UD3 TIPUS DE DADES I PRIMERES PASSES EN C

Autor: Lluís València López
`http://www.xtec.cat/~lvalenci/cfgs.htm`

3.1 Tipus de dades

Les dades bàsiques que intervenen en els programes, per ser tractades i codificades de forma convenient, han de ser d'un determinat tipus (ja hem vist a la UD2 que no és el mateix que 4 bytes codifiquin un nombre enter o un nombre real, o que un byte codifiqui un nombre o un caràcter).
Les dades de les que parlarem en aquest capítol són dades elementals que només podran ser:

- nombres enters
- nombres amb coma flotant
- caràcters

En propers capítols ja parlarem de dades de tipus lògic, adreces de memòria i dades estructurades com els vectors/matrius o els registres.

Tipus de dades bàsics en C

En C, distingirem dos tipus de dades elementals:

NOMBRES ENTERS

Tipus	**Mida**	**Utilitat**
char	1 byte	Per enters sense signe (cal posar `unsigned char`) des de 0 fins 255. Per enters amb signe de -128 fins a 127 Per codificar caràcters
short	2 bytes	Per enters sense signe (unsigned short) de 0 a 65535 Per enters amb signe de -32768 fins a 32767
int o long	4 bytes	Sense signe 0.. 4 294 967 295 Amb signe -2 147 483 648 .. 2 147 483 647
long long	8 bytes	Només per alguns compiladors, no és estàndard. Per enters de fins a 20 xifres

NOMBRES EN COMA FLOTANT

Tipus	**Mida**	**Utilitat**
float	4 bytes	Correspon a l'IEEE-754 de 32 bits De ±1.4012985e-45 fins ±3.4028235e+38
double	8 bytes	Correspon a l'IEEE-754 de 64 bits De ± 5.0e-324 fins ± 1.7976931348623157e+308
long double	12 bytes	No estàndard. Més precisió

3.2. Variables

Les dades que manipulen els programes poden emmagatzemar-se en variables. Per poder fer-ho, cal donar un nom a cada variable.
En C, estem obligats a declarar totes les variables que fem servir segons aquest esquema:

```
tipus identificador;
```

on tipus és el tipus de variable a triar entre els que anirem estudiant (de moment char, short, ...) i identificador és qualsevol paraula construïda amb una lletra (de les ASCII,

res d'ñ, ç o accents) seguida de nombres, lletres o el caràcter "guió baix". Les lletres majúscules i les minúscules són diferents.
Sobretot: el nom d'una variable en C mai comença per un nombre.

Un cop declarada una variable, se li pot assignar un valor. Suposem que hem declarat la variable numero així:
```
int numero;
```
Per assignar un valor enter, ho podem fer directament:
```
numero=2509;
```
Encara que també el podem escriure en octal, cosa que indicarem posant un 0 davant:
```
numero=04715;
```
O en hexadecimal, la qual cosa quedarà clara perquè començarà per 0x:
```
numero=0x9CD;
```

Per assignar un valor decimal també hi ha diverses possibilitats. Declarada la variable numdecimal així: `float numdecimal`, podem assignar-li valors d'aquestes maneres:
```
numdecimal = - 23.51;
numdecimal = - 2.351e+1;
numdecimal = - 235.1E-1;
```

Per assignar un caràcter a una variable de tipus char (a la qual també li podem assignar enters), hem d'envoltar el caràcter entre cometes simples, per exemple:
```
char lletra='k';
```

Recordem: els caràcters sempre entre cometes simples o apòstrofs.

Si un caràcter no es pot imprimir (o encara que es pugui) l'indiquem amb una antibarra seguida del seu codi ASCII. Per exemple '\0' indica el caràcter que té codi ASCII 0, o '\13' un retorn de carro.

3.3. Operadors

Amb les dades, podem realitzar operacions.

OPERACIONS ARITMÈTIQUES

Nom operació	**Símbols que potser heu vist fins ara**	**Símbol en C**
Suma	+	+
Resta	–	–
Multiplicació	x, ·, *	*
Quocient (divisió entera)	div, /	/
Residu	mod, %	**%**
Divisió amb decimals	:, /, –	/
Potència	^, **	***biblioteca***

Podem veure que en C se'ns planteja un problema ja que un mateix símbol **(/) pot indicar dues operacions diferents**. Per discernir aquesta ambigüitat, mirarem com són els operands. Si tots dos són enters, es farà la **divisió entera**. Si un d'ells és decimal, es durà a terme la **divisió amb decimals**.

Per exemple, quan escrivim 41 / 3, en C es farà la divisió entera, és a dir, donarà 13 com a resultat. Mentre que 41 / 3.0 donarà 13.666667

En C podem abreujar algunes assignacions d'aquesta manera:
`a=a+b;` es pot escriure `a+=b;`
`a=a-b;` es pot escriure `a-=b;`
`a=a*b;` es pot escriure `a*=b;`
`a=a/b;` es pot escriure `a/=b;`

A més, disposem de l'operador postincrement/postdecrement (`a++` o `a--`) i preincrement/predecrement (`--a` o `++a`) que sumen o resten 1 a la variable a la qual s'apliquen, després o abans, respectivament, d'executar la instrucció de la qual formen part.

Un altre operador molt interessant de C és `sizeof(variable)` que s'aplica sobre una variable o sobre un tipus de dada. Si escrivim `sizeof(int)` estarem escrivint 4, perquè la mida d'un int en C és 4 bytes.

OPERACIONS DE RELACIÓ

Nom operació	**Símbols que potser heu vist fins ara**	**Símbol en C**
Són iguals	=	==
Són diferents	<>, ≠	!=
Més petit	<	<
Més petit o igual	≤	<=
Més gran	>	>
Més gran o igual	≥	>=

Quan establim una comparació entre dues dades, el resultat és una expressió lògica simple que pot ser CERT o FALS.
Podem construir expressions lògiques més complexes relacionant les més simples amb els operadors de la taula següent.

OPERACIONS LÒGIQUES

Nom operació	**Símbols que potser heu vist fins ara**	**Símbol en C**
Negació	**no**, ¯, ¬	**!**
I lògica	**i**, and, ∧	**&&**
O lògica	**o**, or, ∨	||

En general, les expressions lògiques les posarem dins un parèntesi, de manera que la següent és una expressió lògica complexa correctament formada:
`((!((c=='a')||(c=='b')))&&(n<=5))`
No ens ha de preocupar gastar més parèntesis del compte si guanyem concreció.

En C, el resultat d'una comparació és CERT (1) o FALS (0). També el resultat d'una expressió lògica més complexa acaba sent 1 o 0.

3.4. Esquema bàsic d'un programa en C

Els nostres programes en C tindran aquest esquema:

```
#include <stdio.h>

main()
{

}
```

El programa s'escriurà entre les claus. Cada instrucció del programa ha d'acabar en punt i coma (`;`).

La primera instrucció, include, és una instrucció del preprocessador, com totes les que comencen per #.
Vol dir que potser farem servir al nostre programa alguna funció declarada al fitxer de capçalera stdio.h (podem veure aquest i la resta de fitxers de capçalera al directori /usr/include de qualsevol linux).
stdio ve de: standard (std) input (i) output (o) i correspon a la biblioteca estàndard d'entrada/sortida. Una biblioteca és la compilació de certes funcions per tal d'utilitzar-les des d'altres programes. Les funcions de biblioteca (*library*) s'enllaçaran amb el nostre programa estàtica o dinàmicament en el moment de muntar (*linkar*) el programa. Ens pot anar be pensar que són com les DLL de Windows.

La primera de les funcions declarades a stdio.h que hem de conèixer és la que ens permet mostrar un missatge per pantalla: printf.
Per començar, la utilitzarem així:

```
#include <stdio.h>

main()
{
printf("El meu missatge\n");
}
```

Hem pogut veure com al final de la frase posàvem `\n`. Això és una manera d'anomenar un caràcter especial: el de salt de línia.
Aquest no és l'únic caràcter especial que podem incloure a printf, també podem utilitzar altres dels anomenats caràcters d'escape:

\n	Nova línia
\t	Tabulador
\b	Espai enrere
\r	Retorn de carro
\f	Començament de pàgina (per impressores)
\a	Xiulet
\'	Cometa simple
\"	Cometa doble
\\	Barra invertida
\xdd	Codi ASCII en hexadecimal (cada d representa un dígit)
\ddd	Codi ASCII en notació octal (cada d representa un dígit)

Analitza la sortida d'aquest programa:

```
#include <stdio.h>

main()
{
printf("1\t dilluns\n2\t dimarts\n3\t dimecres\n");
}
```

Provem ara d'entendre el resultat del programa:

```
#include <stdio.h>

main()
{
printf("Un int ocupa %d bytes\n",sizeof(int));
}
```

Veiem que s'imprimeix
Un int ocupa 4 bytes

A la frase que hem fet imprimir, s'ha substituït %d pel valor que hem posat després de la coma. Doncs bé, printf funciona així, quan posem dins la frase %d vol dir que en el seu lloc imprimirem un nombre i hem de facilitar aquest nombre afegint un altre argument a printf.
Observem que printf, en el primer exemple té un argument mentre que en aquest segon en te dos, el vermell i el verd, separats per una coma:

```
printf("Un int ocupa %d bytes\n",sizeof(int));
```

Analitza la sortida d'aquest codi:

```
#include <stdio.h>

main()
{
printf("char\t%d\n",sizeof(char));
printf("short\t%d\n",sizeof(short));
printf("int\t%d\n",sizeof(int));
printf("long long\t%d\n",sizeof(long long));
printf("float\t%d\n",sizeof(float));
printf("double\t%d\n",sizeof(double));
printf("long double\t%d\n",sizeof(long double));
}
```

Podem afegir comentaris als nostres programes: text que en realitat no forma part del programa sinó que el posem per recordar millor què fa cada cosa o per entendre millor una sentència.
El comentaris es poden escriure en una línia després d'una doble barra // o poden ocupar diverses línies si es tanquen entre /* i */

Observeu i reflexioneu sobre el resultat del programa:

```
#include <stdio.h>

main()
{
char caracter;
caracter='k'; //el codi ASCII de la lletra k és 107
printf("El codi ASCII de la lletra %c és %d\n",caracter,caracter);
}
```

Aquí, dins printf tenim dos paràmetres extres, encara que en aquest cas corresponen a la mateixa variable. El primer s'imprimirà com un caràcter (%c) i el segon com un enter (%d).

Regla d'or: si en un printf hi ha **x** %, també haurà d'haver **x** comes i **x** variables després de les cometes dobles.

Els tipus de dades que hem estudiat s'imprimeixen amb:
%d, %i → Nombres int en base 10, amb signe.
%u → Nombres enters sense signe.
%h → Nombres short en base 10 (%hu seria per unsigned short i %hd amb signe)
%hh → Per imprimir char com a nombre decimal (%hhu o %hhd, com el cas d'abans)
%x, %X → Nombres enters en base 16 amb les lletres minúscules/majúscules
%o → Nombres enters en octal
%lld → Nombres long long
%c → Caràcters
%f → Decimals de simple o doble precisió amb notació usual (per exemple 230000.0)
%e, %E → Decimals en notació científica (p.e. 23.0e+4 o 23.0E+4)
%g, %G → Tria la millor de les dues notacions anteriors.
%llf, %lle... → Per decimals long double
%p → Apuntadors del tipus que sigui

Ara bé, podem indicar més coses amb un expressió de format. En general aquestes expressions tindran aquest esquema:

%[*flag*][*ample*][.*precisió*]*tipus*

On tipus és algun dels que hem vist abans o dels que estudiarem en el futur (com ara %s).

Els claudàtors [ ] indiquen opcionalitat. És per això que fins ara no havien aparegut ni flags ni ample, no res entre el % i l'indicador de tipus de dada.

Flag	**Efecte**
–	Justifica a l'esquerra (per defecte és a la dreta)
+	Sempre posa el signe (+ o –) quan es tracta d'un nombre
Espai	Posa signe meys per negatius i un espai per positius
#	En enters, indica els octals començant-los per 0 i els hexadecimals per 0x. En decimals, obliga a escriure'ls amb la coma
0	Omple amb zeros els nombres justificats a la dreta

Ample correspon amb la longitud que reservem per imprimir la dada. Si la dada ha d'ocupar més espai no es fa cas d'*ample*.

Precisió indica el nombre de decimals en un nombre decimal o el màxim que ocuparà un text (ara, no s'imprimirà tot el text si no hi cap, no com amb ample). Amb nombres enters, omple amb zeros per l'esquerra si el nombre no és tan llarg com precisió (pràcticament, `%05d` equival a `%.5d`)

Ja sabem imprimir dades amb el format que ens convingui. Per fer els nostres programes més interessants, ens anirà bé veure el funcionament bàsic de la instrucció que serveix per recollir dades des del teclat.
Aquesta instrucció és scanf.
De moment, hem de creure que funciona així:

1. Al començament del programa es declara una variable
   Exemple: `int num;`
2. Es mostra un missatge que convidi a l'usuari a introduir la dada
   Exemple: `printf ("Escriu un nombre enter: ");`
3. Es recull la dada amb scanf, posant un & davant del nom de la variable.
   Exemple: `scanf("%d",&num);`

Si al "buffer" del teclat queden dades, ens pot passar que un scanf no s'aturi.
Donem alguns truquets per resoldre aquest problema:

- Quan llegim un char, deixar un espai abans del %c: `scanf(" %c", &car);`a
- Afegir després del nostre scanf el següent codi, que ja explicarem més endevant:
  `while (fgetc(stdin)!='\n");`

Veiem un exemple on deixem l'espai abans de %c:

```
#include <stdio.h>

main()
{
char lletra;
int dni;
printf("Introdueix el teu DNI sense la lletra: ");
scanf("%d",&dni);
printf("Introdueix la lletra (minuscula) del teu NIF: ");
scanf(" %c",&lletra);
printf("El teu NIF es %.8d-%c\n",dni,lletra-'a'+'A');
}
```

Observa com restant el valor de l'`a`minúscula i sumant el de la majúscula, es converteix una lletra minúscula qualsevol en la corresponent majúscula.

Conversió de tipus
Què passa si assignem una variable d'un tipus a una d'un altre? Hi ha vegades que és molt difícil saber-ho però hi ha unes altres que està clar el que passarà.
Mira de trobar algunes respostes executant el següent codi:

```
#include <stdio.h>

main()
```

```
{
int n;
char c;
float f;
double df;
long long ln;
n=-456;
c='f';
f=-1.45e-34;
df=1.45E-34;
ln=1234567890;
ln=ln*ln;
printf("n -->\t%d\nc -->\t%c\nf -->\t%g\ndf -->\t%g\nln --
>\t%lld\n",n,c,f,df,ln);
c=n;
f=df;
n=f;
printf("n -->\t%d\nc -->\t%c\nf -->\t%g\ndf -->\t%g\nln --
>\t%lld\n",n,c,f,df,ln);
df=1234567890123456.7002;
f=df;
ln=df;
n=df;
printf("n -->\t%d\nc -->\t%c\nf -->\t%f\ndf -->\t%f\nln --
>\t%lld\n",n,c,f,df,ln);
}
```

De vegades, si volem convertir una dada en un altra d'un altre tipus, convé fer un "cast", és a dir, posar el nou tipus davant entre parèntesi. Per exemple, `(float)3` és el mateix que `3.0`

Funcions de biblioteca

Quan hem d'utilitzar alguna funció que no és a la biblioteca estàndard, com ara les que són a la biblioteca d'operacions matemàtiques, cal incloure la biblioteca en les opcions d'enllaçat. Hem de localitzar la biblioteca (en el cas de la de les funcions matemàtiques és libm.a o libm.so) i treure el lib del començament i l'extensió quedant-nos amb el que anomenarem *nom* (en l'exemple ens queda només la lletra m) afegim –l*nom* a les opcions del gcc i llestos. Això no obsta de posar al començament del nostre fitxer font l'include corresponent.

El següent programa, compilat amb gcc posant l'opció –lm, calcularà la potència tercera de 2:

```
#include <stdio.h>
#include <math.h>
main()
{
printf("2 al cub val %.0f\n", pow(2,3));
}
```

"No es lo mismo"
En el següent exemple veiem una de les errades que es comenten en començar a fer programes. No hem de confondre un caràcter (entre cometes simples) amb una variable (no porta cometes).

```
#include <stdio.h>
main()
{
char c;
char f;
f=100;
c='f';
printf("El caracter %c te codi ASCII %d\n", c,c);
printf("El caracter %c te codi ASCII %d\n\n", f,f);
c=f;
printf("El caracter %c te codi ASCII %d\n", c,c);
printf("El caracter %c te codi ASCII %d\n", f,f);
}
```

Als primers printf, la variable c conté el caràcter f, que en ASCII és 102, mentre que f conté el nombre 100, que codifica la lletra d.
Als darrers, tant les variables c com f contenen el mateix.

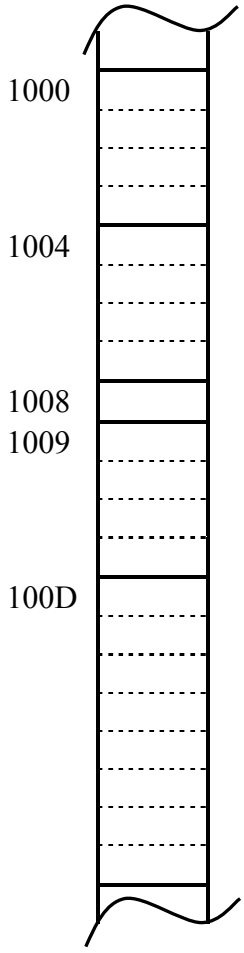
3.5. Ubicació de les variables a memòria

Quan declarem una variable, com ara
`int numero;`
estem reservant un cert nombre de bytes a partir d'una certa posició de memòria.

Si, posem per cas, un programa comença amb les següents declaracions:
int num1, num2;
char lletra;
float taxa;
double taxa_fina;
Si la primera de les variables s'emmagatzema a la posició 1000 (hexadecimal) de la memòria, i suposant que la resta de variables s'emmagatzemaran de forma contigua, podem fer el següent quadre:

Adreça de memòria	Nom variable
1000	num1
1004	num2
1008	lletra
1009	taxa
100D	taxa_fina

Que correspondria a aquest dibuix de la memòria:



Així, si cada variable té el seu lloc a la memòria, ha de ser possible demanar al programa quina és aquesta posició.

En efecte, només cal precedir el nom de la variable d'un & i llestos.
Vegem un exemple:

```
#include <stdio.h>

main()
{
int num;
printf("La variable num és a la posició de memòria %X\n",&num);
}
```

La sortida del qual pot ser una cosa així:

La variable num és a la posició de memòria FEEE9BE4

Observem que les adreces són de 4 bytes, tot i que nosaltres, als exercicis només posarem 2, per simplificar.

Adonem-nos també que la posició de memòria d'una variable no la podem triar ni modificar.

3.6. Variables de tipus apuntador

Si volem emmagatzemar adreces de memòria, ens caldrà un nou tipus de dada, de fet tota una col·lecció de tipus de dada ja que juntament amb el valor de l'adreça, també ens interessarà disposar de quina mena de dada és adreça, per saber si s'estan reservant 1, 2, 4, 8 o 12 bytes.

Una variable per emmagatzemar adreces s'anomea ***apuntador*** (en anglès pointer).

Per declarar una variable que emmagatzemi l'adreça d'un int, hauré d'escriure int * davant el nom de la variable.

La declaració
`int * apint;`
es llegeix *apint és un apuntador a un enter* i serveix per declarar la variable *apint*.

El següent tros de codi:

```
#include <stdio.h>
main()
{
char lletra;
int nombre;
char * apchar;
int * apint;

apchar=&lletra;
apint=&nombre;

printf("Variable\tAdressa\nlletra  \t%X\nnombre \t%X\n",apchar,apint);
printf("apchar  \t%X\napint   \t%X\n",&apchar,&apint);
}
```

Produeix la següent sortida:

```
Variable        Adressa
lletra          FEEE6397
nombre          FEEE6390
apchar          FEEE638C
apint           FEEE6388
```

L'explicació de perquè no es correspon al que havíem explicat és doble:
En primer lloc, en molts compiladors, la primera variable que s'ubica a la memòria és la darrera que es declara (en aquest cas apint) i des d'aquesta fins la primera. Així, primer s'ubica la variable apint, a la posició FEEE6380 i d'aquí cap amunt.
En segon lloc, C no té cap obligació de posar les variables en posicions consecutives de memòria.
Així, podem veure com a *apint* li toca l'adreça 6388, a *apchar* la que va 4 llocs després (clar, *apint* ocupa de la 6388 fins la 638B), és a dir la 638C, a *nombre* 4 llocs més, la 6390 i en canvi, a *lletra*, que li hauria de correspondre la 6394 perquè *nombre* és int i per tant ocupa 4 bytes, li assigna la 6397 (la que mai no li podria haver assignat és la 6391, o 6392, que pertanyen a *nombre*)

Analitzeu el següent codi de programa mirant de trobar explicació al que passa:

```
#include <stdio.h>
```

```
main()
{
char lletra;
int nombre;
char * apchar;
int * apint;

apchar=&lletra;
apint=&nombre;

printf("apchar val: %X\napint val:%X\n",apchar,apint);

apchar++;
apint++;

printf("apchar val ara: %X\napint val ara: %X\n",apchar,apint);
}
```

Observareu que no té el mateix efecte sumar 1 a un apuntador a int que a un apuntador a char, perquè un int i un char no ocupen el mateix nombre de bytes de memòria.
D'això es diu ***aritmètica d'apuntadors***.

Tal com donada una dada podem demanar la seva adreça mitjançant l'operador &, a una adreça se li pot demanar quina dada conté amb l'operador *.
D'aquesta manera, `* apchar` vol dir *el contingut de l'adreça de memòria que guarda la variable apchar*.

Abans d'executar el següent tros de codi, mira de deduir el valor de la variable nombre quan la imprimim:

```
#include <stdio.h>

main()
{
int nombre;
int * apint;

nombre=4;
apint=&nombre;
*apint+=10;

printf("nombre val: %d\n ", nombre);
}
```

3.7. Introducció a les estructures de control: if

En tots els exemples que hem posat fins ara hi havia només dos tipus de sentències en els nostres programes: assignacions i entrades/sortides.
Això ha limitat força el que aquests programes poden demanar.
Amb la intenció de poder anar resolent problemes més interessants, introduïm ara la sentència de control if.
Es fa servir per protegir una part del codi amb una condició, de manera que si aquella condició es dona, s'executa el codi i altrament no s'executa.

Inicialment, la seva sintaxi és:

```
if (condició)
     {
     ...
     }
```

I per utilitzar-la, posem per exemple un programa que hagi de dir si un nombre introduït per l'usuari es parell:

```
#include <stdio.h>

main()
{
int num;
printf("Introdueix un nombre enter: ");
scanf("%d",&num);
if (num%2==0)
     {
     printf("El nombre %d és parell\n", num);
     }
}
```

També pot anar seguida de la clàusula *else*, que servirà per escriure el codi que volem que s'executi si no es dóna la condició:

```
#include <stdio.h>

main()
{
int num;
printf("Introdueix un nombre enter: ");
scanf("%d",&num);
if (num%2==0)
     {
     printf("El nombre %d és parell", num);
     }
else
     {
     printf("El nombre %d és senar", num);
     }
printf("\n");
}
```

Exercicis:

1. Completeu les línies de punts amb una variable del programa perquè no digui mentides:

```
#include <stdio.h>
main()
{
int a,b,c, d;
a=2; b=3; c=5; d=6;
printf("%d + %d = %d \n", a, b, .....);
printf("%d x %d = %d \n", a, b, .....);
printf("%d / %d = %d \n", d, b, .....);
}
```

2. Demaneu dos nombres enters i doneu: la seva suma, la seva resta, el seu producte, el quocient de la seva divisió entera, el residu de la divisió entera, la seva divisió amb decimals.

3. Sabent que el codi següent ens mostra l'any i el mes de la data actual, fes un programa que demani l'any i mes de naixement i digui l'edat.

```
#include <stdio.h>
#include <time.h>
main()
{
time_t temps;
struct tm *tmPtr;
int any_actual, mes_actual;

temps = time(NULL);
tmPtr = localtime(&temps);
any_actual=tmPtr->tm_year+1900;
mes_actual= tmPtr->tm_mon+1;
printf("Estem al mes %d de l\'any %d\n",mes_actual,
any_actual);
}
```

Examen tipus:

L'examen d'aquesta unitat constarà de dues parts:
- Una prova teòrica
- Una prova de programació amb tres exercicis

La nota es calcularà com si la prova teòrica fos un exercici més (25% part teòrica, 75% programes).
L'enunciat de la part teòrica podria ser:

Considereu el següent programa en C:

```
#include <stdio.h>

main()
{
int a;
int b;
long double c;
int * d;
a=7; //linia 1
b=24; //linia 2
b+=a; //linia 3
c=b/a; //linia 4
c=a; //linia 5
c=b/c; //linia 6
d=&b; //linia 7
(*d)++; //linia 8
a=--b; //linia 9
}
```

a) Empleneu la taula variables-memòria:

Variable	Adreça de memòria
a	1000
b	
c	
d	

b) Ompliu la memòria després de l'execució de la instrucció que s'indica:

	1	2	3	4	5	6	7	8	9
...	...	...	...	...	...	...	...	...	...
1000									
1001									
1002									
1003									
1004									
1005									
1006									
1007									
1008									
1009									
100A									
100B									
100C									
100D									
100E									
100F									
1010									
1011									
1012									
1013									
1014									
1015									
1016									
1017									
1018									
1019									
101A									
101B									
101C									
101D									
...	...	...	...	...	...	...	...	...	...

La part de programació tindrà un exercici bàsic, un altre amb un xic més de complicació i un tercer on calgui fer servir *if*:

Heu de fer tres programes en C.

1. Aquest programa, mostrarà el següent missatge:
   `Introdueixi el preu d'un article:`
   Si, posem per cas, l'usuari introdueix la quantitat 34.32, el programa acabarà la seva execució responent:
   ```
   Preu:       34.32
   Iva:         5.49
   Total:      39.81
   ```
   Considerarem que l'IVA és el general: del 16%.

2. Aquest programa, mostrarà el següent missatge:
   `Introdueixi el preu de tres articles:`
   Si, posem per cas, l'usuari introdueix les quantitats 34.32, 1225.23 i 67.3, el programa acabarà la seva execució responent:
   ```
   Preu:     1326.85
   Iva:       212.30
   Total:    1539.15
   ```
   On hi figura la suma dels tres preus, l'IVA del 16% i el total.

3. Ara, considerarem un comerç que ven articles amb tipus general d'IVA (16%) i de tipus reduït (7%). En aquesta ocasió, es demanarà a l'usuari el preu i el tipus d'IVA (`g` o `r`) per tal de calcular adequadament el total. El programa mostrarà tres cops el missatge:
   `Introdueixi el preu d'un article seguit de g o r:`
   Si, per exemple, l'usuari introdueix 34.32 r, 1225.23 g i 67.3 r, el programa mostrarà:
   ```
   Preu:     1326.85
   Iva:       203.15
   Total:    1530.00
   ```