



# INFORMATICA I

Arreglos o " **vectores** "

Ing. Juan Carlos Cuttitta

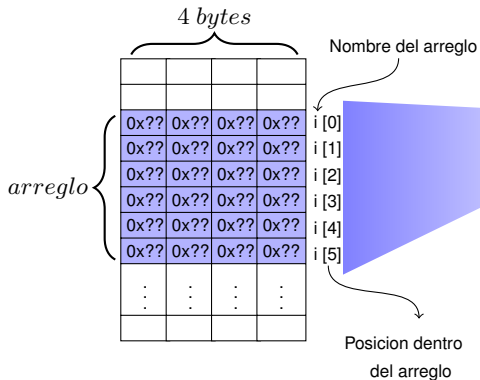
*Universidad Tecnológica Nacional  
Facultad Regional Buenos Aires  
Departamento de Ingeniería Electrónica*

4 de junio de 2020

# Declaración y disposición en memoria

## Arquitectura X86-32 bits

Disposición de la variable *i*  
en memoria



## Código en programa fuente

```
1  int main ()
2  {
3      int i [6]; ← Declara arreglo
4      .....
5      i [1]=10;
6      .....
7  }
```

Declara arreglo de 6 enteros

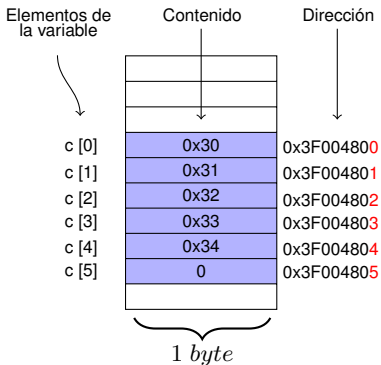


# Dirección no es lo mismo que orden de elemento

Declaración en código

`unsigned char c[6]`

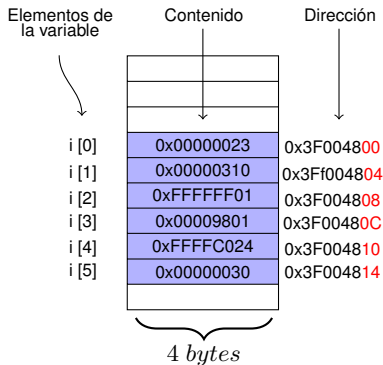
Cardena de caracteres



Declaración en código

`int c[6]`

Arreglo de enteros



# Dirección no es lo mismo que orden de elemento

Declaración en código

`unsigned char c[6]`

Cardena de caracteres

Elementos de la variable

Contenido

Dirección

c [0]	0x30	0x3F004800
c [1]	0x31	0x3F004801
c [2]	0x32	0x3F004802
c [3]	0x33	0x3F004803
c [4]	0x34	0x3F004804
c [5]	0	0x3F004805

1 byte

*notar la diferencia  
uno termina con '\0' y el otro no*

Declaración en código

`int c[6]`

Arreglo de enteros

Elementos de la variable

Contenido

Dirección

i [0]	0x00000023	0x3F004800
i [1]	0x00000310	0x3F004804
i [2]	0xFFFFFFFF01	0x3F004808
i [3]	0x00009801	0x3F00480C
i [4]	0xFFFFC024	0x3F004810
i [5]	0x00000030	0x3F004814

4 bytes

# Dirección no es lo mismo que orden de elemento

Declaración en código

`unsigned char c[6]`

Cardena de caracteres

Elementos de la variable

Contenido

Dirección

c [0]	0x30='0'	0x3F004800
c [1]	0x31='1'	0x3F004801
c [2]	0x32='2'	0x3F004802
c [3]	0x33='3'	0x3F004803
c [4]	0x34='4'	0x3F004804
c [5]	0='\0'	0x3F004805

1 byte

*notar la diferencia  
uno termina con '\0' y el otro no*

Declaración en código

`int c[6]`

Arreglo de enteros

Elementos de la variable

Contenido

Dirección

i [0]	0x00000023	0x3F004800
i [1]	0x00000310	0x3F004804
i [2]	0xFFFFFFFF01	0x3F004808
i [3]	0x00009801	0x3F00480C
i [4]	0xFFFFC024	0x3F004810
i [5]	0x00000030	0x3F004814

4 bytes

# Un caso especial de arreglo

## Arreglo de caracteres

Se trata de un tipo muy usual de dato que llamamos cadena o string (del inglés).

Se inicializa de los siguientes modos:

```
1 /*Una forma de inicializar un arreglo de caracteres con una
   cadena*/
2
3 char cad []= "Hola mundo!";
4
5 /* Otra forma... */
6
7 char cad []={'H','o','l','a',' ','m','u','n','d','o','!','\0'};
```

# Un caso especial de arreglo

## Arreglo de caracteres

Se trata de un tipo muy usual de dato que llamamos cadena o string (del inglés).

Se inicializa de los siguientes modos:

```
1 /*Una forma de inicializar un arreglo de caracteres con una
   cadena*/
2
3 char cad []= "Hola mundo!";
4
5 /* Otra forma...*/
6
7 char cad []={ 'H', 'o', 'l', 'a', ' ', 'm', 'u', 'n', 'd', 'o', '!', '\0' };
```



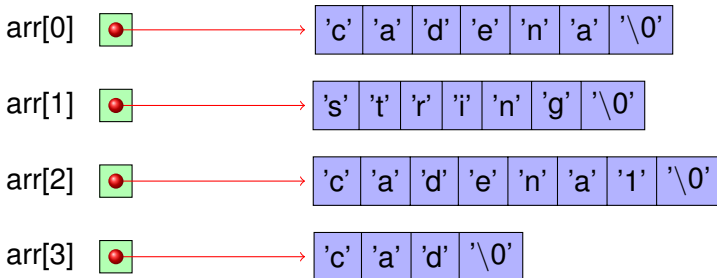
## En código....

```
1  /* Utiliza una lista de inicialización para
   2     inicializar el arreglo arr.*/
3  int arr[8] = {23,0,-669,-1,995,1277,90,-9000};
4  /* La cantidad de elementos en la línea anterior
   5     es redundante. Puede hacerse lo mismo de la
   6     siguiente forma*/
7  int arr[] = {23,0,-669,-1,995,1277,90,-9000};
8  /* Si lo vamos a inicializar con el mismo valor
   9     para todos los elementos, la forma adecuada
  10     es la siguiente*/
11
12 int arr[8], i;
13 for (i = 0 ; i < 8 ; i++)
14 {
15     arr[i] = 0;
16 }
```

# Arreglos de punteros

```
1 char *arr[4] = { "cadena", "string", "cadena1", "cad" };
```

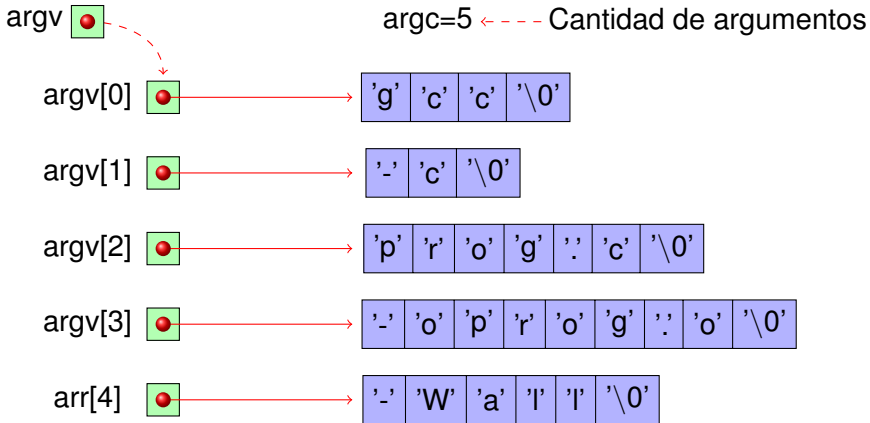
Genera esta disposición en memoria



# main, la función multifacética

Ejemplo con función conocida **gcc -c prog.c -oprogram.o -Wall**

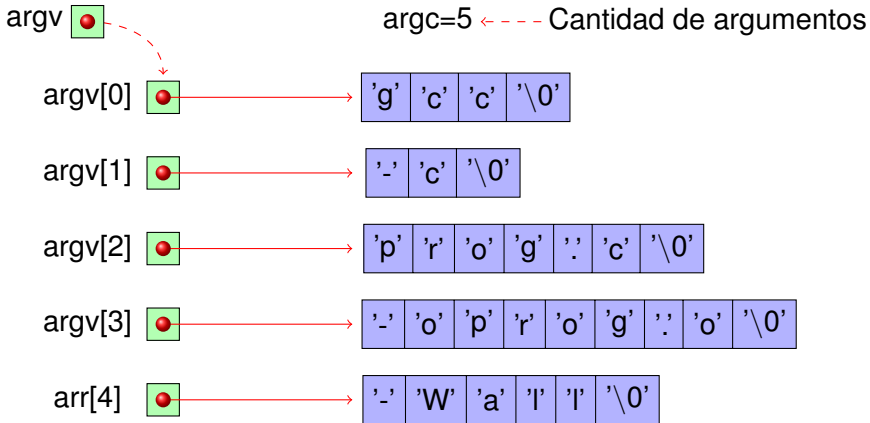
```
int main (int argc, char *argv[])
```



# main, la función multifacética

Ejemplo con función conocida **gcc -c prog.c -oprogram.o -Wall**

```
int main (int argc, char **argv)
```

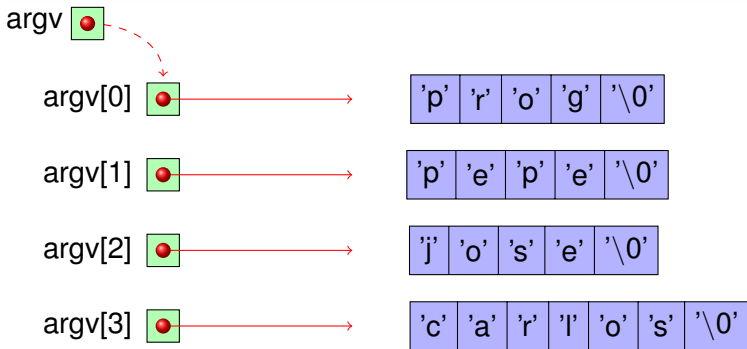


# Ejemplo de *array de direcciones*

```
./prog pepe jose carlos
```

Los argumentos del programa son:

- pepe
- jose
- carlos



## Arquitectura X86-32 bits

argv 0xFFE07650

0xFFE07650

argv[0]	0xFFF47600
argv[1]	
argv[2]	
argv[3]	

'p' 'r' 'o' 'g' '\0'

'p' 'e' 'p' 'e' '\0'

'j' 'o' 's' 'e' '\0'

'c' 'a' 'r' 'l' 'o' 's' '\0'

## Arquitectura X86-32 bits

argv 0xFFE07650

0xFFE07650

argv[0]	0xFFFF47600
argv[1]	
argv[2]	
argv[3]	

0xFFFF47600

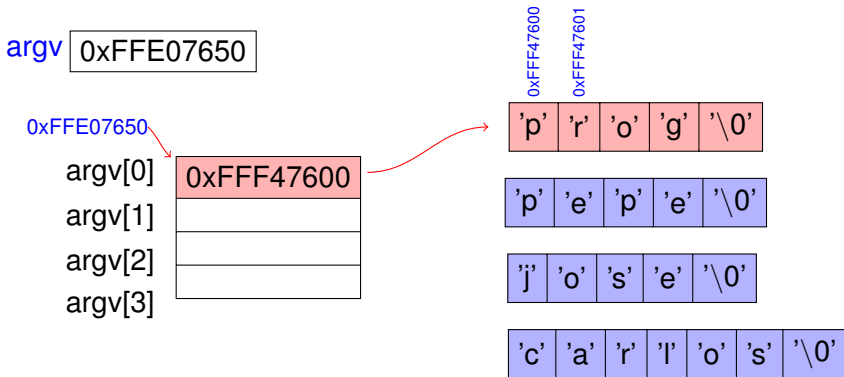
'p' 'r' 'o' 'g' '\0'

'p' 'e' 'p' 'e' '\0'

'j' 'o' 's' 'e' '\0'

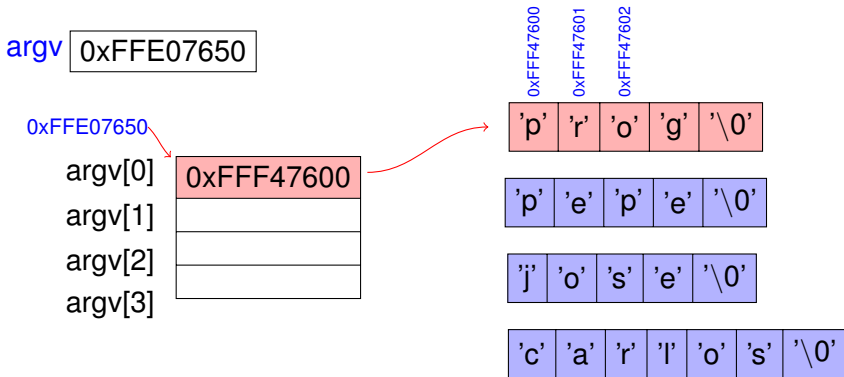
'c' 'a' 'r' 'l' 'o' 's' '\0'

## Arquitectura X86-32 bits

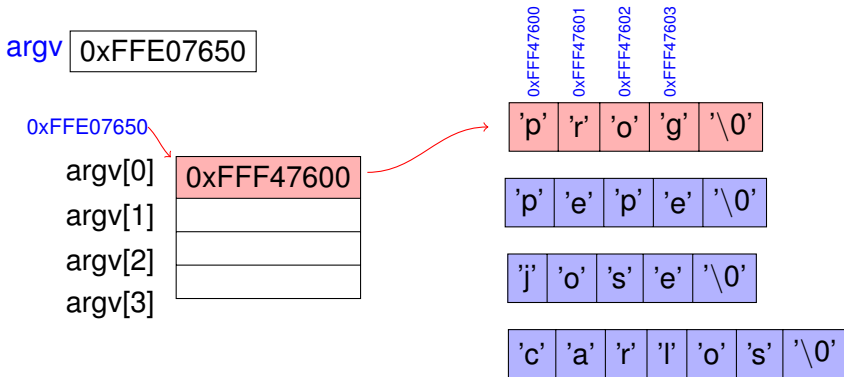




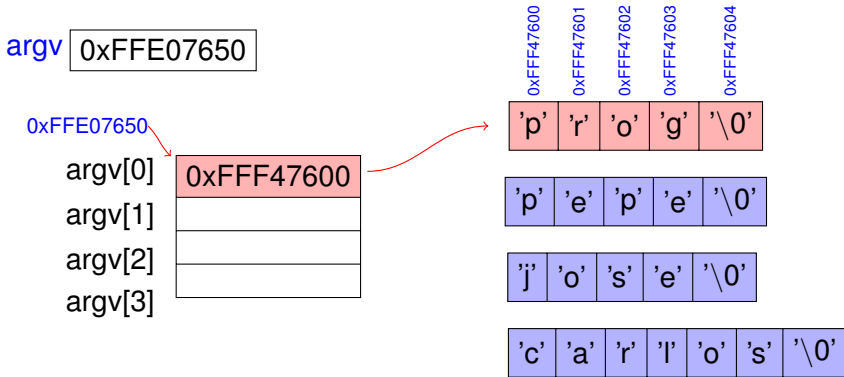
## Arquitectura X86-32 bits



## Arquitectura X86-32 bits



## Arquitectura X86-32 bits



## Arquitectura X86-32 bits

argv 0xFFE07650

0xFFE07650

argv[0]	0xFFF47600
argv[1]	0xFFF47605
argv[2]	
argv[3]	

'p' 'r' 'o' 'g' '\0'

'p' 'e' 'p' 'e' '\0'

'j' 'o' 's' 'e' '\0'

'c' 'a' 'r' 'l' 'o' 's' '\0'

## Arquitectura X86-32 bits

argv 0xFFE07650

0xFFE07650

argv[0]	0xFFF47600
argv[1]	0xFFF47605
argv[2]	0xFFF4760A
argv[3]	

'p' 'r' 'o' 'g' '\0'

'p' 'e' 'p' 'e' '\0'

'j' 'o' 's' 'e' '\0'

'c' 'a' 'r' 'l' 'o' 's' '\0'

## Arquitectura X86-32 bits

argv 0xFFE07650

0xFFE07650

argv[0]	0xFFF47600
argv[1]	0xFFF47605
argv[2]	0xFFF4760A
argv[3]	0xFFF4760F

'p' 'r' 'o' 'g' '\0'

'p' 'e' 'p' 'e' '\0'

'j' 'o' 's' 'e' '\0'

'c' 'a' 'r' 'l' 'o' 's' '\0'

# Argumentos por línea de comandos

4 bytes

0xFFE07600	0xFFFF47650	argv[0]
0xFFE07654	0xFFFF47657	argv[1]
0xFFE07658	0xFFFF4765B	argv[2]
	3	argc

*ejecutamos el programa*

*./prog pio hoy*

1 bytes

0xFFFF47650	':'
0xFFFF47651	'/'
0xFFFF47652	'p'
0xFFFF47653	'r'
0xFFFF47654	'o'
0xFFFF47655	'g'
0xFFFF47656	'\0'
0xFFFF47657	'p'
0xFFFF47658	'i'
0xFFFF47659	'o'
0xFFFF4765A	'\0'
0xFFFF4765B	'h'
0xFFFF4765C	'o'
0xFFFF4765D	'y'
0xFFFF4765E	'\0'

```
1 #include <stdio.h>
2
3
4 int main (int argc , char ** argv)
5 {
6     int    i;
7
8     printf("primer ejemplo\n");
9     printf("%d es argc\n",argc);
10    for ( i=0 ; i<argc ; i++ )
11    {
12        printf(" %s \t" , *(argv+i) );
13    }
14 }
```

# Argumentos por línea de comandos

4 bytes

0xFFE07600	0xFFFF47650	argv[0]
0xFFE07654	0xFFFF47657	argv[1]
0xFFE07658	0xFFFF4765B	argv[2]
	3	argc

*ejecutamos el programa*

*./prog pio hoy*

*argc = 3*

1 bytes

0xFFFF47650	'.'
0xFFFF47651	'/'
0xFFFF47652	'p'
0xFFFF47653	'r'
0xFFFF47654	'o'
0xFFFF47655	'g'
0xFFFF47656	'\0'
0xFFFF47657	'p'
0xFFFF47658	'i'
0xFFFF47659	'o'
0xFFFF4765A	'\0'
0xFFFF4765B	'h'
0xFFFF4765C	'o'
0xFFFF4765D	'y'
0xFFFF4765E	'\0'

```
1 #include <stdio.h>
2
3
4 int main (int argc , char ** argv)
5 {
6     int    i;
7
8     printf("primer ejemplo\n");
9     printf("%d es argc\n",argc);
10    for ( i=0 ; i<argc ; i++ )
11    {
12        printf(" %s \t" , *(argv+i) );
13    }
14 }
```



# Argumentos por línea de comandos

4 bytes

*ejecutamos el programa*

0xFFE07600  
0xFFE07654  
0xFFE07658

0xFFFF47650
0xFFFF47657
0xFFFF4765B
3

argv[0] ←  
argv[1]  
argv[2]  
argc

*./prog pio hoy*  
argv[0] argv[1] argv[2]

1 bytes

0xFFFF47650  
0xFFFF47651  
0xFFFF47652  
0xFFFF47653  
0xFFFF47654  
0xFFFF47655  
0xFFFF47656  
0xFFFF47657  
0xFFFF47658  
0xFFFF47659  
0xFFFF4765A  
0xFFFF4765B  
0xFFFF4765C  
0xFFFF4765D  
0xFFFF4765E

':'
'/'
'p'
'r'
'o'
'g'
'\0'
'p'
'i'
'o'
'\0'
'h'
'o'
'y'
'\0'

```
1 #include <stdio.h>
2
3
4 int main (int argc , char ** argv)
5 {
6     int    i;
7
8     printf("primer ejemplo\n");
9     printf("%d es argc\n",argc);
10    for ( i=0 ; i<argc ; i++ )
11    {
12        printf(" %s \t" , *(argv+i) );
13    }
14 }
```

# Argumentos por línea de comandos

4 bytes

0xFFE07600	0xFFFF47650	argv[0]
0xFFE07654	0xFFFF47657	argv[1]
0xFFE07658	0xFFFF4765B	argv[2]
	3	argc

*ejecutamos el programa*

*./prog pio hoy*

*argv[0] argv[1] argv[2]*

1 bytes

0xFFFF47650	''
0xFFFF47651	'/'
0xFFFF47652	'p'
0xFFFF47653	'r'
0xFFFF47654	'o'
0xFFFF47655	'g'
0xFFFF47656	'\0'
0xFFFF47657	'p'
0xFFFF47658	'i'
0xFFFF47659	'o'
0xFFFF4765A	'\0'
0xFFFF4765B	'h'
0xFFFF4765C	'o'
0xFFFF4765D	'y'
0xFFFF4765E	'\0'

```
1 #include <stdio.h>
2
3
4 int main (int argc , char ** argv)
5 {
6     int     i;
7
8     printf("primer ejemplo\n");
9     printf("%d es argc\n",argc);
10    for ( i=0 ; i<argc ; i++ )
11    {
12        printf("%s \t" , *(argv+i) );
13    }
14 }
```

# Argumentos por línea de comandos

4 bytes

0xFFE07600  
0xFFE07654  
0xFFE07658

0xFFF47650
0xFFF47657
0xFFF4765B
3

argv[0]  
argv[1]  
argv[2] ←  
argc

*ejecutamos el programa*

*./prog pio hoy*

*argv[0] argv[1] argv[2]*

1 bytes

0xFFF47650  
0xFFF47651  
0xFFF47652  
0xFFF47653  
0xFFF47654  
0xFFF47655  
0xFFF47656  
0xFFF47657  
0xFFF47658  
0xFFF47659  
0xFFF4765A  
0xFFF4765B  
0xFFF4765C  
0xFFF4765D  
0xFFF4765E

''
'/'
'p'
'r'
'o'
'g'
'\0'
'p'
'i'
'o'
'\0'
'h'
'o'
'y'
'\0'

}

```
1 #include <stdio.h>
2
3
4 int main (int argc , char ** argv)
5 {
6     int    i;
7
8     printf("primer ejemplo\n");
9     printf("%d es argc\n",argc);
10    for ( i=0 ; i<argc ; i++ )
11    {
12        printf("%s \t" , *(argv+i) );
13    }
14 }
```

# Accediendo a los caracteres

*ejecutamos el programa*

*./prog pio hoy*

		4 bytes	
0xFFE07600	0xFFFF47650	}	argv[0]
0xFFE07654	0xFFFF47657		argv[1]
0xFFE07658	0xFFFF4765B		argv[2]
	3		argc

		1 bytes	
0xFFFF47650	:	}	
0xFFFF47651	'r'		
0xFFFF47652	'p'		
0xFFFF47653	'r'		
0xFFFF47654	'o'		
0xFFFF47655	'g'		
0xFFFF47656	'\0'		
0xFFFF47657	'p'		
0xFFFF47658	'i'		
0xFFFF47659	'o'		
0xFFFF4765A	'\0'		
0xFFFF4765B	'h'		
0xFFFF4765C	'o'		
0xFFFF4765D	'y'		
0xFFFF4765E	'\0'		

p = 0xFFFF47657

```
1 #include <stdio.h>
2
3 int main (int argc , char ** argv)
4 {
5     int     i , j;
6     char    *p;
7
8     printf("ejemplo de a caracteres\n");
9
10    for ( j=1 ; j<argc ; j++ )
11    {
12        p = *(argv+j);
13
14        for ( i=0;*( p+i )!= '\0' ; i++ )
15        {
16            printf("%c " , *( p+i ));
17        }
18        printf("\t");
19    }
20 }
```

# Accediendo a los caracteres

*ejecutamos el programa*

*./prog pio hoy*

4 bytes

0xFFE07600	0xFFFF47650	argv[0]
0xFFE07654	0xFFFF47657	argv[1]
0xFFE07658	0xFFFF4765B	argv[2]
	3	argc

1 bytes

0xFFFF47650	':'
0xFFFF47651	'/'
0xFFFF47652	'p'
0xFFFF47653	'r'
0xFFFF47654	'o'
0xFFFF47655	'g'
0xFFFF47656	'\0'
0xFFFF47657	'p'
0xFFFF47658	'i'
0xFFFF47659	'o'
0xFFFF4765A	'\0'
0xFFFF4765B	'h'
0xFFFF4765C	'o'
0xFFFF4765D	'y'
0xFFFF4765E	'\0'

p = 0xFFFF47657

*\*(p+0) -> 'p' ←*

```
1 #include <stdio.h>
2
3 int main (int argc , char ** argv)
4 {
5     int     i , j;
6     char    *p;
7
8     printf("ejemplo de a caracteres\n");
9
10    for ( j=1 ; j<argc ; j++ )
11    {
12        p = *(argv+j);
13
14        for (i=0;*( p+i )!='\0' ; i++ )
15        {
16            printf("%c ", *( p+i ));
17        }
18        printf("\t");
19    }
20 }
```

# Accediendo a los caracteres

*ejecutamos el programa*

*./prog pio hoy*

4 bytes

0xFFE07600	0xFFFF47650	argv[0]
0xFFE07654	0xFFFF47657	argv[1]
0xFFE07658	0xFFFF4765B	argv[2]
	3	argc

1 bytes

0xFFFF47650	':'
0xFFFF47651	'/'
0xFFFF47652	'p'
0xFFFF47653	'r'
0xFFFF47654	'o'
0xFFFF47655	'g'
0xFFFF47656	'\0'
0xFFFF47657	'p'
0xFFFF47658	'i'
0xFFFF47659	'o'
0xFFFF4765A	'\0'
0xFFFF4765B	'h'
0xFFFF4765C	'o'
0xFFFF4765D	'y'
0xFFFF4765E	'\0'

p = 0xFFFF47657

\*(p+0) -> 'p'

\*(p+1) -> 'i' ←

```
1 #include <stdio.h>
2
3 int main (int argc , char ** argv)
4 {
5     int     i , j;
6     char    *p;
7
8     printf("ejemplo de a caracteres\n");
9
10    for ( j=1 ; j<argc ; j++ )
11    {
12        p = *(argv+j);
13
14        for ( i=0; *( p+i ) != '\0' ; i++ )
15        {
16            printf("%c ", *( p+i ));
17        }
18        printf("\t");
19    }
20 }
```

# Accediendo a los caracteres

*ejecutamos el programa*

*./prog pio hoy*

4 bytes

0xFFE07600	0xFFFF47650	argv[0]
0xFFE07654	0xFFFF47657	argv[1]
0xFFE07658	0xFFFF4765B	argv[2]
	3	argc

1 bytes

0xFFFF47650	':'
0xFFFF47651	'/'
0xFFFF47652	'p'
0xFFFF47653	'r'
0xFFFF47654	'o'
0xFFFF47655	'g'
0xFFFF47656	'\0'
0xFFFF47657	'p'
0xFFFF47658	'i'
0xFFFF47659	'o'
0xFFFF4765A	'\0'
0xFFFF4765B	'h'
0xFFFF4765C	'o'
0xFFFF4765D	'y'
0xFFFF4765E	'\0'

p = 0xFFFF47657

\*(p+0) -> 'p'

\*(p+1) -> 'i'

\*(p+2) -> 'o' <--

```
1 #include <stdio.h>
2
3 int main (int argc , char ** argv)
4 {
5     int    i , j;
6     char   *p;
7
8     printf("ejemplo de a caracteres\n");
9
10    for ( j=1 ; j<argc ; j++ )
11    {
12        p = *(argv+j);
13
14        for ( i=0;*( p+i )!= '\0 ' ; i++ )
15        {
16            printf("%c ", *( p+i ));
17        }
18        printf("\t");
19    }
20 }
```

# Accediendo a los caracteres

*ejecutamos el programa*

*./prog pio hoy*

4 bytes

0xFFE07600	0xFFFF47650	argv[0]
0xFFE07654	0xFFFF47657	argv[1]
0xFFE07658	0xFFFF4765B	argv[2]
	3	argv

1 bytes

0xFFFF47650  
0xFFFF47651  
0xFFFF47652  
0xFFFF47653  
0xFFFF47654  
0xFFFF47655  
0xFFFF47656  
0xFFFF47657  
0xFFFF47658  
0xFFFF47659  
0xFFFF4765A  
0xFFFF4765B  
0xFFFF4765C  
0xFFFF4765D  
0xFFFF4765E

':'
'p'
'r'
'p'
'r'
'o'
'g'
'\0'
'p'
'i'
'o'
'\0'
'h'
'o'
'y'
'\0'

p = 0xFFFF47657

\*(p+0) -> 'p'

\*(p+1) -> 'i'

\*(p+2) -> 'o'

\*(p+3) -> '\0' <-----

En este caso sale del for y  
finaliza la impresion de los caracteres

```
1 #include <stdio.h>
2
3 int main (int argc , char ** argv)
4 {
5     int     i , j;
6     char   *p;
7
8     printf("ejemplo de a caracteres\n");
9
10    for ( j=1 ; j<argc ; j++ )
11    {
12        p = *(argv+j);
13
14        for ( i=0; *( p+i ) != '\0' ; i++ )
15        {
16            printf(" %c " , *( p+i ) );
17        }
18        printf("\t");
19    }
20 }
```



# Accediendo a los caracteres con una función

4 bytes

0xFFE07600	0xFFFF47650	argv[0]
0xFFE07654	0xFFFF47657	argv[1]
0xFFE07658	0xFFFF4765B	argv[2]
	3	argc

1 bytes

0xFFFF47650	':'
0xFFFF47651	'r'
0xFFFF47652	'p'
0xFFFF47653	'r'
0xFFFF47654	'o'
0xFFFF47655	'g'
0xFFFF47656	'\0'
0xFFFF47657	'p'
0xFFFF47658	'i'
0xFFFF47659	'o'
0xFFFF4765A	'\0'
0xFFFF4765B	'h'
0xFFFF4765C	'o'
0xFFFF4765D	'y'
0xFFFF4765E	'\0'

*ejecutamos el programa*

*./prog pio hoy*

```
1 #include <stdio.h>
2
3 void imprimir_caracteres (char * );
4
5 int main (int argc , char ** argv)
6 {
7     int i;
8
9     for ( i=1 ; i<argc ; i++ )
10    {
11        imprimir_caracteres (*(argv+i));
12    }
13 }
14
15 void imprimir_caracteres (char * p)
16 {
17     int i;
18
19     for ( i=0 ; *(p+i) != '\0' ; i++)
20     {
21         printf(" %c " , *(p+i) );
22     }
23 }
```

*\*(argv+i) -> = 0xFFFF47657 <-*