Llenguatges de Programació

Curs 2006 – 07 / Bloc #2 - Pràctica #1

Notes importants:

- Per fer els exercicis d'aquesta pràctica necessiteu haver llegit el **tema 5 i fins a l'apartat 4.1 (Pas de registres per referència) del tema 6 del manual de C**. **Fixeu-vos bé** en tots els **exemples** que hi ha explicats en el manual.
- Recordeu que si no us dóna temps d'acabar tots els exercicis durant la sessió de pràctiques és **molt important** que els **acabeu després** pel vostre compte.

Exercici #1> La finalitat d'aquest exercici tan simple és triple:

1. Perdre la por a usar el recurs de DEBUGGING.
2. Tenir una idea dels "salts" que fa un programa modular (dividit en varies funcions).
3. Veure de forma ben senzilla la diferència que hi ha entre el pas de variables per valor i per referència.

Creeu un projecte buit i afegiu-li el següent codi. Debugueu-lo de principi a fi i comproveu el flux que segueix, així com el valor de totes les variables.

```
//--------------------------------------------------------------------------------
// Aquí van les llibreries que facin falta
#include <stdio.h>

// Aquí van les declaracions de les funcions. NO LES OBLIDEU!!!
void funcio1(void);
int funcio2(void);
void funcio3(int);
void funcio4(int*);


//--------------------------------------------------------------------------------
void main(void)
{
  int x,y=37;

  funcio1();
  x=funcio2();
  printf("Dins x, ara hi tenim el valor que ha retornat la funcio: x=%d\n\n",x);
  funcio3(y);
  printf("Pero quan surto, la variable original NO s'ha modificat: y=%d\n\n",y);
  funcio4(&y);
  printf("Ara la variable original ha canviat: y=%d\n\n",y);
  printf("Espero que aquest exemple us sigui util!!!\n\n ");
}

//--------------------------------------------------------------------------------
void funcio1(void)
{
  printf("Aquesta funcio no rep ni retorna parametres.\n\n");
}

//--------------------------------------------------------------------------------
int funcio2(void)
{
  int n;
```

```
  printf("Introduiu un nombre siusplau: ");
  scanf("%d",&n);
  printf("Ara retornem aquest nombre a la funcio main...\n\n");
  return(n);
}


//-------------------------------------------------------------------------------------
void funcio3(int y) // Pas de peràmetre per valor
{
  printf("Aquesta funcio rep un parametre que te per valor %d.\n\n",y);
  y=y+10;
  printf("Dins la funcio vario el valor del parametre i ara val %d.\n\n",y);
}

//-------------------------------------------------------------------------------------
void funcio4(int *Py) // Pas de peràmetre per Referència
{
  printf("Aquesta funcio NO rep un valor, sino l'adressa d'una variable: Py=%x\n",Py);
  printf("Si accedim al contingut d'aquesta adressa, el podem modificar...\n\n");
  *Py=*Py+10;
}.
```

' Per entrar dins de la funció utilitzeu l'opció ***Step Into*** (F11) del Debugger quan esteu a sobre de la línia del codi que fa la crida a la funció.

' En mode Debugger, podem veure el valor de les variables locals de cada funció. Apareixen a la pestanya ***Locals*** (*figura 1*). Al quadre ***Context*** ens apareix el nom de la funció a la qual pertanyen les variables i se'ns permet canviar a qualsevol altra funció que tingui variables en memòria. Utilitzeu aquest recurs per comprovar que quan entrem a la funció ***valor_abs*** tenim definides dues variables 'x', 'y', però associades a contextos diferents. Fixeu-vos que després de retornar al ***main( )***, les variables que pertanyen a la funció ***valor_abs*** han deixat d'existir. Són variables locals a la funció.

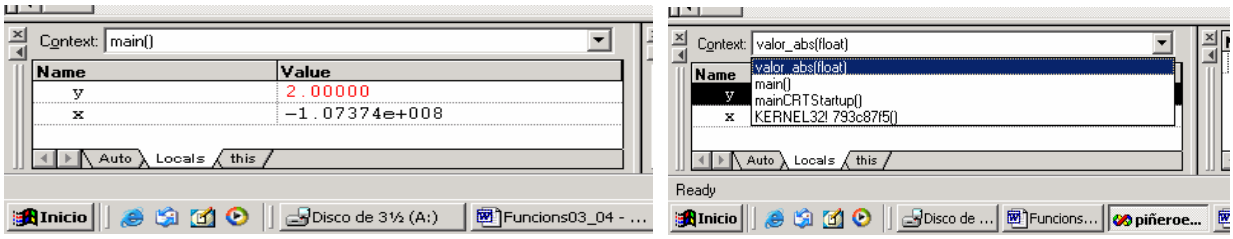

Figura 1. Variables locals

Exercici #2> Definiu una funció (digueu-li **`ObtenirNombreValid`**) que faci el mateix que el codi en negreta i modifiqueu el programa per què cridi a la funció en substitució del codi en negreta.

```
void main( )
{
     int x,y;

     y=5;

     do {
         printf ("Introdueix un número entre 1 i 10\n");
         scanf("%d",&x);
     } while ((x<1) || (x>10));
```

```
        if(x==y)
            printf(“Has encertat!!!”\n);
    }
```

Fixeu-vos que ara el codi queda més entenedor...

Exercici #3> Suposeu que sou espies al servei d'alguna súper-potència que esteu infiltrats dins territori enemic. Periòdicament aneu enviant informació però, donat que hi ha la possibilitat que us interceptin els missatges, us heu inventat una manera de codificar la informació que passeu:
Cada caràcter `c` del missatge, el desplaceu `d` posicions dins la taula de caràcters ASCII, on `d` és un paràmetre del vostre codificador que depèn del dia del mes que envieu el missatge, és a dir, `d={1,2,···,31}`. Així per exemple, si `c=='w'` i `d==31`, el caràcter codificat serà `'-'`. Per descodificar feu el procés invers.

Creeu dues funcions:

- `EncriptaMissatge(char*,int);`
- `DesEncriptaMissatge(char*,int);`

a les quals passareu un string per referència i un enter per valor, i efectuareu l'encriptació/desencriptació del missatge, segons hem explicat.

Un cop tingueu les funcions fetes, testegeu-les com vosaltres vulgueu.

Exercici #4> Declareu una variable entera (a) i un punter a enter (p_a). Inicialitzeu l'enter a 2 i el punter amb l'adreça de a.

A. Imprimiu els següents valors per pantalla: el valor de a, l'adreça de a, el valor de p_a, el valor de la variable a que apunta p_a, l'adreça de p_a. Useu el format %x (tipus hexadecimal) per a les adreces de memòria.

B. Useu el **watch** del ***Debugger*** per a accedir a tota la informació anterior. Comenceu a Debbugar. Afegiu al **watch** les següents expressions: a, p_a, &a (*figura 1*). Les expressions que són adreces de memòria d'alguna variable contenen 2 valors: l'adreça (el valor que es veu a la *figura 1)* i el valor contingut en aquesta adreça. Useu el + per a desplegar tots els valors (*figura 2*) . Afegiu l'expressió *p_a i debuggeu fins el final per a comprovar que els valors coincideixen amb els que heu imprès per pantalla.

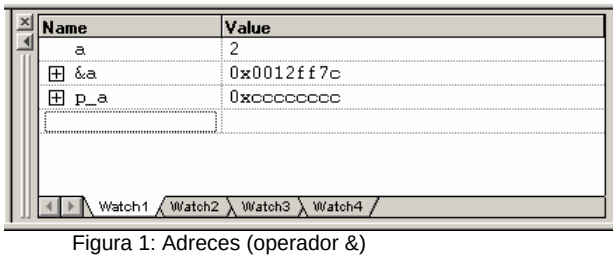

Figura 1: Adreces (operador &)

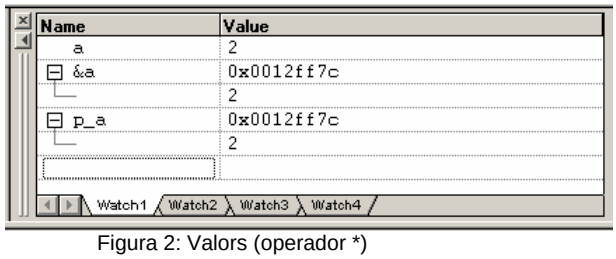

Figura 2: Valors (operador *)

Exercici #5> A partir del següent codi, que és similar a l'exemple de la secció 2 del manual, substituïu la part en negreta per una funció *intercanviar* que intercanviï el valor de dues variables. Per fer aquesta funció heu d'utilitzar el pas de paràmetres per referència a través

d'apuntadors. Un cop feta, utilitzeu el debugger per veure com evoluciona el valor de les variables del programa principal i de la funció.

```
#include <stdio.h>

void main ()
{
	int n1, n2, n3;
	int tmp;

	printf ("Introdueix tres nombres enters: \n");
	scanf ("%d %d %d", &n1, &n2, &n3);
	if (n1> n2)
	{
		tmp = n1;
		n1 = n2;
		n2 = tmp;
	}
	if (n1 > n3)
	{
		tmp = n1;
		n1 = n3;
		n3 = tmp;
	}
	if (n2 > n3)
	{
		tmp = n2;
		n2 = n3;
		n3 = tmp;
	}
	printf ("Els tres nombres ordenats són: %d %d %d\n", n1,
	n2, n3);
}
```

Exercici#6> Creeu un nou tipus de variable que es digui `FullAgenda` i que contingui els camps: `Nom`, `Cognom1`, `Cognom2`, `Adresa`, `Telefon` i `Email` amb els tipus més apropiats. Declareu un vector anomenat `Agenda` de 5 posicions i de tipus `FullAgenda`. Recorrereu aquest vector i emplenareu cada casella cridant a la funció que té per capçalera:

```
EmplenaFullAgenda(FullAgenda*);
```

i que haureu de crear vosaltres.

m Tingueu en compte que l'àmbit del nou tipus, `FullAgenda`, ha de ser tot el programa per tant no l'heu de posar dins el `main`, sinó fora. Utilitzeu `typedef`!!!

Exercici que heu d'entregar abans de la sessió vinent a través del Campus Virtual

En aquest exercici no haureu de picar gaire codi nou, sinó més aviat reorganitzar el que ja teniu fet: agafeu la pràctica que vareu entregar al final del bloc#1, en què només hi ha una funció (el `main`), i subdividiu el codi en funcions. No hi una única manera de fer això, però heu de pensar que utilitzar funcions serveix per:

- Fer el codi més entenedor.
- No repetir porcions de codi que facin el mateix.

X RECORDEU que **NO** podeu fer servir variables **GLOBALS**!!

 NOTA: ***Aquest programa el podreu aprofitar per pràctiques posteriors.***