define zero 1.e-6` Amb aquesta instrucció de preprocessador estem definint `step` com a constant simbòlica, i li estem donant el valor 0.01. De la mateixa manera, definim `zero` i `max` com a constants simbòliques, amb valors de 30 i 10^{-6}, respectivament.
- `&&`, `||`, `!` Operadors lògics respectivament: and, or, not. (Vist anteriorment).
- `double v[i], m[i][j];` Declara com a vector de `i` components `v`, i `m` com a matriu de `i` files i `j` columnes. Entenent que la primera component respectivament és `v[0]` i `m[0][0]`.Per tant les components dels vectors comencen amb l'índex 0. Quan definim un vector (per exemple `float v[3]`), la variable v guarda l'adreça de memòria on hi ha la component `v[0]`, `(v+1)` és l'adreça de `v[1]` , etc). Per tant per obtenir la component i-èsima podem fer `v[i]` o bé `*(v+i)`. Si volem demanar per pantalla les components d'un vector no cal utilitzar `&`, ja que, en general, `&v[i]` és el mateix que `(v+i)`. Les matrius nxm són vectors que tenen n*m posicions, i la posició (i, j) de la matriu és la posició i*n + j del vector.
- Quan utilitzem la direcció de memòria d'un vector `v[n]` en un programa el definirem per exemple com `imprimeixvector(double *v,int dim)` però quan cridem la funció ficarem `imprimeixvector(v,4)` en el cas concret.

- Si fem servir un programa en una funció com l'anterior pode declarar el vector com `double vect[4]`$=\{1,2,3,4\}$.

- A vegades quan definim funcions no solament utilitzem `return` quan a acabat la funció sinó que podem fer que retorni un valor o un altre , per exemple, si s'ha executat correctament `return 0;` o que si no ho fa `return 1;`. A més en cas negatiu és bon costum imprimir un missatge d'error.

- Les cadenes de caràcters es tracten com a *"vectors de caràcters que acaben amb el caràcter nul* $'\backslash 0'$ *"* i per tant el que sabem sobre vectors és aplicable. Tot i això tenen unes certes funcions específiques, que estan declarades al fitxer `string.h` i implementades a la biblioteca estàndard del C.

- `char` Serveix per declarar com a caràcter. Exemple:
  `char cadena1[]="Hola";`
  On veiem que no hem especificat la quantitat de caràcters de les cadenes. El que fa el compilador és reservar el nombre de caràcters que necessita més un (aquest últim és per a marcar el final de la cadena). Per tant en aquest cas ha reservat 5 *"bytes"* (1 per cada caràcter) per a `cadena1`.A vegades si utilitzem caràcters especials potser aquest és interpretat com a més d'un *byte*.

- La funció `sprintf()`, amb una sintaxi molt semblant a `printf()`, però amb un primer argument que és un apuntador a la cadena. I retorna la longitud de la cadena que s'ha creat. Exemple: `i=sprintf(cadena3,"Pi es aproximadament %g",PI);` guarda a la variable `cadena3` prèviament definida com a `char` la cadena entre cometes i inicialitza `i=28` (que és la mida en bytes de `cadena1`).

- `strlen()` Dóna la longitud de la cadena que hi entrem com argument. Més precisament, dóna quants caràcters hi ha entre la posició de memòria que li donem com a argument i el final de cadena $'\backslash 0'$ que troba. Sigui `cadena2` un `char` on `strlen(cadena2)` és 7 llavors `strlen(cadena2+n)` és la longitud de la `cadena2` començant pel caràcter 2.

- `strcpy()` Serveix per copiar cadenes, la funció tant es pot fer servir per a copiar un text literal entre cometes a una variable de cadena com per a copiar el contingut d'una variable de cadena a una altra. Exemples dels dos casos `strcpy(cadena1,"Hola");` i `strcpy(cadena2,cadena1);`. També podem fer servir `strncpy()` que accepta un tercer argument que és el nombre màxim de caràcters que volem copiar. En aquest cas no es copia el caràcter de final de cadena; per tant, si és el cas, l'haurem d'afegir manualment.

- `strcat()` Permet ajuntar dues cadenes de caràcters en una. El que fa és eliminar el caràcter de final de cadena de la primera i copiar el contingut de la segona a continuació. També existeix `strncat()` que afegeix un argument més on s'indica la longitud màxima de caràcters que es guardaran, si el nombre és major que el màxim llavors es guarda només el màxim. Tot i que després haurem d'afegir $'\backslash 0'$, és a dir, el caràcter de final de cadena.

- `strcmp()` Permet comparar cadenes. Si les dues cadenes són idèntiques, la funció retornarà el valor 0. Si les dues cadenes són diferents retornarà

un nombre negatiu o positiu depenent de si l'ordre en què entrem els arguments és el lexicogràfic (ordre del diccionari) o no. També existeix `strncmp()` que té un tercer argument que és el nombre màxim fins on volem comparar.

- `strchr()` Té dos paràmetres: una cadena de caràcters i un caràcter concret. Retorna la posició de memòria del primer caràcter de la cadena que coincideix amb el que entrem com a segon argument, si aquest existeix, i NULL altrament. També existeix `strrchr` que té els mateixos paràmetres però retorna la ultima posició de memòria on apareix el caràcter que busquem.

- `atof()`, `atoi()` Serveixen per transformar una cadena en el seu valor numèric, en el primer cas transforma la variable en `double` i en el segon cas `int`. Per ser utilitzades necessitem la capçalera `<stdlib.h>`

- `sizeof()` Retorna la mida en *bytes* d'un tipus de variable o bé la reservada per a un vector.

- `malloc()` Permet reservar un espai de memòria a un vector definit prèviament com apuntador.
  Exemple: primer declarem v com punter `int *v` després reservem el espai de memòria que vulguem per el vector
  `v=(int*)malloc(mida*sizeof(int);`
  (La variable `mida` esta definida com enter,tot i que per ser més correctes hauria de ser de tipus `size_t`). A la funció `malloc` li hem d'entrar la quantitat en bytes que volem reservar i retorna la posició de memòria que reservem, i que aquí guardarem la v , en aquest cas. Si no es pot reservar la memòria torna un valor de `NULL`, i per tant en aquest cas no podem utilitzar-la. A partir de llavors pot ser tractat com un vector com els definits directament a la declaració de variables. Aquesta funció i altres relacionades amb la memòria dinàmica estan a la llibreria `<stdlib.h>`.
  Exemple per declarar un vector de vectors(matriu) en memòria dinàmica:

```
...
double **v;
int i, n;
scanf("%d",&n)
v=(double**)malloc(n*sizeof(double*);
for(i=0;i<n;i++);{
v[i]=(double*)malloc(3*sizeof(double));}
...
```

- `free()`Serveix per quan ja no ens facin falta allibera les posicions de memòria no utilitzades generades amb `malloc`.

- `calloc()` Funciona de manera molt similar a `malloc` però el canvi és que en lloc de dir la mida que volem reservar en bytes, primer li diem la quantitat d'unitats que volem i després la mida de cada unitat.

- `realloc` permet redimensionar l'espai de memòria que ocupa un vector. Exemple: `v=realloc(v,novamida*sizeof(int));` On `novadimensio` és el nombre d'enters que volem que contingui el vector.

- `FILE* meufitxer` Defineix la variable `meufitxer` com a apuntador a un fitxer.
- `meufitxer=fopen("montoliu.txt","w")` Agafa un fitxer concret que es diu `montoliu.txt` en aquest cas, i fa que `meufitxer` apunti a ell. El `"w"` és un indicador de què volem fer amb el fitxer. En aquest cas significa que volem escriure (write) dins el fitxer. Si per el contrari volem llegir un fitxer usarem `"r"` (read).
- `fprintf(meufitxer,"%s\n", arg[1])` És el mateix que `printf()` amb la diferència que en lloc d'escriure a pantalla s'escriuen en el fitxer especificat.
- `fscanf()` Retrona el nombre de variables que ha llegit. Si `fscanf()` no pot llegir cap variable perquè hem arribat al final del fitxer, llavors retorna un valor impossible (-1), que és com esta definida la constant `EOF` (End Of File). `fscanf()` no entén de i columnes. Llegeix nombre rere nombre del fitxer. Si el programa espera quatre variables i a la línia actual només n'hi ha tres,continuarà llegint la fila següent.
- `fclose(meufitxer);` Retorna el control del fitxer al sistema operatiu. Mentre el tenim obert, el sistema no permetrà que cap altre procés el modifiqui.
- Hi ha un *fitxers* especial,que no cal obrir ni tancar que són `stdin`, `stdout` i `stderr`.
  - `stdin` Entrada estàndard, apunta al teclat. La funció `scanf()` llegeix del `stdin`, és el mateix fer `scanf(...)` que `fscanf(stdin,...)`.
  - `stdout` Apunta a la pantalla (sortida estàndard). ÉS el mateix fer `printf(...)` que `fprintf(stdout,...)`.
  - `stderr` També escriu per pantalla, però ho fa per un altere canal, que normalment es fa servir pels missatges d'error.

Referències

[1] *Llenguatge C.* Departament de Matemàtiques-Universitat Autònoma de Barcelona (Material de l'assignatura Eines informàtiques per a les matemàtiques)

[2] `https://www.tutorialspoint.com/c_standard_library/`