

FONAMENTS D'ORDINADORS

TEMA4: Funcions

Manel Guerrero

[H14] Funcions

```
#include <stdio.h>

int sumar(int a, int b);

main(){
    int n = 3, m = 2;
    printf("%d + %d = %d\n",
           n, m, sumar(n,m));
}

int sumar(int a, int b) {
    return a+b;
}
```

- Una funció es un bloc de codi que opera amb uns paràmetres d'entrada i que pot retornar un valor de tipus elemental o structs (no vectors).
- Abans del main(), els prototipus: [tipus_ret nom_func(tipus_par1 nom_par1, ..., tipus_parN, nom_parN);] i després del main(), el cos de la funció.
- Si no retorna res tipus_ret és "void" i no fa servir el "return".
- Paràmetres formals: 'a' i 'b'. Paràmetres reals: 'n' i 'm'.
- int main(int argc, char *argv[]);
 - (Fora de temari)

Variables globals i locals

```
#include <stdio.h>

int foobar;

int sumar(int a, int b);

main(){
    int n = 3, m = 2;
    printf("%d + %d = %d\n",
        n, m, sumar(n,m));
}

int sumar(int a, int b) {
    int c = a + b;
    return c;
}
```

- En aquest exemple les variables 'n' i 'm' només existeixen dins del main() i les variables 'a', 'b', 'c' només existeixen dins de la funció sumar(). Són variables locals.
- La variable 'foobar' no es declara dins de cap funció i pot ser utilitzada a tot arreu. És una variable global.
- El prototipus d'una funció i la capçalera d'una funció són exactament iguals.

Funcions: problemes

- Col·lecció de problemes:
 - #2: Funció double factorial(int n) ($0! = 1$, $1! = 1$, $n > 1$, $n! = 1 \times 2 \times 3 \times \dots \times n$)
 - #3: Funció double potencia(float base, int exp) para exp enter positiu.

[H15] Funcions: problemes (2)

- #4 de la col·lecció: Amb la funció factorial, calcular sumatori de $1/i!$ per 'i' de 0 fins a 'N'.
- #5 (3era part) Recupera la funció factorial ($0! = 1$. $1! = 1$. $n > 1$, $n = 1 \times 2 \times 3 \times \dots \times n$) i afegeix una funció potencia i fes que ara retorni e^x (sumatori de i de 0 a infinit de $x^i/i!$). Enlloc d'infinit, que li puguis dir el número d'iteracions.
- (Aquests dos estan a `t4_e_power_x.c`)
- [Fora de temari] Calcular factorial de manera recursiva.
 - $\text{factorial}(0) = 1$. $\text{factorial}(1) = 1$.
 - per $n > 1$, $\text{factorial}(n) = \text{factorial}(n-1) * n$.

[H16] Funcions: problemes (3)

- Col·lecció de problemes:
 - #5 (1era i 2ona part) Usant factorial i potencia calcular $\sin(x)$, $\cos(x)$. (Aquest està a t4_sin_cos.c)
 - #6 Canvi de base
- Dubtes.
 - I si no algun de la col·lecció addicional.

[H17] Problema amb les funcions

- Aquest programa "t4_swap1.c" intercanvia els valors de les variables 'a' i 'b' amb ajut d'una variable auxiliar 'aux'

```
$ ./t4_swap1
```

```
a=2, b=3
```

```
a=3, b=2
```

```
$
```

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

```
main() {
```

```
    int a=2, b=3, aux;
```

```
    printf("a=%d, b=%d\n", a, b);
```

```
    aux = a;
```

```
    a = b;
```

```
    b = aux;
```

```
    printf("a=%d, b=%d\n", a, b);
```

```
}
```

Problema amb les funcions (2)

- Però si fem l'intercanvi en una funció “swap()” (t4_swap2.c), veiem que el intercanvi de valors es produeix dins la funció però al retornar al “main()” els valors no estan intercanviats.

```
$ ./t4_swap2
```

```
a=2, b=3
```

```
a=2, b=3
```

```
a=3, b=2
```

```
a=2, b=3 UPS!
```

```
$
```

```
#include <stdio.h>
void swap(int a, int b) {
    int aux;
    printf("a=%d, b=%d\n", a, b);
    aux = a; a = b; b = aux;
    printf("a=%d, b=%d\n", a, b);
}
main() {
    int a=2, b=3;
    printf("a=%d, b=%d\n", a, b);
    swap(a,b);
    printf("a=%d, b=%d UPS!\n", a, b);
}
```

Problema amb les funcions (3)

- Però si fem que les variables siguin dos posicions d'un vector d'enters si que funciona (t4_swap3.c).

```
$ ./t4_swap3
```

```
v[0]=2, v[1]=3
```

```
v[0]=2, v[1]=3
```

```
v[0]=3, v[1]=2
```

```
v[0]=3, v[1]=2
```

```
$
```

```
#include <stdio.h>
void swap(int v[]) {
    int aux;
    printf("v[0]=%d, v[1]=%d\n", v[0], v[1]);
    aux = v[0]; v[0] = v[1]; v[1] = aux;
    printf("v[0]=%d, v[1]=%d\n", v[0], v[1]);
}
main() {
    int v[2]={2,3};
    printf("v[0]=%d, v[1]=%d\n", v[0], v[1]);
    swap(v);
    printf("v[0]=%d, v[1]=%d\n", v[0], v[1]);
}
```

Problema amb les funcions (4)

- Però si fem que les variables siguin dos camps d'un struct tampoc funciona (t4_swap4.c).

```
$ ./t4_swap4
```

```
s.a=2, s.b=3
```

```
s.a=2, s.b=3
```

```
s.a=3, s.b=2
```

```
s.a=2, s.b=3 UPS!
```

```
$
```

```
#include <stdio.h>
typedef struct { int a, b; } t_s;
void swap(t_s s) {
    int aux;
    printf("s.a=%d, s.b=%d\n", s.a, s.b);
    aux = s.a; s.a = s.b; s.b = aux;
    printf("s.a=%d, s.b=%d\n", s.a, s.b);
}
main() {
    t_s s={2,3};
    printf("s.a=%d, s.b=%d\n", s.a, s.b);
    swap(s);
    printf("s.a=%d, s.b=%d UPS!\n", s.a, s.b);
}
```

¿Perquè? (1) Memòria RAM

- La memòria RAM d'un I32 pot arribar a adreçar 4Gb (2^{32} bytes).
- La memòria és una seqüència indexada de bytes numerats començant pel '0'.
- Allà es guarda tant el codi del programa com les seves variables.
- Quan es declara una variable es reserven les posicions de memòria necessàries per emmagatzemar-la:
 - char: 1 byte; short: 2 bytes; int, float: 4 bytes; double: 8 bytes)
 - vector: dimensió x mida 1 element
 - struct: suma mides dels diferents camps

- L'@ de memòria és la posició de memòria més baixa (petita) de tots els bytes que ocupa la variable.
- Com quedaria la memòria en aquest cas?

```
typedef struct {  
    short s;  
    int i;  
} ttipo;  
main() {
```

```
    int x; ttipo t; int v[4];
```

- Això no és sempre exactament així perquè el compilador fa optimitzacions i alineacions.

¿Perquè? (2) Punters

```
#include <stdio.h>
main() {
    int a=2, b=3, *p_i;
    p_i = &a;
    printf("apunta a %d\n", *p_i);
    a = 7;
    printf("apunta a %d\n", *p_i);
    p_i = &b;
    printf("apunta a %d\n", *p_i);
}
```

apunta a 2

apunta a 7

apunta a 3

- Un punter no és res més que un enter que conté la posició de memòria d'una altra variable.

Posició Var Val

.....60 p_i68

.....64 b 3

.....68 a 7

Pas de paràmetres per referencia

- La solució és passar a “swap()” punters a les variables que volem que es modifiquin. A això se li diu “pas de paràmetres per referencia”.

```
./t4_swap5
```

```
a=2, b=3
```

```
Posicions de memoria:
```

```
&a=BF96D748, &b=BF96D74C
```

```
a=2, b=3
```

```
a=3, b=2
```

```
a=3, b=2
```

```
$
```

```
#include <stdio.h>
void swap(int *a, int *b) {
    int aux;
    printf("Posicions de memoria:\n");
    printf("&a=%X, &b=%X\n", a, b);
    printf("a=%d, b=%d\n", *a, *b);
    aux = *a; *a = *b; *b = aux;
    printf("a=%d, b=%d\n", *a, *b);
}
main() {
    int a=2, b=3;
    printf("a=%d, b=%d\n", a, b);
    swap(&a, &b);
    printf("a=%d, b=%d\n", a, b);
}
```

Més exemples de punters

```
int *p, *q; /* punters a
enters*/

p = &a; /* 'p' apunta a 'a'
('p' = l'@ en la que està
'a') */

*p = 2; /* assignar 2 a la
variable apuntada per 'p' */

int i; i = *p; /* assignar el
valor de la variable apuntada
per 'p' a 'i' */

q = p; /* 'q' apunta a allò
que apunti 'p'. */

int *p=0; p++; // p == 4
```

- '&' = “punter a...” = “adreça on està emmagatzemat ...”
- '*' = “valor d'allò apuntat per ...”. = “valor en aquesta direcció de memòria”.
- “int *p;” = “allò apuntat per 'p' és un enter” = “'p' és un punter a enters”.
- Aritmètica de punters: ++, --, + i - : cada unitat es tradueix a tants bytes com ocupi el tipus al què apunta.

[H18] Repàs de pas de paràmetres per referència

```
#include<stdio.h>
void swap(int *p_a, int *p_b) {
    int aux;
    aux = *p_a;
    *p_a = *p_b;
    *p_b = aux ;
}
main() {
    int a,b;
    scanf("%d %d", &a,&b);
    swap(&a,&b);
    printf("%d %d\n", a, b);
}
```

- Fins ara fèiem el que s'anomena “pas de paràmetres per valor”.
- Quan cridem una funció els paràmetres real i els formals ocupen posicions de memòria diferents! Els formals són un duplicat. Per això modificant els formals no modifiquem els reals!
- Dins de “swap()” el valor del punter no es pot modificar, però el de la variable apuntada per el punter si.

Repàs paràmetres per referència (2)

```
#include<stdio.h>
void swap(int v[]) {
    int aux;
    aux = v[0];
    v[0] = v[1];
    v[1] = aux ;
}
main() {
    int v[2];
    scanf("%d %d", &v[0],&v[1]);
    swap(v);
    printf("%d %d\n", v[0], v[1]);
}
```

- El mateix programa però ara amb vectors.
- Quan tu passes per paràmetre un vector en realitat estàs passant un punter a la seva posició '0'.
- Per tant un vector no es pot “passar per valor”. Si el passes per paràmetre a una funció que el modifica, quan retornis el vector seguirà modificat.

Repàs paràmetres per referencia (3)

- Fer t4_factorial_pref.c
 - void factorial(int num, int *res);
- Fer t4_divisio_pref.c
 - void divisio(int n, int d, int *p_q, int *p_r);
- Fi de repas: Ara fer t4_coordenades com a fil argumental de les següents dues transparències sobre punters i structs.

Punters i structs

```
#include <stdio.h>
typedef struct {
    int x;
    int y;
} tpunt;
void invertir_coordenades(tpunt *p);
main() {
    tpunt p = {1,2};
    printf("%d %d\n", p.x, p.y);
    invertir_coordenades(&p);
    printf("%d %d\n", p.x, p.y);
}
void invertir_coordenades(tpunt *p) {
    int aux = p->x;
    p->x = p->y;
    p->y = aux;
}
```

- Quan combinem structs amb punters la cosa es complica.
- Un camp d'un element 'e' apuntat per un punter 'p': "p->e".
- Tot i que també podríem obtenir primer la variable apuntada i després l'element: "(*p).e". Parèntesis no opcionals.

- Aquests codis són equivalents:

```
aux = p->x;
```

```
p->x = p->y;
```

```
p->y = aux;
```

i

```
aux = (*p).x;
```

```
(*p).x = (*p).y;
```

```
(*p).y = aux;
```

Punters i structs (2)

```
#include <stdio.h>
typedef struct {
    int x;
    int y;
} tpunt;
void introduir_coordenades(tpunt *p);
main() {
    tpunt p;
    introduir_coordenades(&p);
    printf("%d %d\n", p.x, p.y);
}
void introduir_coordenades(tpunt *p) {
    scanf("%d %d", &p->x, &p->y);
}
```

- “Rizando el rizo”. Si necessitem un punter a un element 'e' d'un struct apuntat per un punter 'p': “&p->e”. Wow!
- Però ja està, en aquesta assignatura ja no compliquem més les coses.

[H19] i amb structs de structs

- Utilitzem `t4_rectangles.c` com a guió argumental per veure com treballem amb structs de structs i com passem per referència el camp d'un struct que és camp d'un struct.
- `printf("(%f,%f)\n", r.p2.x, r.p2.y);`
- `scanf("%f,%f", &p_r->p2.x, &p_r->p2.y);`
- Veiem perquè es «`&p_r->p2.x`» i no «`&p_r->p2->x`».

[H20] per referencia: problemes

- #14: Defina el tipo de dato tpersona para almacenar la siguiente información:
 - a) NIF (número y letra (en un mismo campo))
 - b) Fecha de nacimiento (día, mes y año (por separado))
 - c) Sexo (H-hombre, M-Mujer)
- y escriba el código en C de las siguientes funciones:
 - void leer_persona(tpersona *p) /* Lee del teclado la información de una persona */
 - void mostrar_persona(tpersona p) /* Muestra por pantalla la información de una persona */
 - main() /* programa principal */

[**1] per referencia: problemes (2)

- Problema adicional #7:

- Agafar el codi t4_pa7_b_vacio.c i omplir les funcions que estan buides. La solució està a t4_pa7_b.c.

```
#define MAX_JUG 10
#define MAX_CAR 20
typedef struct {
    • int dia;
    • int mes;
    • int anyo;
} tfecha;
```

```
typedef struct {
    • char nombre[MAX_CAR];
    • int edad;
    • int goles;
    • tfecha fecha;
} tjugador;
```

```
typedef struct {
    • char equipo[MAX_CAR];
    • int num;
    • tjugador lista[MAX_JUG];
} tequipo;
```

[**2] per referencia: problemes (3)

- Problema adicional #4:
 - funció void num_digitos(char num[MAX], int *ent, int *dec)
 - retorna díigits que té la part entera i la part decimal.
 - Main que llegeix de teclat el num i imprimeix per pantalla ent i dec.
- Problema adicional #6:
 - void serie_arit (int ini, int dife, int vec[MAX]);
 - void serie_geom (int ini, int razon, int vec[MAX]);

Introduzca el inicio y la diferencia para la serie aritmetica: 3 5

Serie Aritmetica (ini = 3, dife = 5): 3 8 13 18 23 28 33 43 48

Introduzca el inicio y el factor para la serie geometrica: 1 2

Serie Geometrica (ini = 1, factor = 2): 1 2 4 8 16 32 64 128 256 512