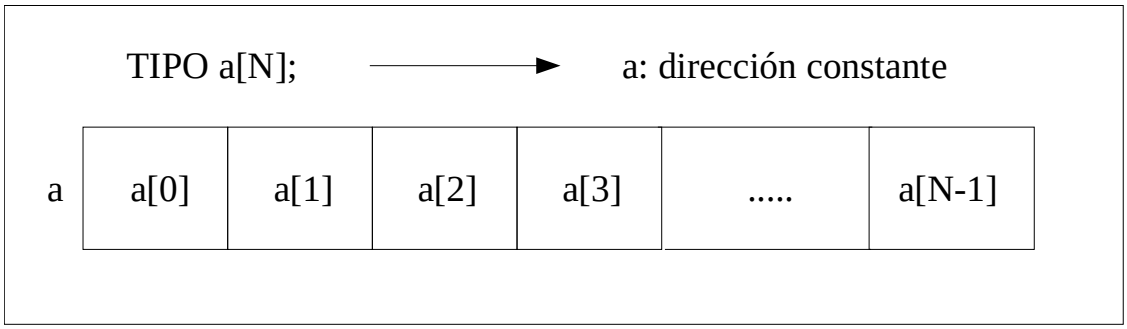
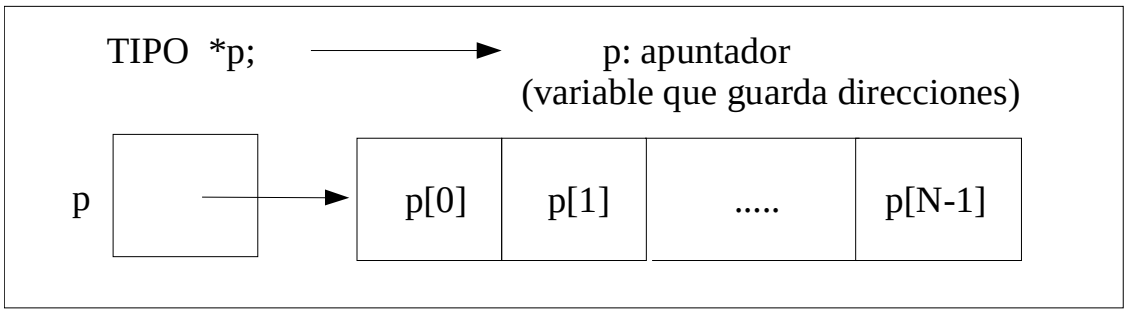
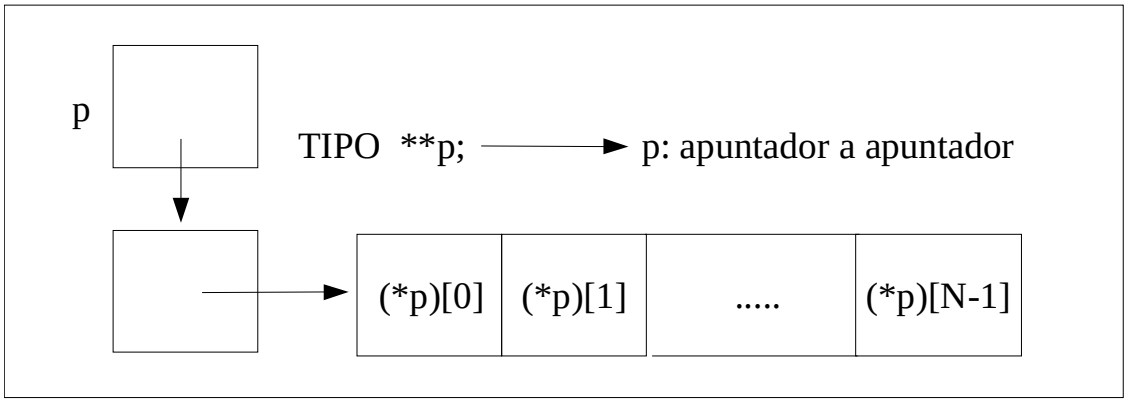







```
int strlen(char s[])
{
     int i;

     for (i=0; s[i]!='\0'; i++)
       /* avanzamos hasta el final de la cadena */ ;

     return i;
}
```

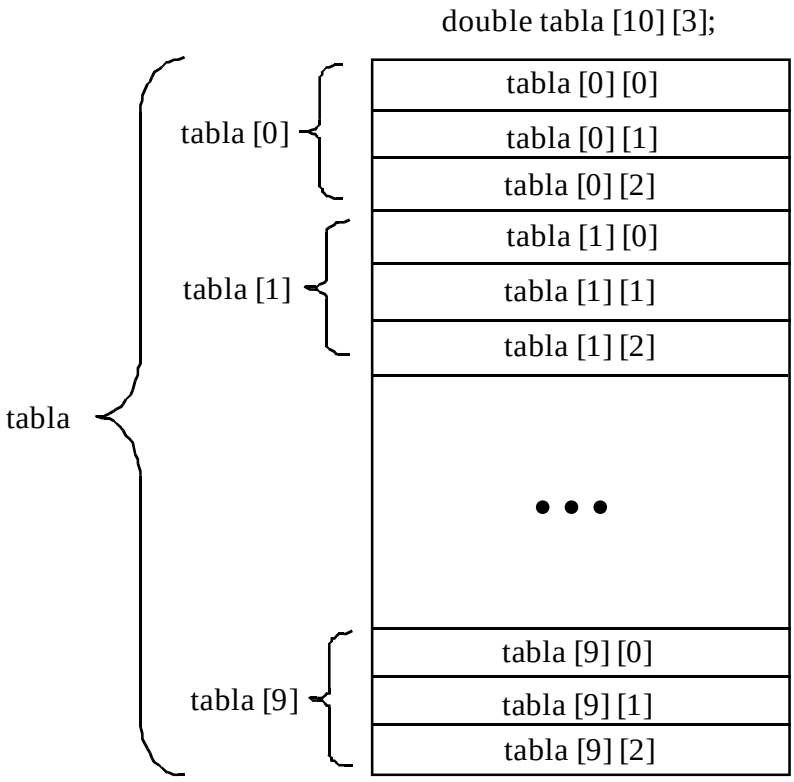
```
int strlen(char *s)
{
     int i;

     for (i=0; *s++; i++)
       /* avanzamos hasta el final de la cadena */ ;

     return i;
}
```

```
void strcpy(char destino[], char origen[])
{
     int i;

     for (i=0; origen[i]!='\0'; i++)
          destino[i]=origen[i];

     destino[i]='\0';
}
```

```
void strcpy(char *destino, char *origen)
{
     /* todo en la misma instrucción:
          asignación
          comparación
          post-incremento          */

     while (*destino++ = *origen++)
               ;
}
```

```
void strcmp(char s1[], char s2[])
{
     int i;

     for (i=0; s1[i] == s2[i]; i++)
          if (s1[i] == '\0')
               return 0;

     return s1[i] – s2[i];
}
```

```
void strcmp(char *s1, char *s2)
{
     for (; *s1 == *s2; s1++, s2++)
          if (*s1 == '\0')
               return 0;

     return *s1 – *s2;
}
```



Operador []

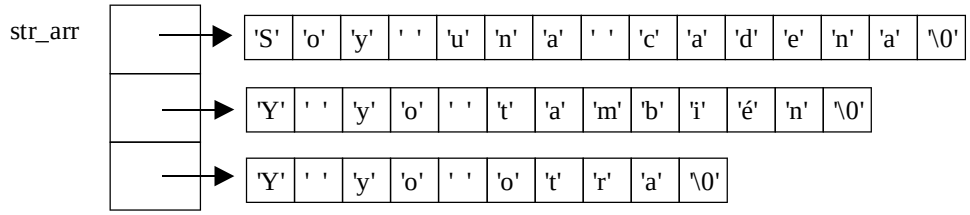
```
tabla[5][1]  →     *(tabla[5]+1)  →
*(*(tabla+5)+1)
```

Declaraciones

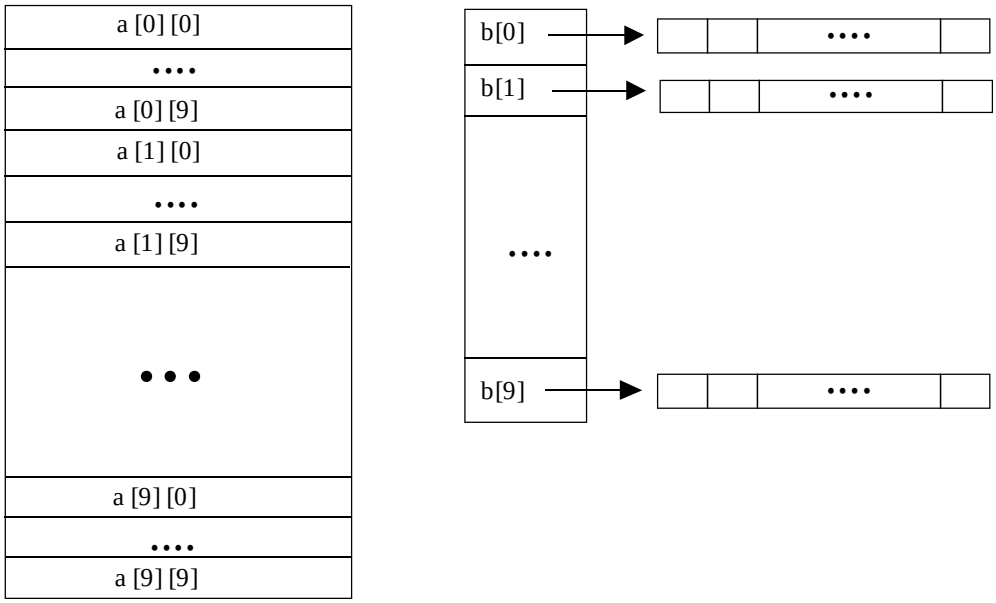
```
double t[10][3];
double t[][3];          →   equivalentes
double (*t)[3];

double *t[3];      →  CUIDADO! (array de apuntadores)
```

Arrays de apuntadores

```
char *str_arr[3];

str_arr[0] = "Soy una cadena";
str_arr[1] = "Y yo también";
str_arr[0] = "Y yo otra";
```




```
int a[10][10];          /* sizeof a  →  400 */

int *b[10];             /* sizeof b  →   40 */

for (i=0; i<10; i++)
     b[i] = (int *) malloc(10*sizeof(int));
```



Inicialización de arrays

ANSI C: *es posible inicializar variables automáticas de tipo array y estructura*

```
int a[10] = {43, -2, 0, 0, 34, 79, -22, 4, -4, 33};
```

```
char nombre[] = "Guillermo Puertas";
        /* nombre: constante  →  se reservan 17 bytes */

char *nombre = "Guillermo Puertas";
        /* nombre: variable  →  se reservan 4 + 17 = 21 bytes
*/

char *mes[] = {"enero", "febrero", "marzo", "abril",
                "mayo", "junio", "julio", "agosto",
            "septiembre", "octubre", "noviembre", "diciembre"};
```

```
int tab[3][4] = {{1, 2, 3, 4},
                 {5, 6, 7, 8},
                 {9, 10, 11, 12}};

int tab[3][4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```

```
int tab[3][4] = {1, 2, 3, 4, 5, 6};

int tab[3][4] = {{1, 2, 3, 4}, {5, 6}};
```

```
int tab[3][4] = {{1},{2},{3, 4}};

      /*    tab:  variable automática     */

int tab[3][4] = {{1, ?, ?, ?},
                 {2, ?, ?, ?},
                 {3, 4, ?, ?}};

      /*    tab:  variable estática       */

int tab[3][4] = {{1, 0, 0, 0},
                 {2, 0, 0, 0},
                 {3, 4, 0, 0}};
```

Apuntadores y funciones

Funciones que retornan apuntadores

```
char *strchr(char *s, char c)
{
      while (*s && *s != c)
            s++;
      return *s? s: NULL;
}
```

Apuntadores a funciones

```
double integra(double (*f)(double), double x1, double x2, int n)
{
 double inc, resultado, x;

 inc = (x2 - x1) / n;
 resultado = 0.0;

 for (x = x1; x < x2; x += inc)
       resultado += inc * (*f)(x);
 return resultado;
}
```

Declaraciones complicadas

`int (*v)[10];`
apuntador a un array de 10 enteros

`int *v[10];`
array de 10 apuntadores a entero

`char *f(char *, char);`
función que retorna un apuntador a char y que tiene como argumentos un apuntador a char y un char

`char (*f)(char *, char)`
apuntador a función que retorna char y que tiene como argumentos un apuntador a char y un char

`char (*(*x(char *))[])(char *);`
función que tiene como argumento un apuntador a char y que retorna un apuntador a un array de apuntadores a función que retorna char y tiene como argumento un apuntador a char

`char (*(*x[3])(char *))[5];`
array de 3 apuntadores a función que tiene como argumento un apuntador a char y que retorna un apuntador a array de 5 char

Argumentos en línea de comando

Objetivo

Pasar argumentos en el momento de llamar al ejecutable

Qué se pasa

- datos
- nombres de ficheros que han de procesarse
- opciones que modifiquen el comportamiento del programa

Método

la función *main()* admite dos argumentos, *argc* y *argv*, que permiten procesar las cadenas de caracteres que pasemos en línea de comando

argc

entero
nº de cadenas que se pasan en línea de comando

argv

array de apuntadores a char
lista de cadenas

```
#include <stdio.h>

void main(int argc, char *argv[])
{
      int i;
      double x, media;

      media=0.0;
      for (i=1; i<argc; i++)
      {
            sscanf(argv[i],"%lf",&x);
            media+=x;
      }
      media/=(argc-1);
      printf("La media es %f\n",media);
}
```

```
luisja@sol[4]: cc media.c -o media
luisja@sol[5]: media 2.38 3.54 4.06 5.68 9.27
La media es 4.986
luisja@sol[6]:
```