Transparencias adicionales S1

Elementos de C

- Tipos de variables
  - Enteros: char, int, long; modificador unsigned
  - Enumerados: enum (equivale a un entero)
  - Coma flotante: float, double
  - Tipo vacío: void (uso especial)
  - Tipos derivados: struct, arrays, punteros
- Expresiones:
  - Asignaciones: =, +=, -=, *=, /=; incrementos: ++, --
  - Aritméticas: +, -, *, /, %; a nivel de bit: ~, &, |, ^, <<, >>
  - Lógicas: ==, !=, <, >, <=, >=, ||, &&, !
  - El cero se asimila a “falso” y cualquier otro a “verdadero”
  - Operador ternario: a? b: c
- Sentencias:
  - Declaración de variables y tipos (dentro/fuera de función)
  - Expresión, típicamente una asignación var=expr
  - Sentencia compuesta (bloque {...})
  - Condicionales (if, switch), bucles (for, while, do)
  - Otras: sentencia vacía (;), salto (goto)

Ejemplos: tipos de variables y operaciones

int a=5, b=3, c;

c=a/b → c=1

c=a%b → c=2

c=++a → a=a+1=6, c=a=6

c=b++ → c=b=3, b=b+1=4

float d, e=12; /*equivalente e=12.0*/

d=a/b → d=1.0

d=(float) a/b → d=6.0/4=1.5

d=e/a → d=12.0/6=2.0

d+=e → d=d+e=2.0+12.0=14.0

d*=e → d=d*e=14.0*12.0=168.0

c=(a<b)?a:b → c=min(a,b)=min(6,4)=4

c=5&4 → c=101_b and 100_b =100_b=4

c=5|4 → c=101_b or 100_b =101_b=5

c=5^4 → c=101_b xor 100_b =001_b=1

c=~5 → ~5=~101_b = 010_b=2

c=5 → c=101_b

c<<=2 → c=10100_b=20

c>>=1 → c=1010_b=10

```
#include <stdio.h>
main(){
  char x='A';
  printf("x=%c - x=%d\n", x, x);
  printf("y=%c - y=%d\n", x+2, x+2);
}
```

→ x=A - x=65

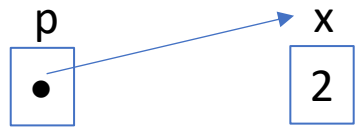
→ y=C- y=67

Arrays y Punteros

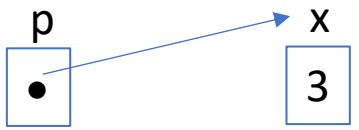
- *Array*: colección de variables del mismo tipo
  - En la declaración se indica la longitud
  - Los elementos se acceden con un índice (empieza en 0)
  - Arrays multidimensionales: double matriz[N][M];
  - Las cadenas son arrays de char acabadas con el carácter ’\0’
- *Puntero*: variable que contiene la dirección de otra variable
  - En la declaración se añade * antes del nombre de variable
  - El operador & devuelve la dirección de una variable
  - El operador * permite acceder al dato apuntado
- Puntero nulo
  - Su valor es cero (NULL)
  - Se usa para indicar un fallo
- Aritmética de punteros
  - Operaciones básicas: +, -, ++
  - El desplazamiento es del tipo al que apunta la variable

```
#define N 10
int i;
double a[N],s=0.0;
for (i=0;i<N;i++)
s = s + a[i];
```

```
double *p, x=2; //p apunta a *p
p = &x;//p apunta a x
```




```
*p=3;
```

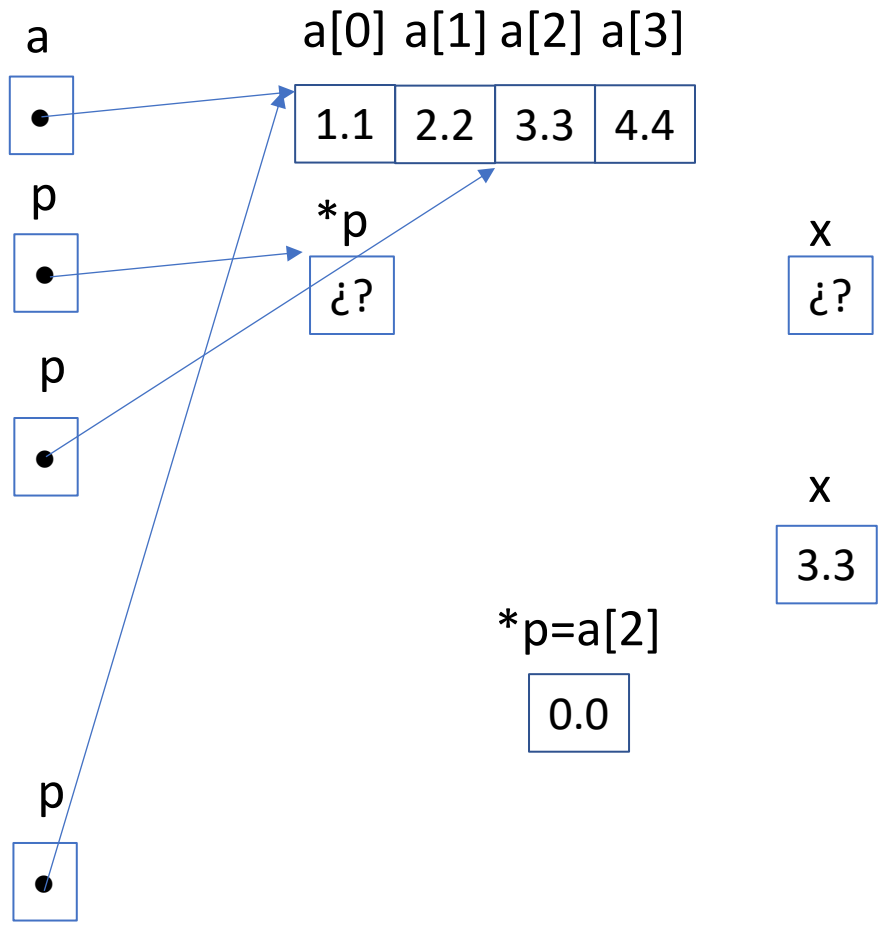



```
double w,*p;
...
if (!p) error("Puntero nulo");
else w = *p;
```

Ejemplos de vectores estáticos y punteros

```
double a[4]={1.1,2.2,3.3,4.4};

double *p, x;

p = &a[2];

x = *p;

*p = 0.0;

p = a;
```

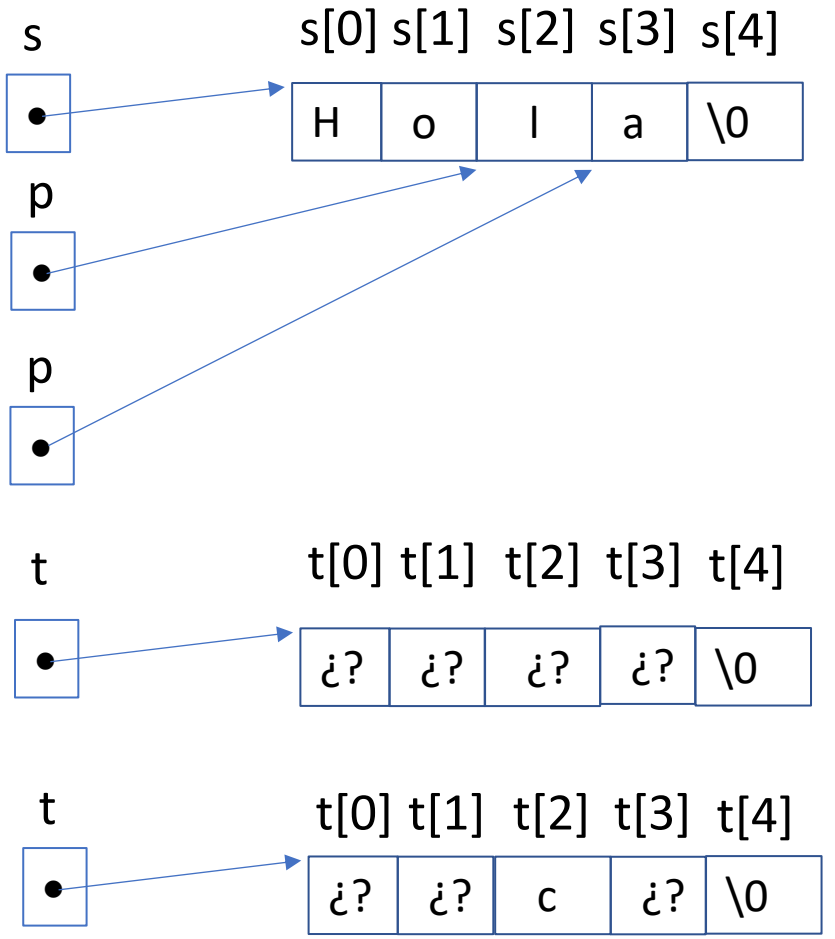
a=p → Error!!!! (a es un puntero constante)



Nota: Vectores estáticos=punteros constantes

Ejemplos cadenas de caracteres

```
char s[]="Hola"

char *p=s+2;

++p;

char t[4];

t[2]= 'c';
```

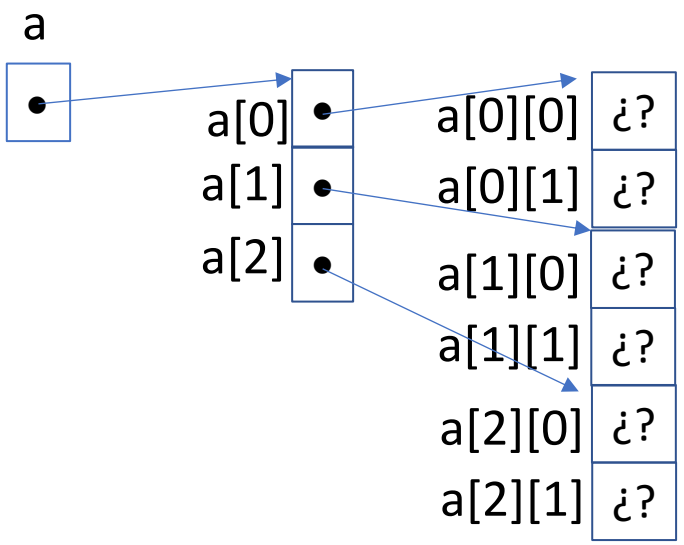


Nota:

char s[]="Hola"≡ char s[]={'H', 'o', 'l', 'a', '\0'} ≠ char s[]={'H', 'o', 'l', 'a' }

Matrices estáticas

double a[3][2];

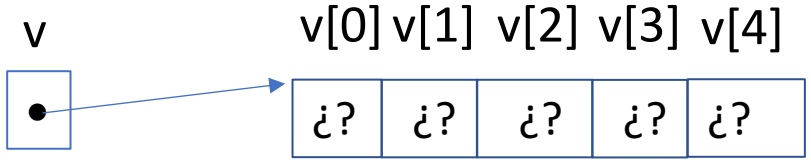


Los elementos del array ocupan posiciones consecutivas de memoria

Nota: Matrices estáticas=punteros dobles constantes

Vectores dinámicos: se crean mediante punteros

```
double *v;
int n;
scanf("%d",&n);/*por ejemplo,en tiempo real tecleamos 5*/
v=(double *) malloc(n*sizeof(double);
….

free(v); /*libera memoria del array v*/
```

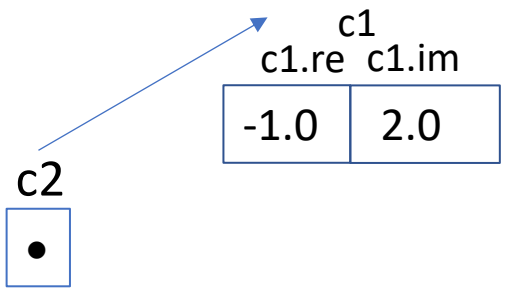


Estructuras en C

- colección de datos heterogéneos
- Los miembros de una estructura se acceden con .
- En el caso de punteros se accede con ->

```
struct complejo {
   double re,im;
};
struct complejo c1, *c2;
c1.re = 1.0;
c1.im = 2.0;
```


```
c2 = &c1;
c2->re = -1.0;
```




```
typedef struct {
   int i,j,k;
   const char *label;
   double data[100];
} mystruct;
mystruct s;
s.label = "NEW";
```

Ejemplo de un programa en C con funciones

funciones_basico.c

```c
#include <stdio.h>
float cuadrado(float x){
  return x*x;
}
void area_perimetro(float b, float h, float *area, float *perimetro){
  *area=b*h;
  *perimetro=2*(b+h);
}
int main(){
  float x, y, base, altura, Area, Perimetro;
  printf("x? ");
  scanf("%f",&x);
  y=cuadrado(x);
  printf("x*x=%f\n",y);
  printf("introduce la base y la altura: ");
  scanf("%f%f",&base,&altura);
  area_perimetro(base,altura,&Area,&Perimetro);
  printf("Area=%f \t Perímetro=%f\n",Area,Perimetro);
}
```

Compilación y ejecución

```
jjibanez@MSI:~/cpa/CPA_Actual/OpenMP$ gcc -o pr funciones_basico.c
jjibanez@MSI:~/cpa/CPA_Actual/OpenMP$ ./pr
x? 3
x*x=9.000000
introduce la base y la altura: 4 6
Area=24.000000   Permetro=20.000000
```

Funciones de Biblioteca

- Operaciones de cadenas <string.h>
  - Copia cadena (strcpy), compara cadena (strcmp)
  - Copia memoria (memcpy), inicializa memoria (memset)
- Entrada-salida <stdio.h>
  - Estándar: printf, scanf
  - Ficheros: fopen, fclose, fprintf, fscanf
- Utilidades estándar <stdlib.h>
  - Gestión de memoria dinámica: malloc, free
  - Conversiones: atof, atoi
- Funciones matemáticas <math.h>
  - Funciones y operaciones: sin, cos, exp, log, pow, sqrt
  - Redondeo: floor, ceil, fabs

Ejemplo de ficheros

Código

```
#include <stdio.h>
int main()
{
  int n1, n2, suma;
  FILE *fe, *fs;
  fe = fopen("entrada.txt", "r");
  fs = fopen("salida.txt", "w");
  fscanf(fe, "%d%d", &n1, &n2);
  suma = n1 + n2;
  fprintf(fs, "La suma es %d\n", suma);
  fprintf(fs,"%d + %d = %d\n", n1, n2, suma);
  fprintf(fs, "Hola");
  fprintf(fs, ", ¿Qué tal?\n");
  fclose(fe);
 fclose(fs);
 return 0;
}
```

Fichero de entrada

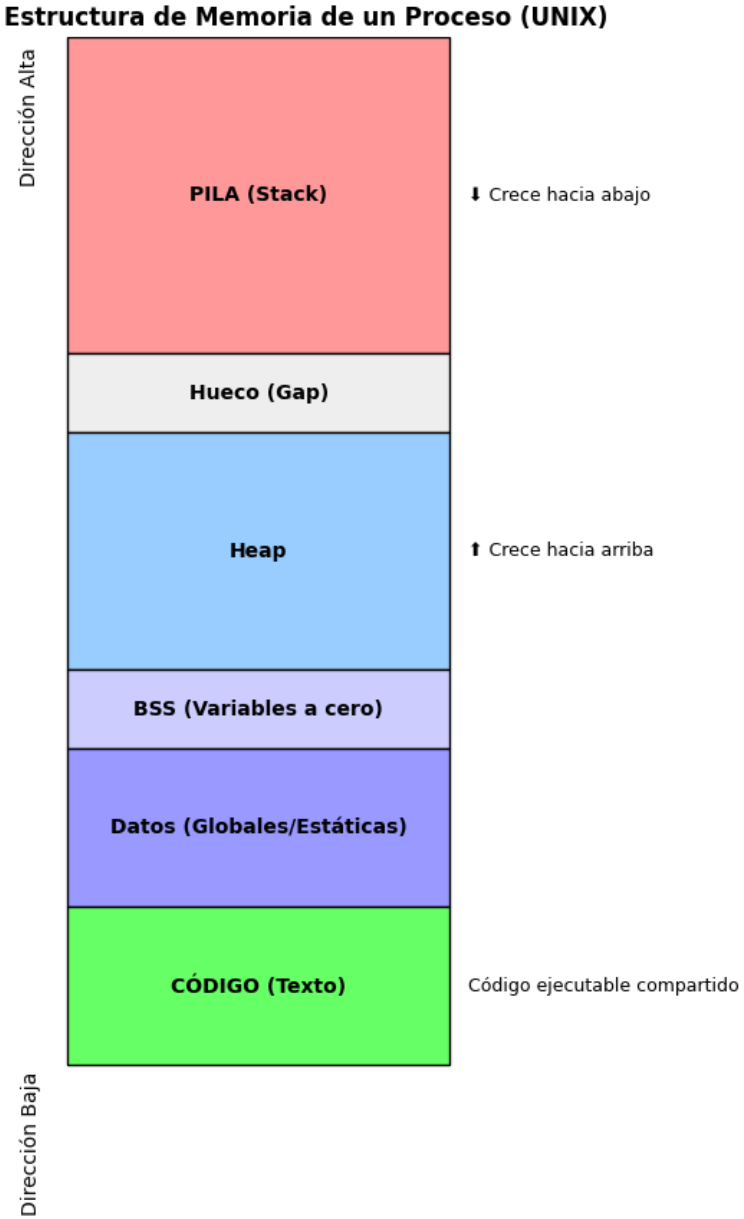
```
4 5
```

Fichero de salida

```
La suma es 9
4 + 5 = 9
Hola, ¿Qué tal?
```

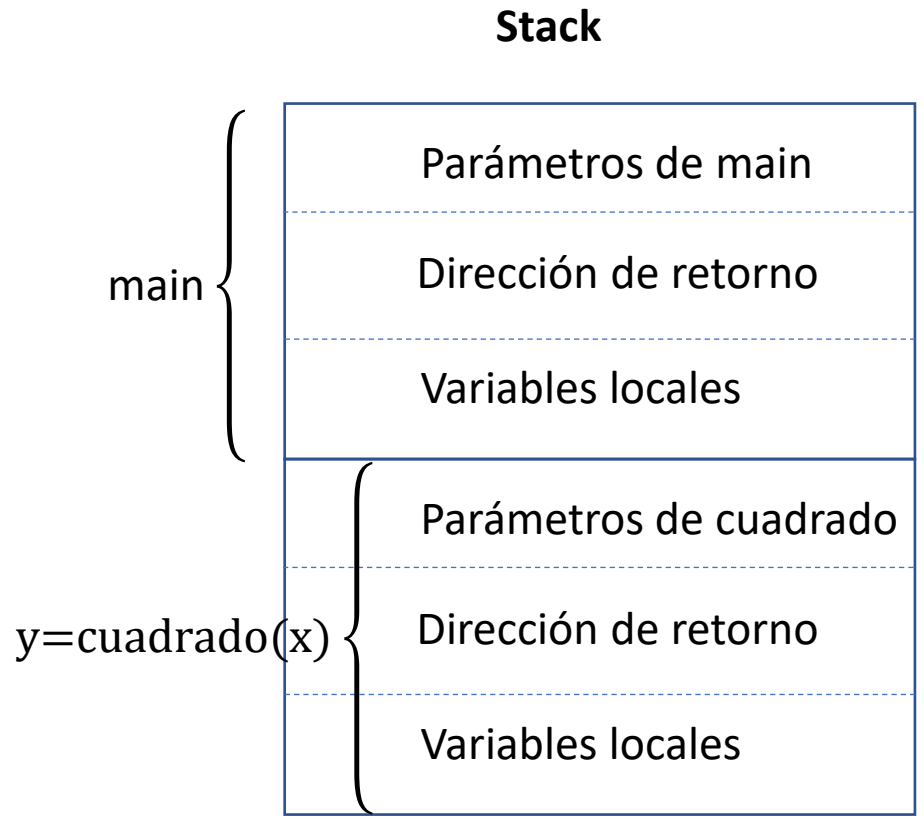
Tipos de Variables

- Variables globales
  - Se declaran fuera de cualquier función
  - Acceso desde cualquier punto del programa
  - Se crean en el segmento de datos
- Variables locales
  - Declaradas dentro de una función
  - Visibles dentro del bloque
  - Se crean en la pila (*stack*), se destruyen al salir
- Variables estáticas
  - Modificador static
  - Ámbito local pero persistentes de una llamada a otra
- Variables en memoria dinámica
  - Memoria reservada con malloc, persisten hasta el free
  - Se crean en el *heap*



Stack (pila)

```c
#include <stdio.h>
float cuadrado(float x){
  return x*x;
}
void area_perimetro(float b, float h, float *area, float *perimetro){
  *area=b*h;
  *perimetro=2*(b+h);
}
int main(){
  float x, y, base, altura, Area, Perimetro;
  printf("x? ");
  scanf("%f",&x);
  y=cuadrado(x);
  printf("x*x=%f\n",y);
  printf("introduce la base y la altura: ");
  scanf("%f%f",&base,&altura);
  area_perimetro(base,altura,&Area,&Perimetro);
  printf("Area=%f \t Perímetro=%f\n",Area,Perimetro);
}
```



Notas:

- Cuando se ejecuta la línea y=cuadrado(x), el contenido del stack es el indicado arriba
- Cuando se ejecuta la línea siguiente, desaparecen los datos de esa función en el **stack**
- Cuando se ejecuta la línea y=area_perimetro(base,altura,&Area, &Perimetro), los datos de la función se almacenan en el stack a continuación de los datos de la función main()

Ciclo de Desarrollo

- El proceso de compilación consta de:
  - Preprocesado: modifica el código fuente en C según una serie de instrucciones (directivas de preprocesado)
  - Compilación (**cc**): genera el código objeto (binario) a partir del código ya preprocesado
  - Enlazado (**ld**): une los códigos objeto de los distintos módulos y bibliotecas externas para generar el ejecutable final
- El ciclo de desarrollo se complementa con otros pasos:
  - Automatizar compilación de programas complejos (make)
  - Depuración de errores (gdb, valgrind)
  - Análisis de prestaciones (gprof)

- Instrucciones más usuales en el preprocesado:
  - include: inserta el contenido de otro fichero
  - define: define constantes y macros (con argumentos)
  - if, ifdef: compilación condicional
  - pragma: directiva del compilador

```
#include "myheader.h“
# include <stdio.h>
#define PI 3.141592
#define DEBUG_
#define AVG(a,b) ((a)+(b))/2
...../*código posterior*/
#ifdef DEBUG_
    printf("variable i=%d\n",i);
#endif
```

Compilación de Programas Paralelos

- OpenMP (memoria compartida): se basa en directivas #pragma omp
  - Los compiladores de C más recientes lo tienen incorporado
  - Es necesario usar –fopenmp en la compilación
  - Ejemplo: gcc -fopenmp -o hola hola.c
- MPI (memoria distribuida)
  - Usaremos la versión gratuita OpenMPI
  - MPI proporciona el comando mpicc, el cual facilita la compilación en diferentes máquinas, añadiendo todas las opciones necesarias (biblioteca de funciones MPI, ruta de mpi.h)
  - mpicc -show muestra las opciones que se usarán
  - Ejemplo: mpicc -o hola_mpi hola_mpi.c