INFORMATICA I

Arreglos o " **vectores** "

Ing.Juan Carlos Cuttitta

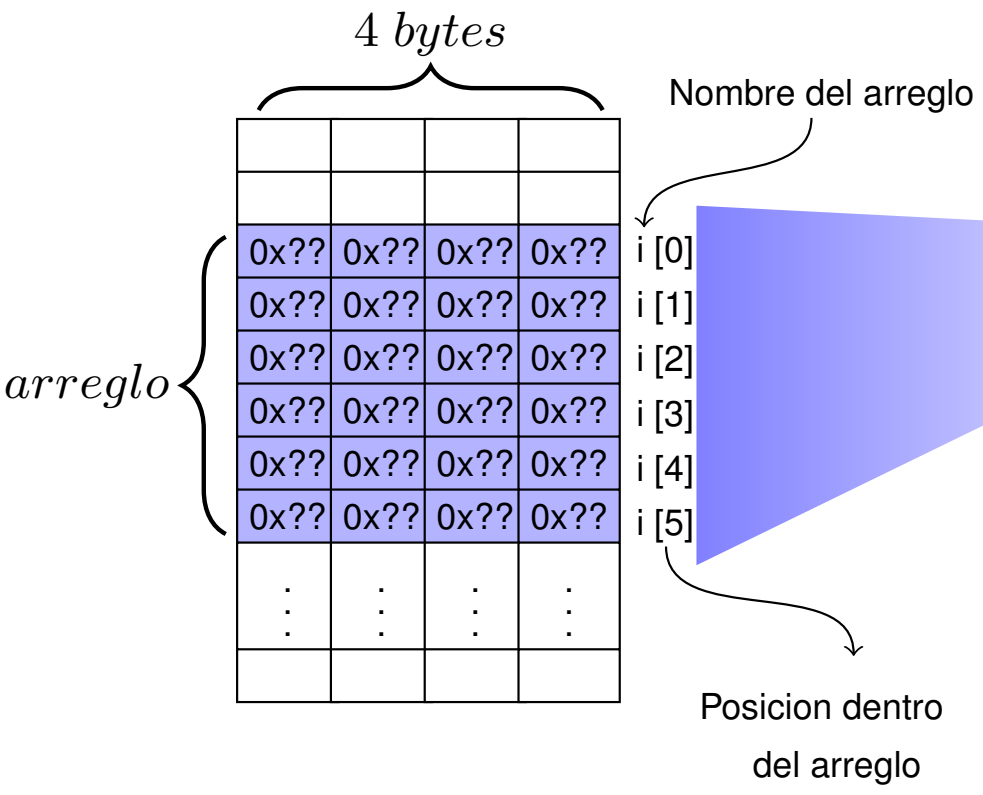
Universidad Tecnológica Nacional
Facultad Regional Buenos Aires
Departamento de Ingeniería Electrónica

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

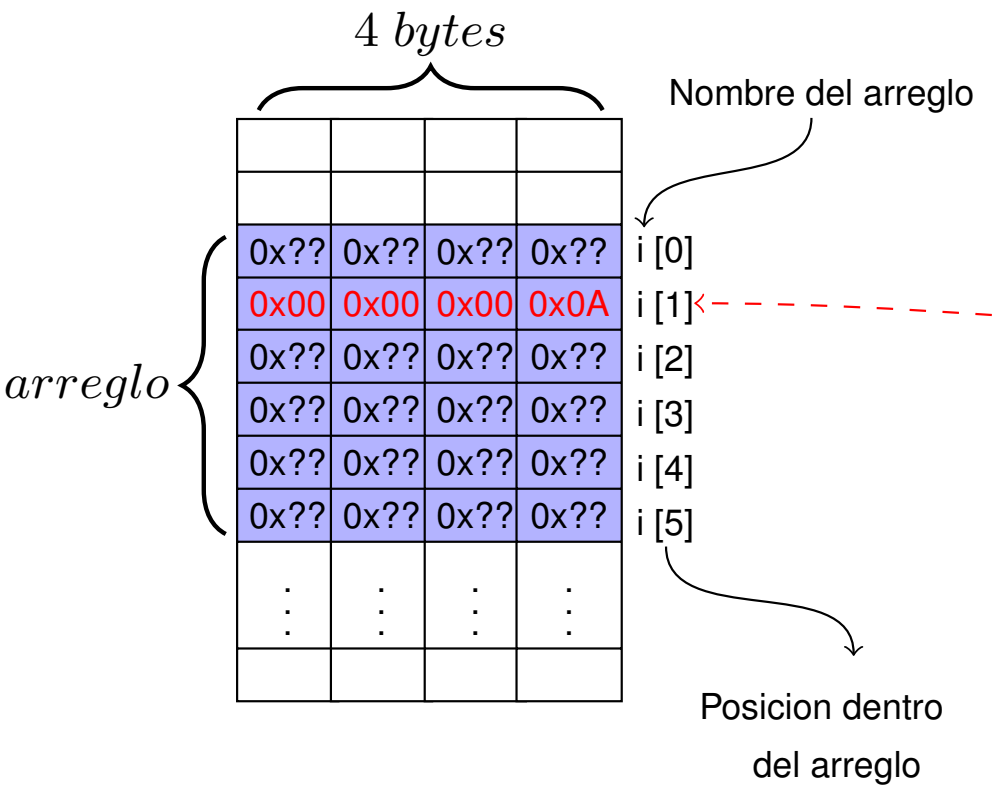
```
1   int main ()
2   {
3     int i[6];
4     .......
5     i[1]=10;
6     .......
7   }
```

Declara arreglo de 6 enteros

Acceso al contenido

Arquitectura X86-32 bits

Disposición de la variable i en memoria



Código en programa fuente

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

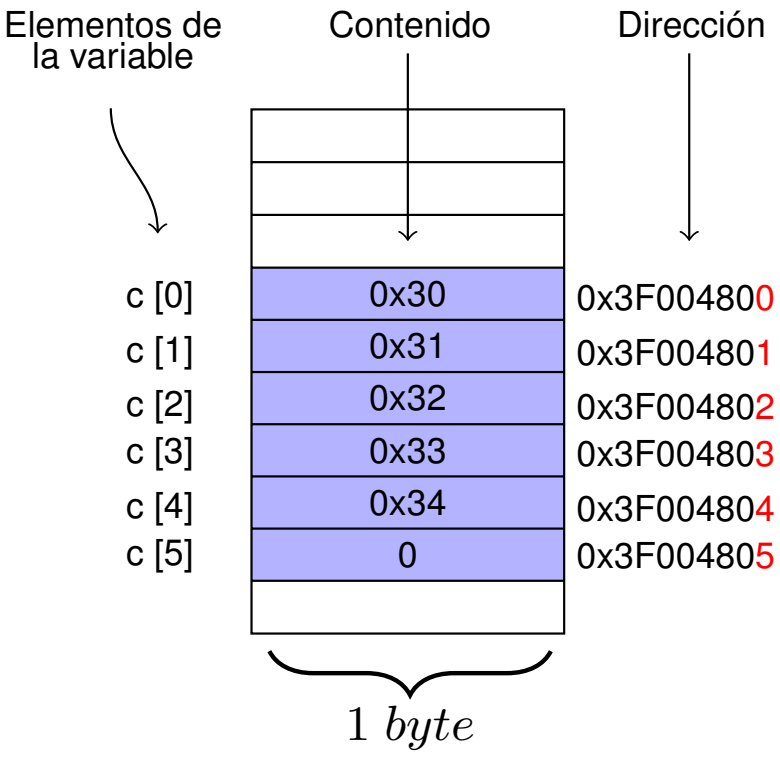
Asigna el valor 10 al elemento 1 del arreglo

Direción no es lo mismo que orden de elemento

Declaración en código

unsigned char c[6]

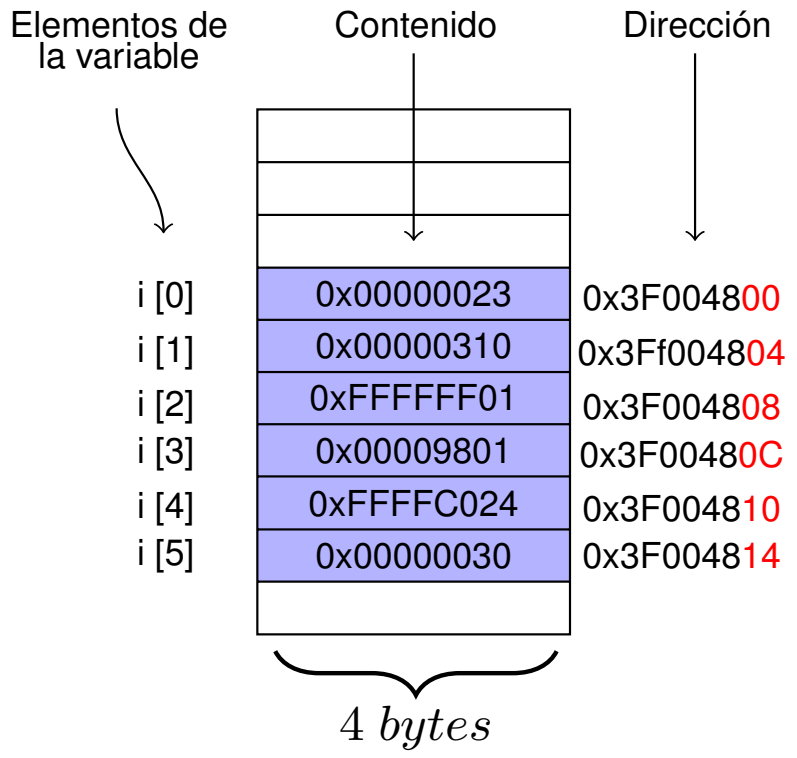
Cardena de caracteres



Declaración en código

int c[6]

Arreglo de enteros

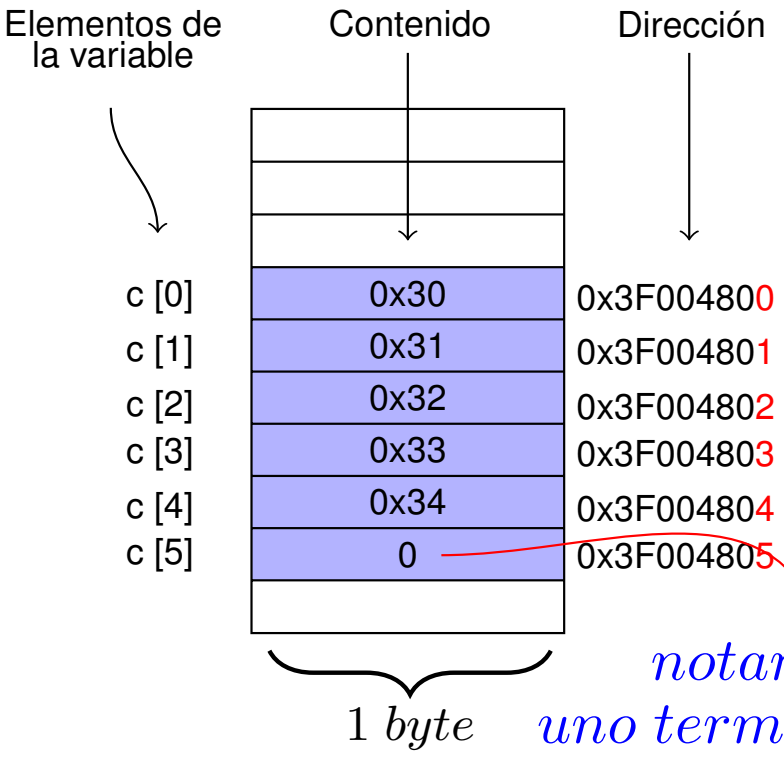


Direción no es lo mismo que orden de elemento

Declaración en código

unsigned char c[6]

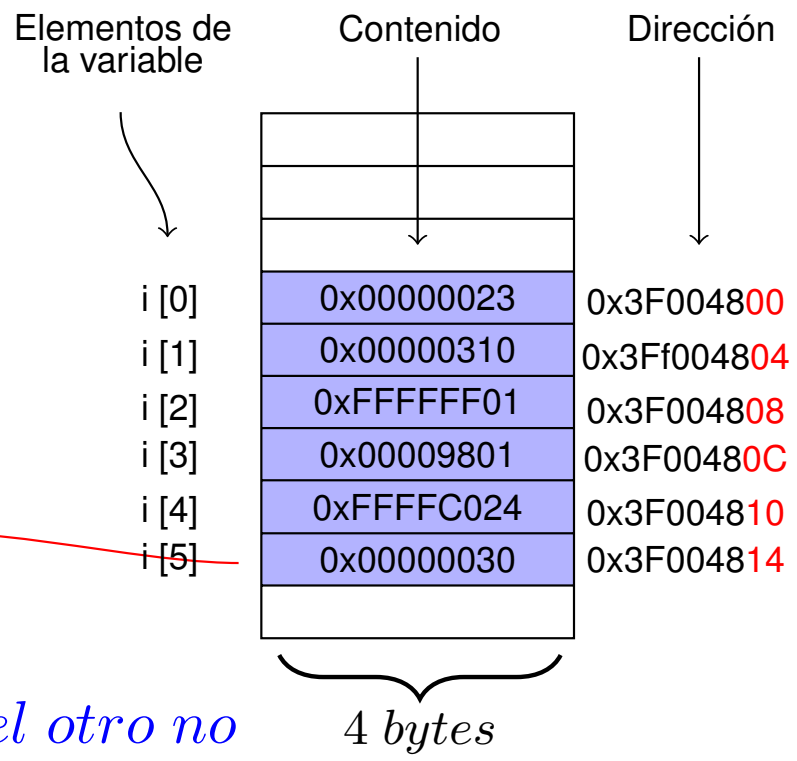
Cardena de caracteres



Declaración en código

int c[6]

Arreglo de enteros



notar la diferencia

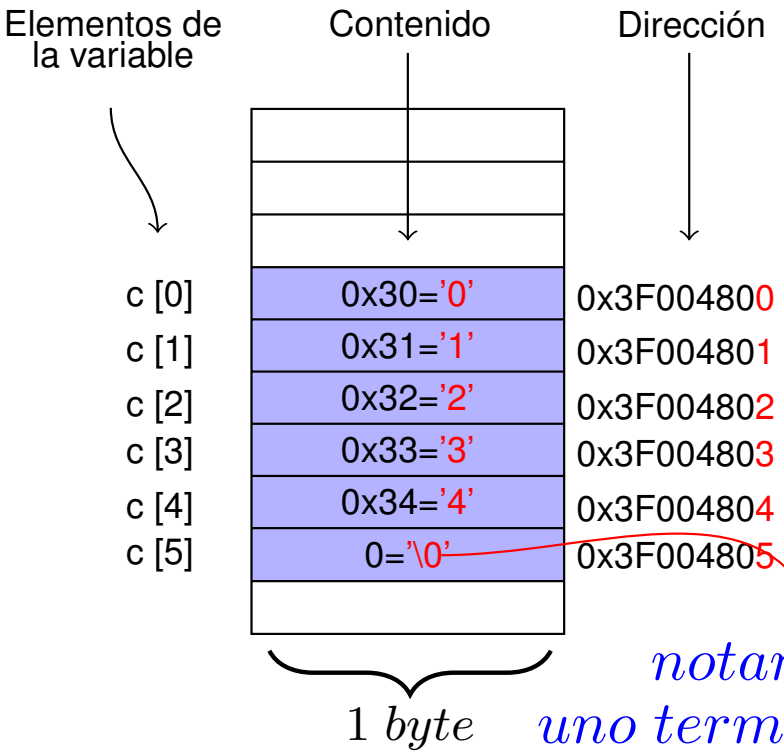
uno termina con '\0' y el otro no

Direción no es lo mismo que orden de elemento

Declaración en código

unsigned char c[6]

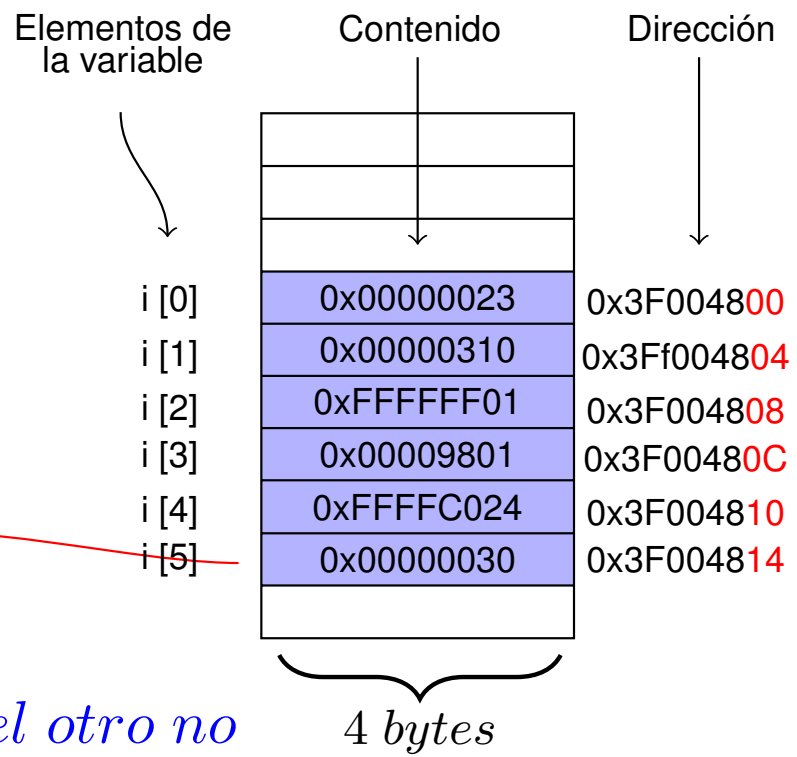
Cardena de caracteres



Declaración en código

int c[6]

Arreglo de enteros



notar la diferencia

uno termina con '\0' y el otro no

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 areglo 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 areglo 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'};
```

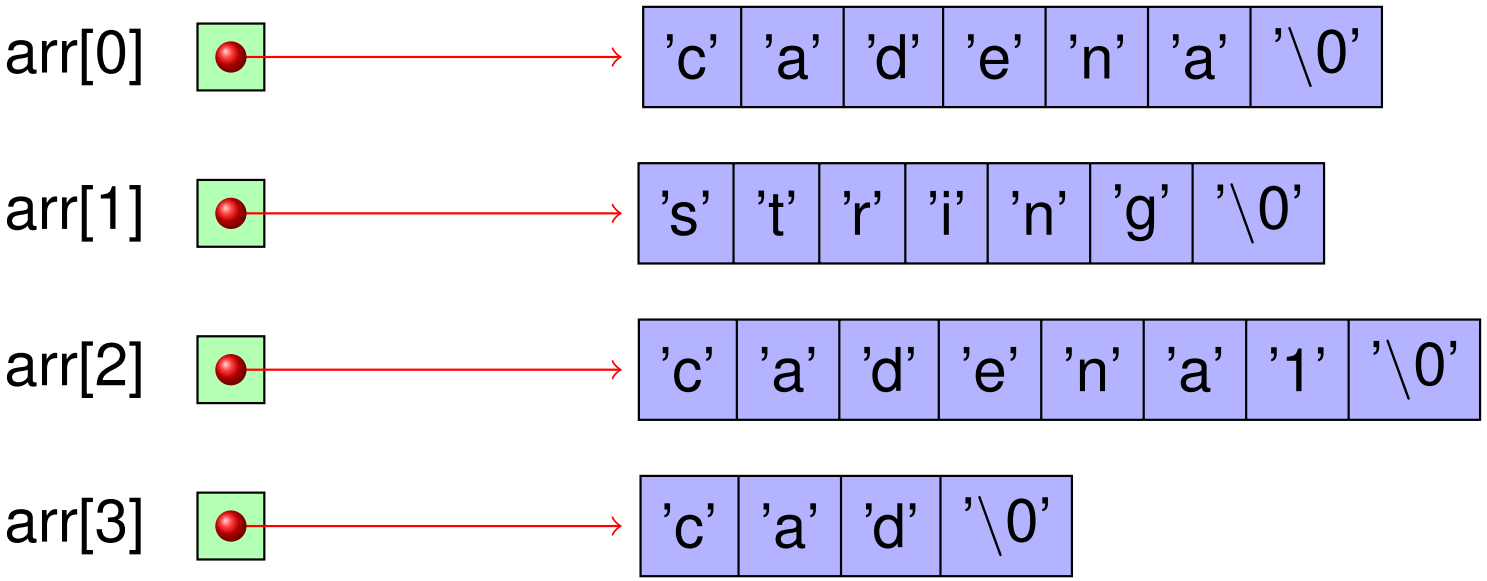
En código....

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

Arreglos de punteros

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

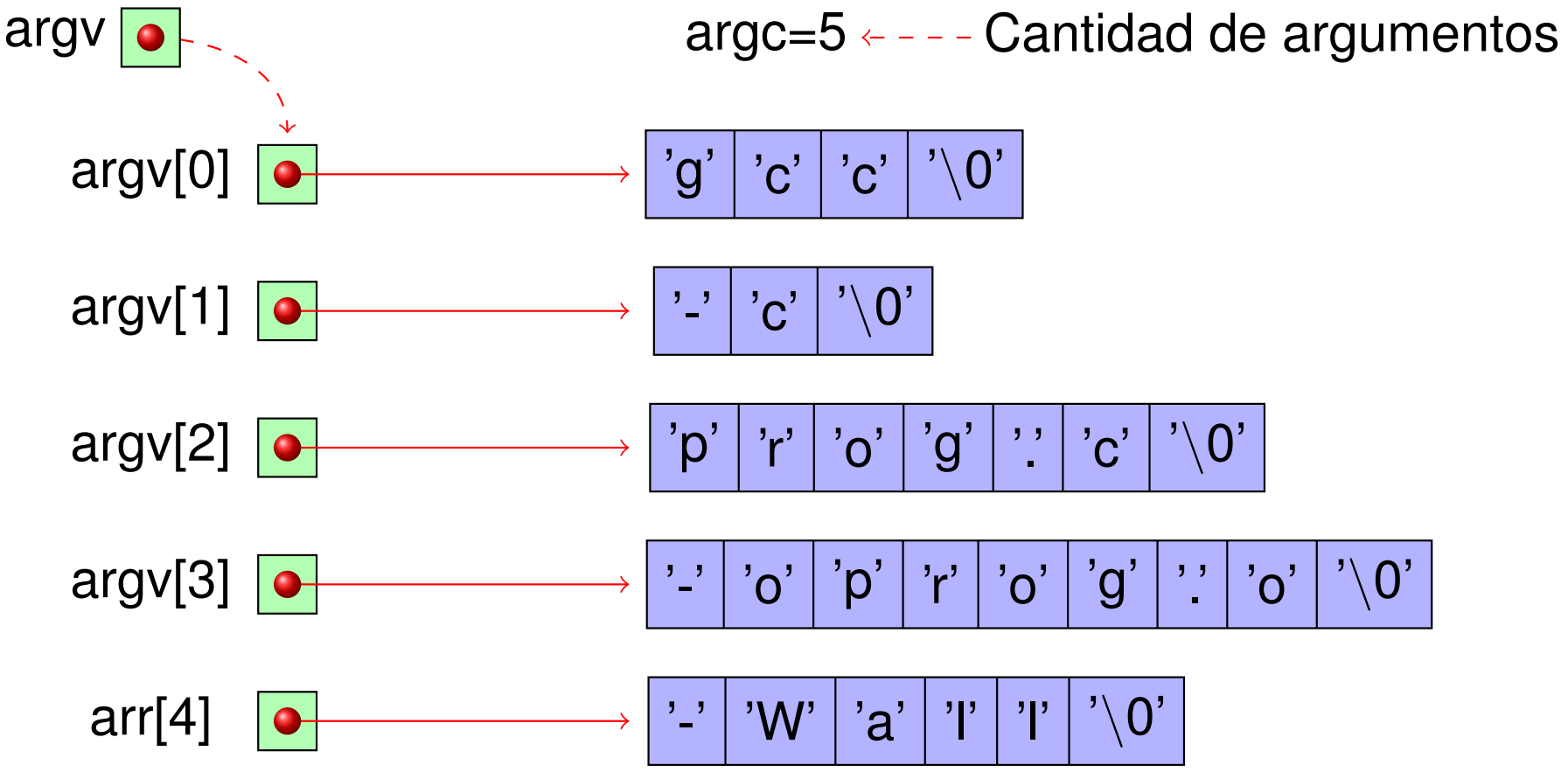
Genera esta disposición en memoria



main, la función multifacética

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

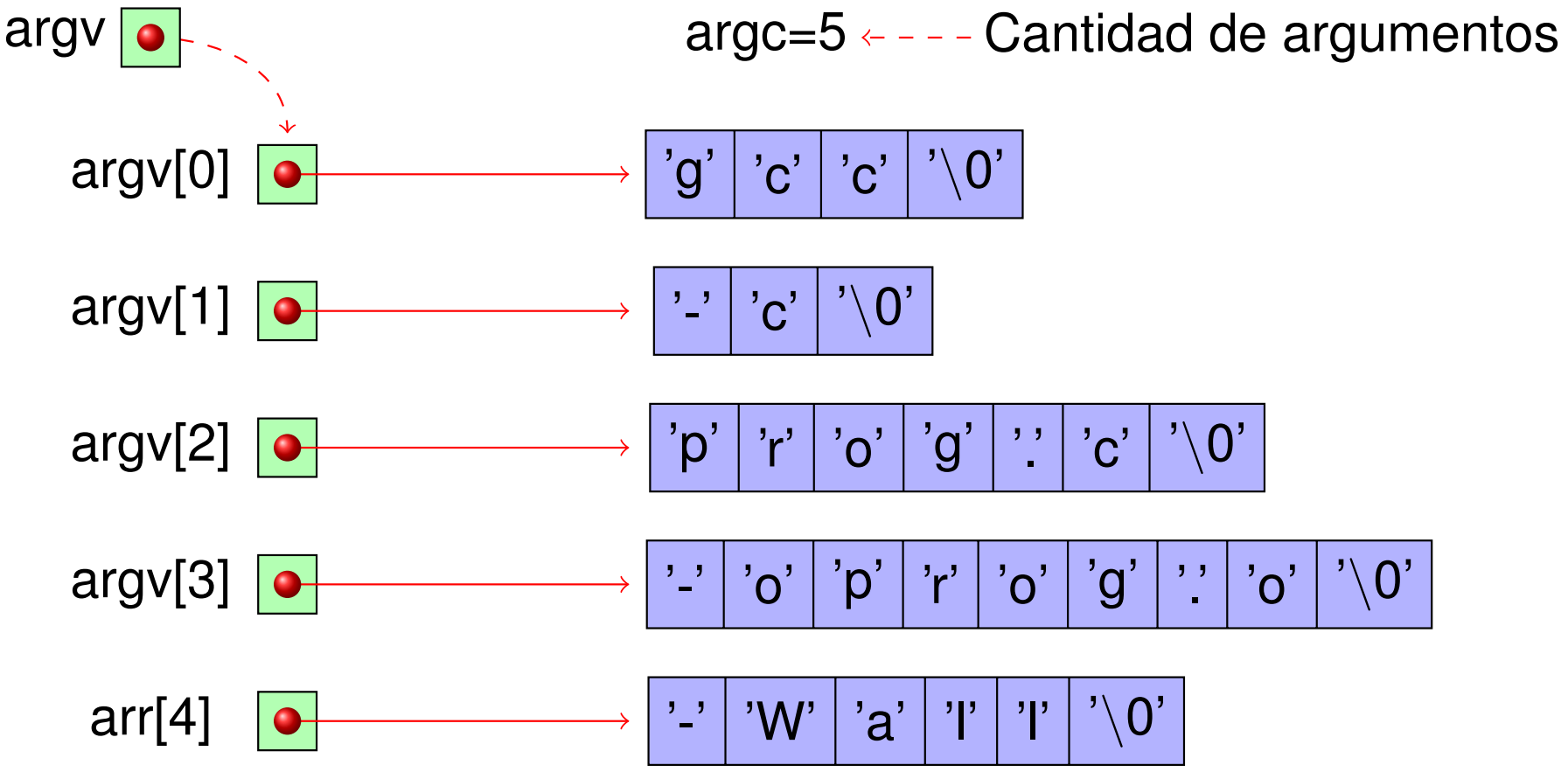
int main (int argc,char *argv[])



main, la función multifacética

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

int main (int argc,char **argv)

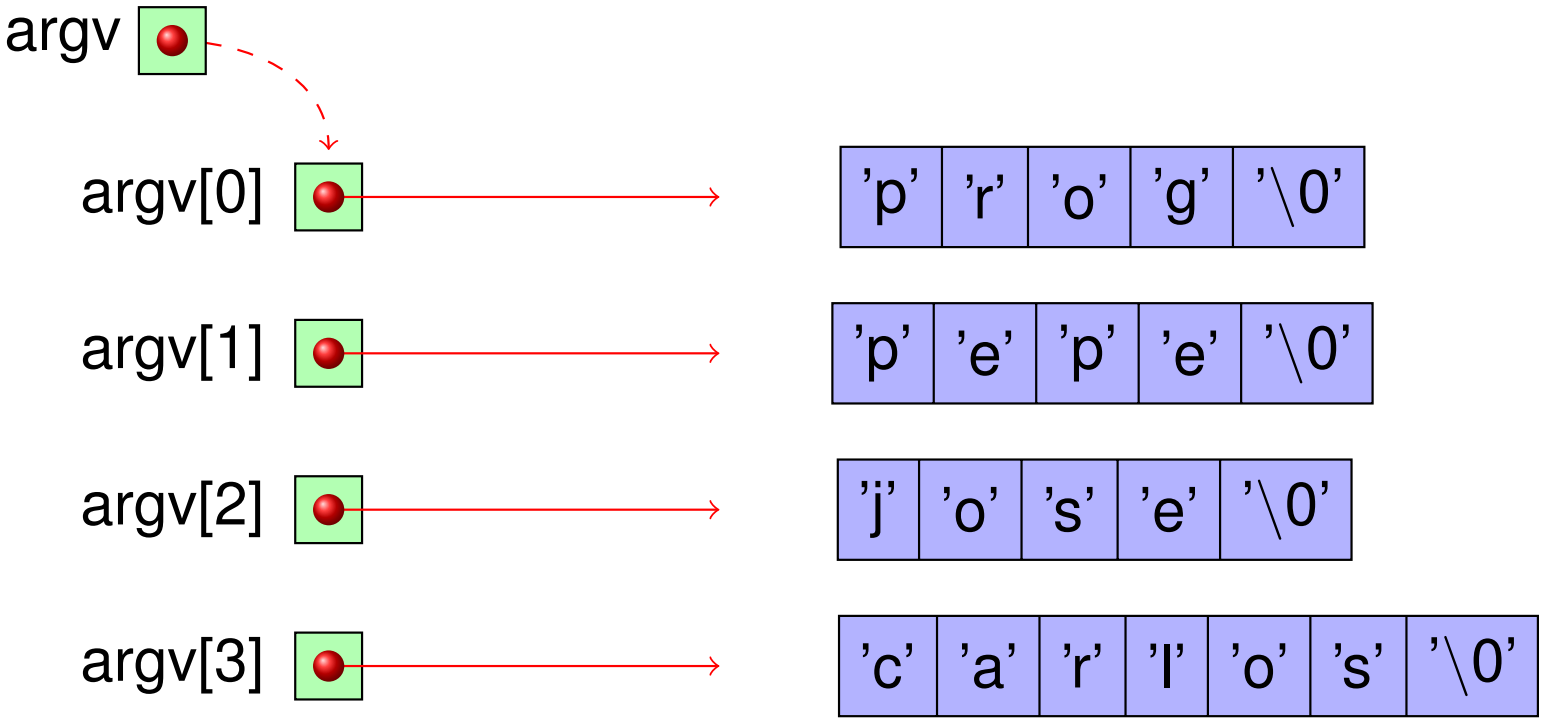


Ejemplo de *array de direcciones*

./prog *pepe jose carlos*

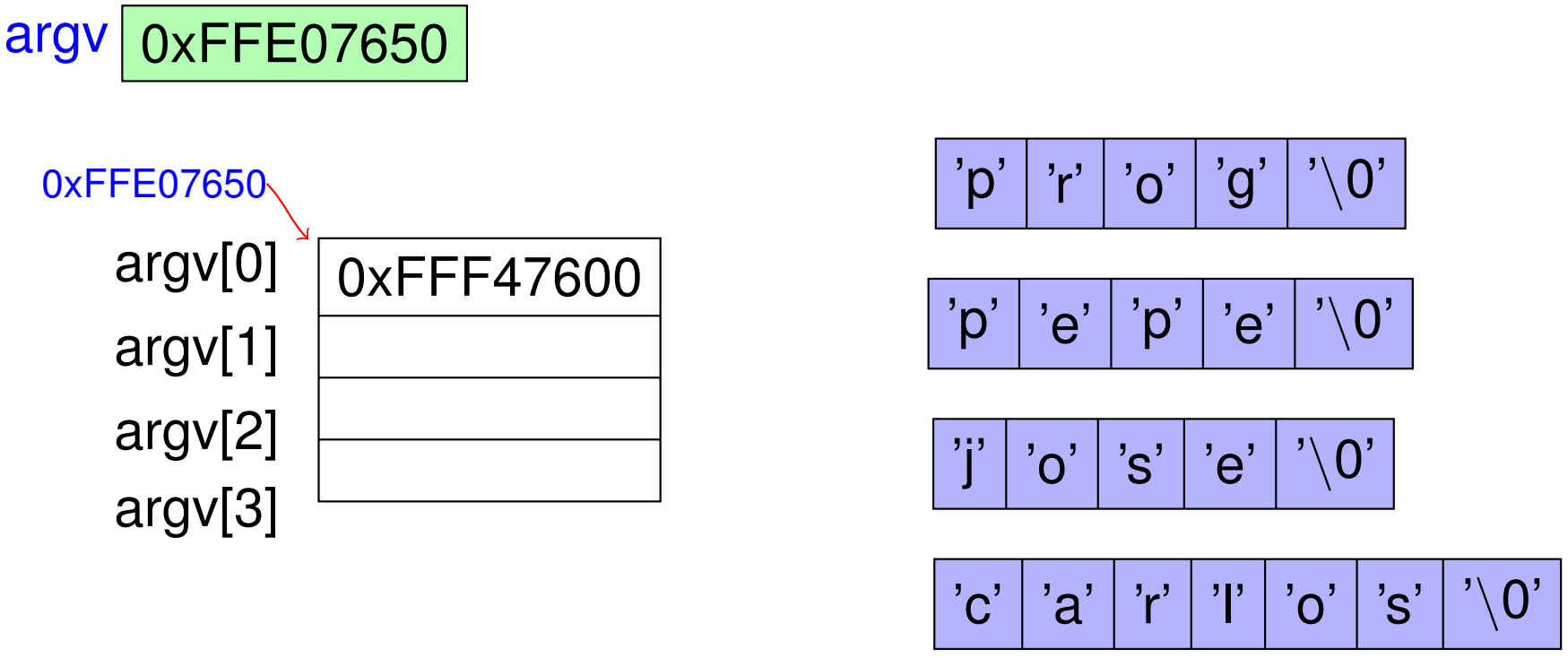
Los argumentos del programa son:

- pepe
- jose
- carlos



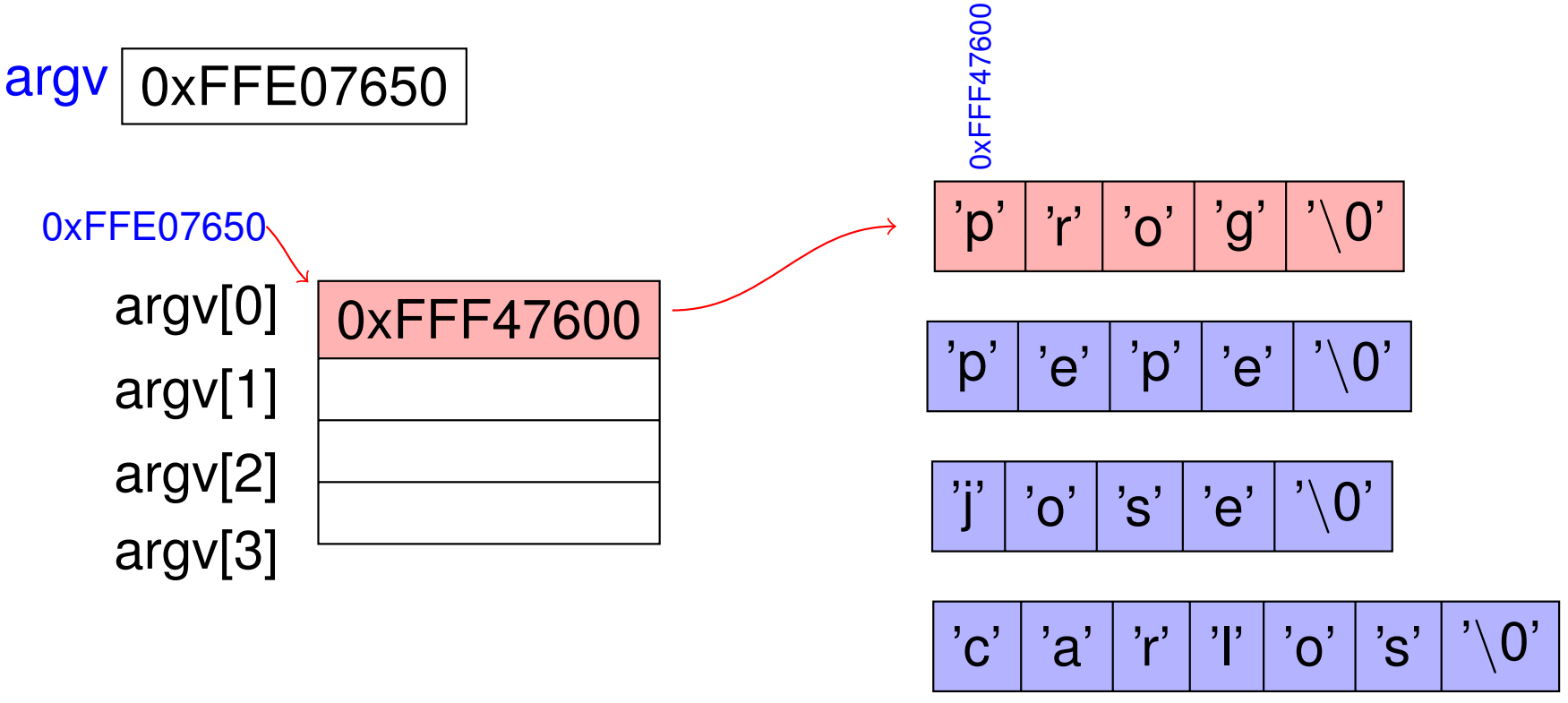
Ejemplo de *array de direcciones*

Arquitectura X86-32 bits



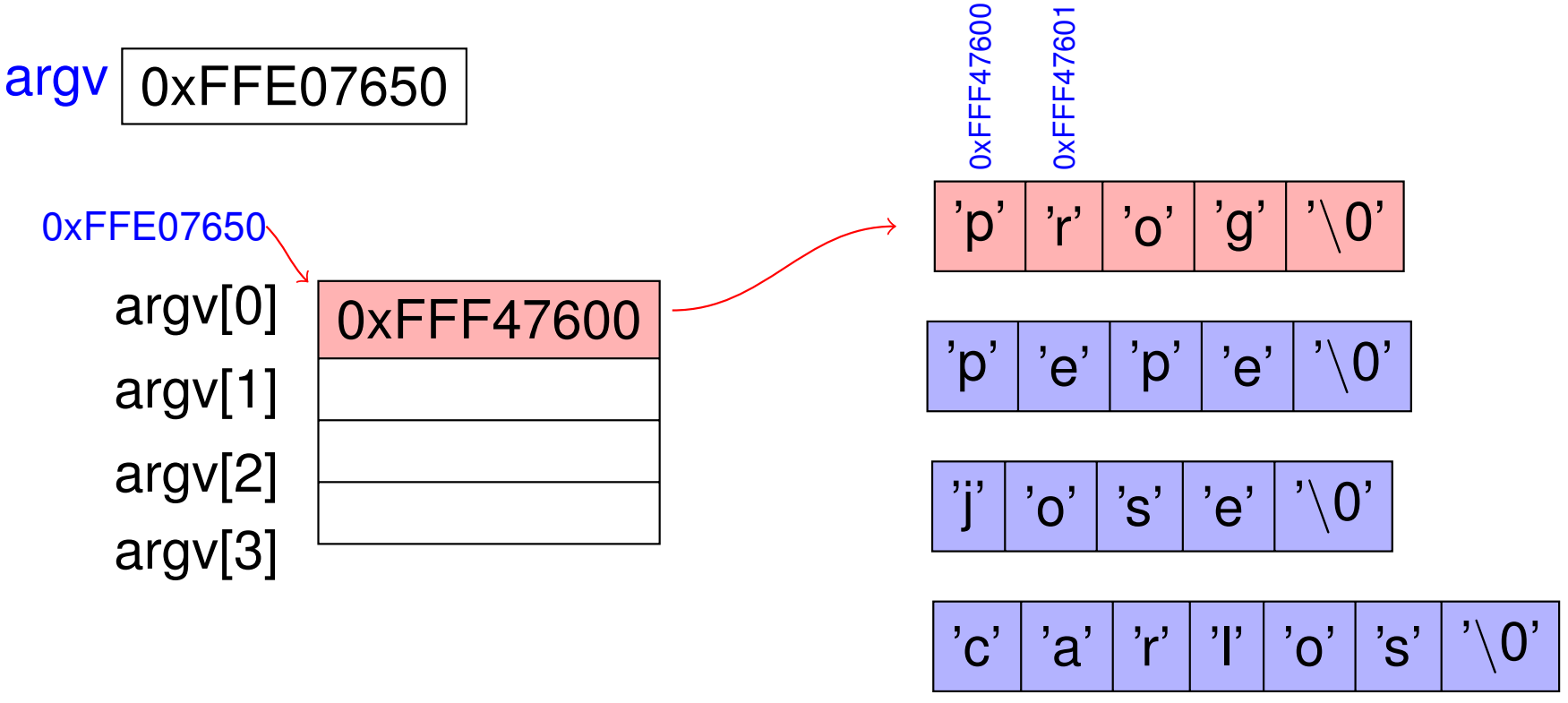
Ejemplo de *array de direcciones*

Arquitectura X86-32 bits



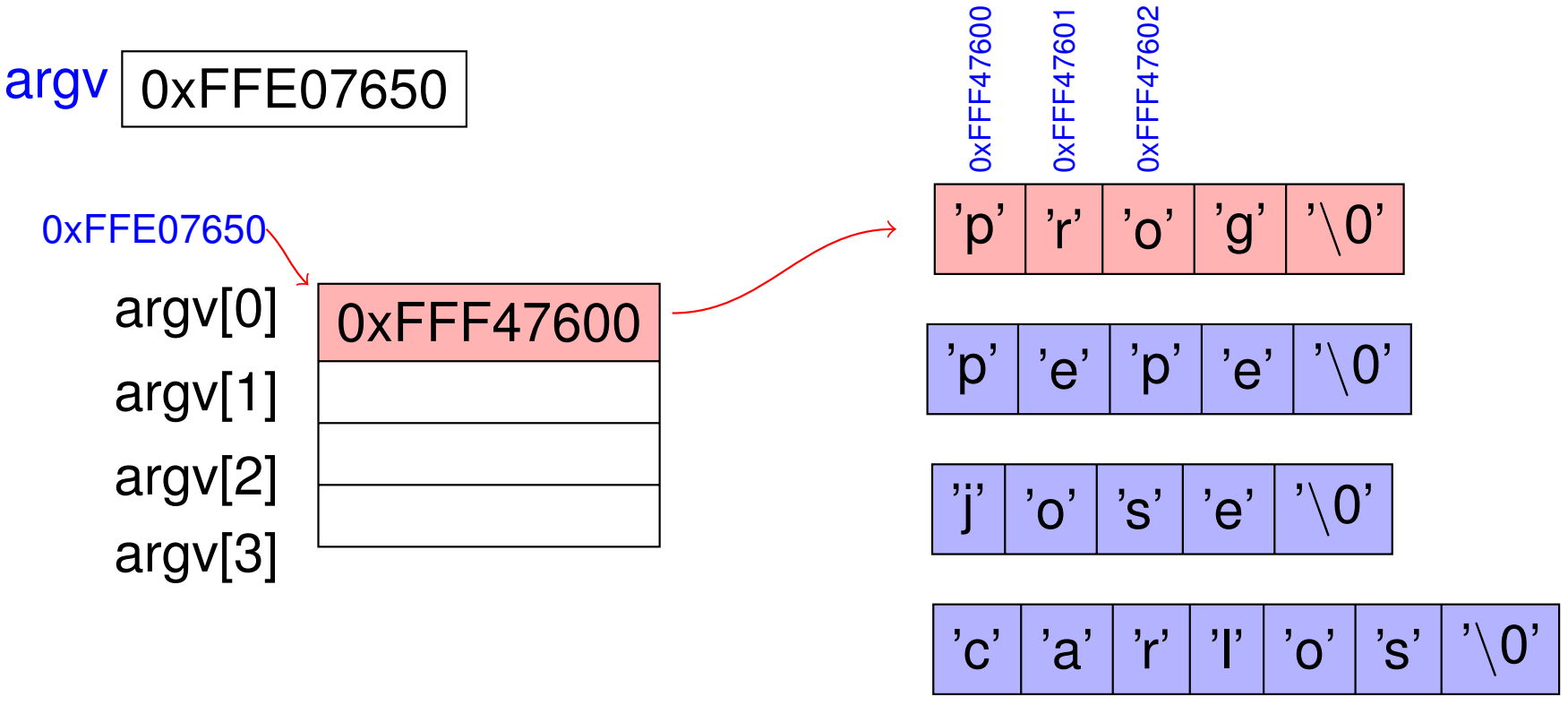
Ejemplo de *array de direcciones*

Arquitectura X86-32 bits



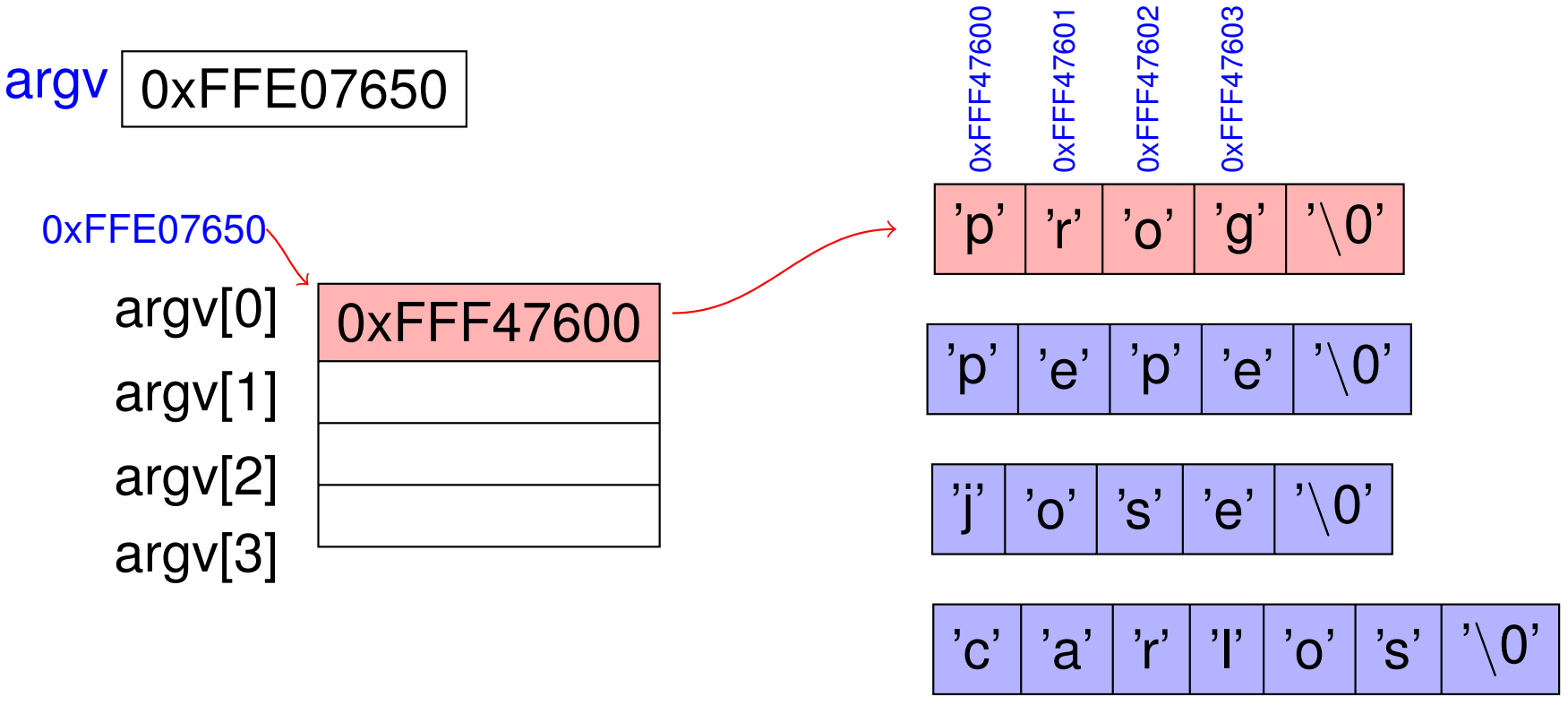
Ejemplo de *array de direcciones*

Arquitectura X86-32 bits



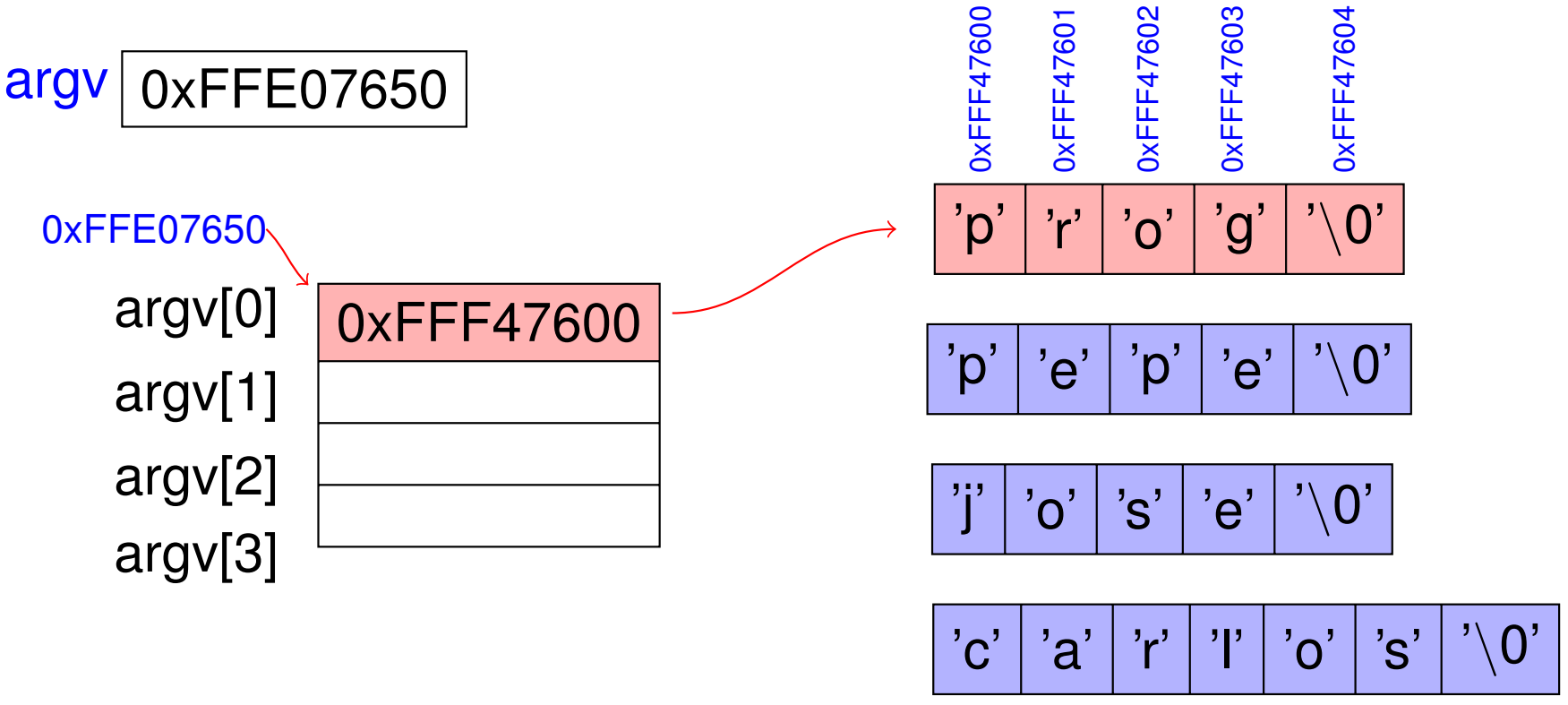
Ejemplo de *array de direcciones*

Arquitectura X86-32 bits



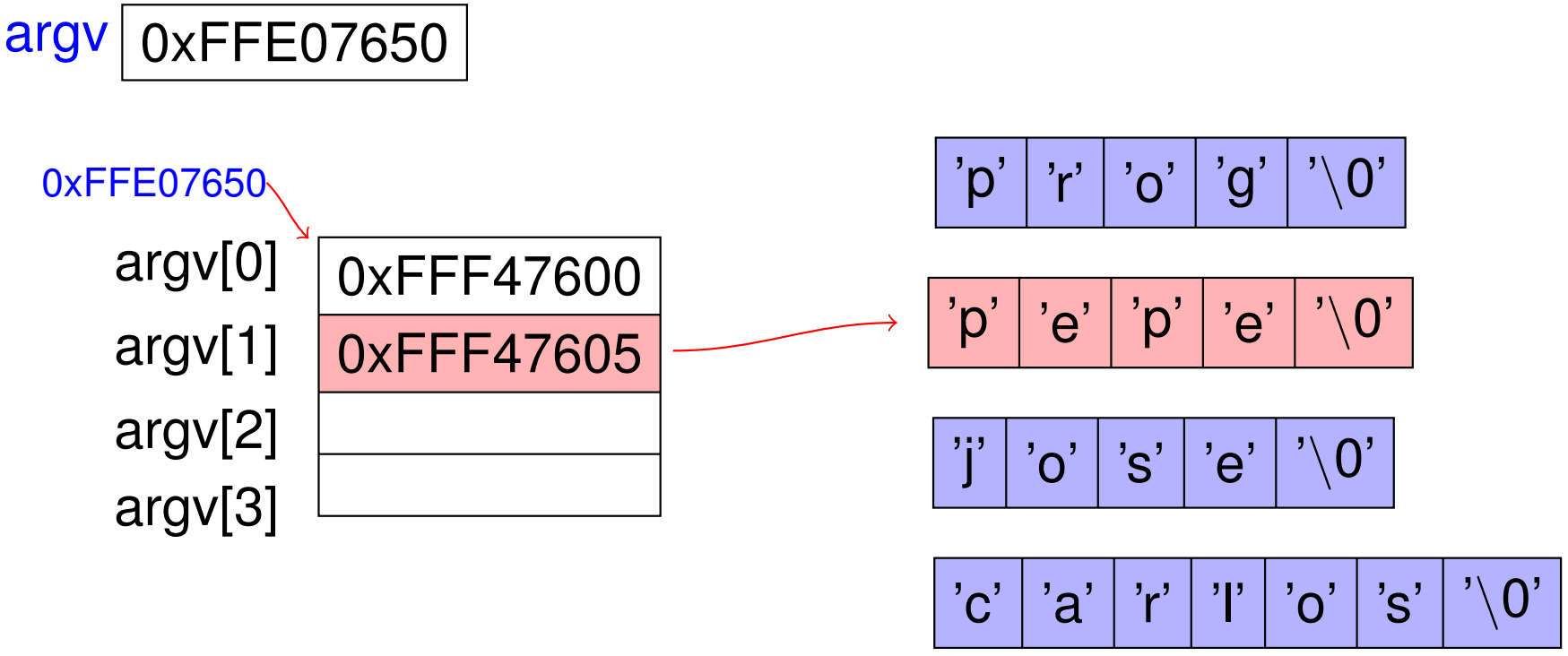
Ejemplo de *array de direcciones*

Arquitectura X86-32 bits



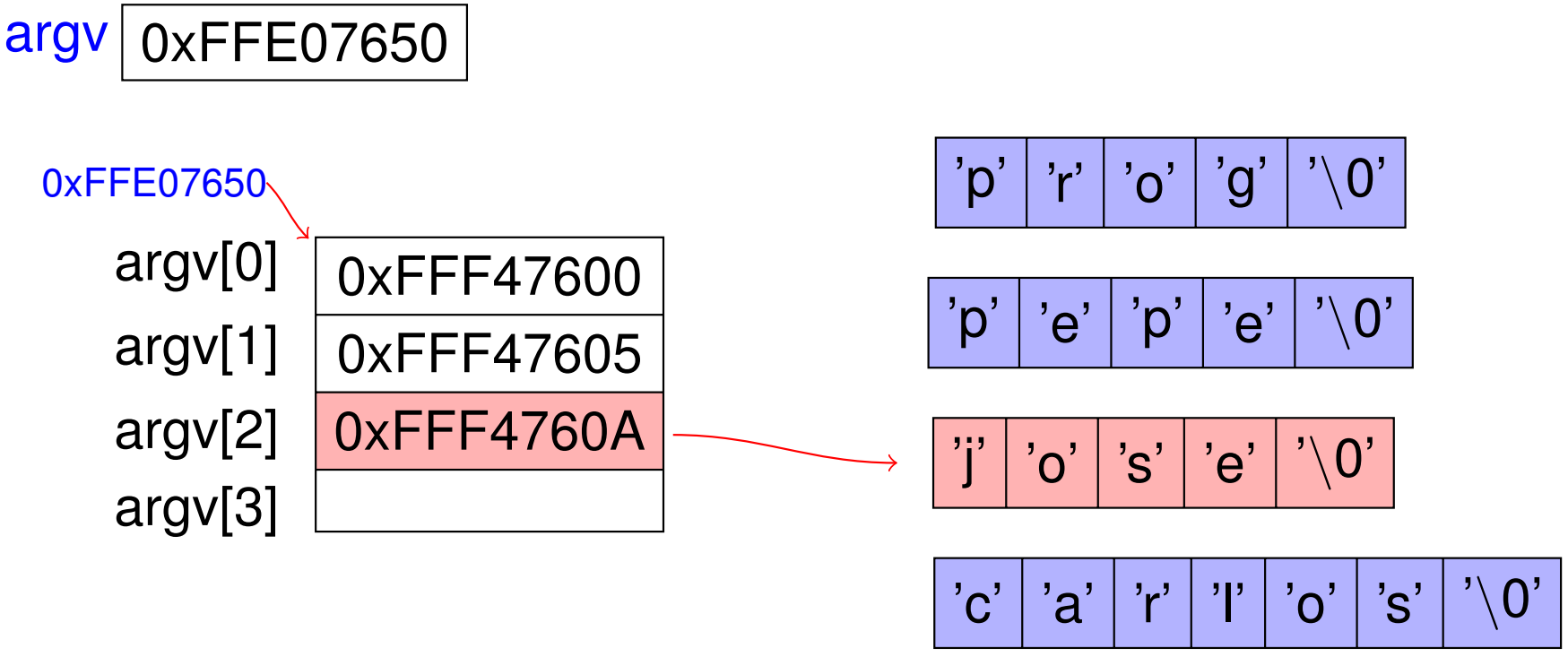
Ejemplo de *array de direcciones*

Arquitectura X86-32 bits



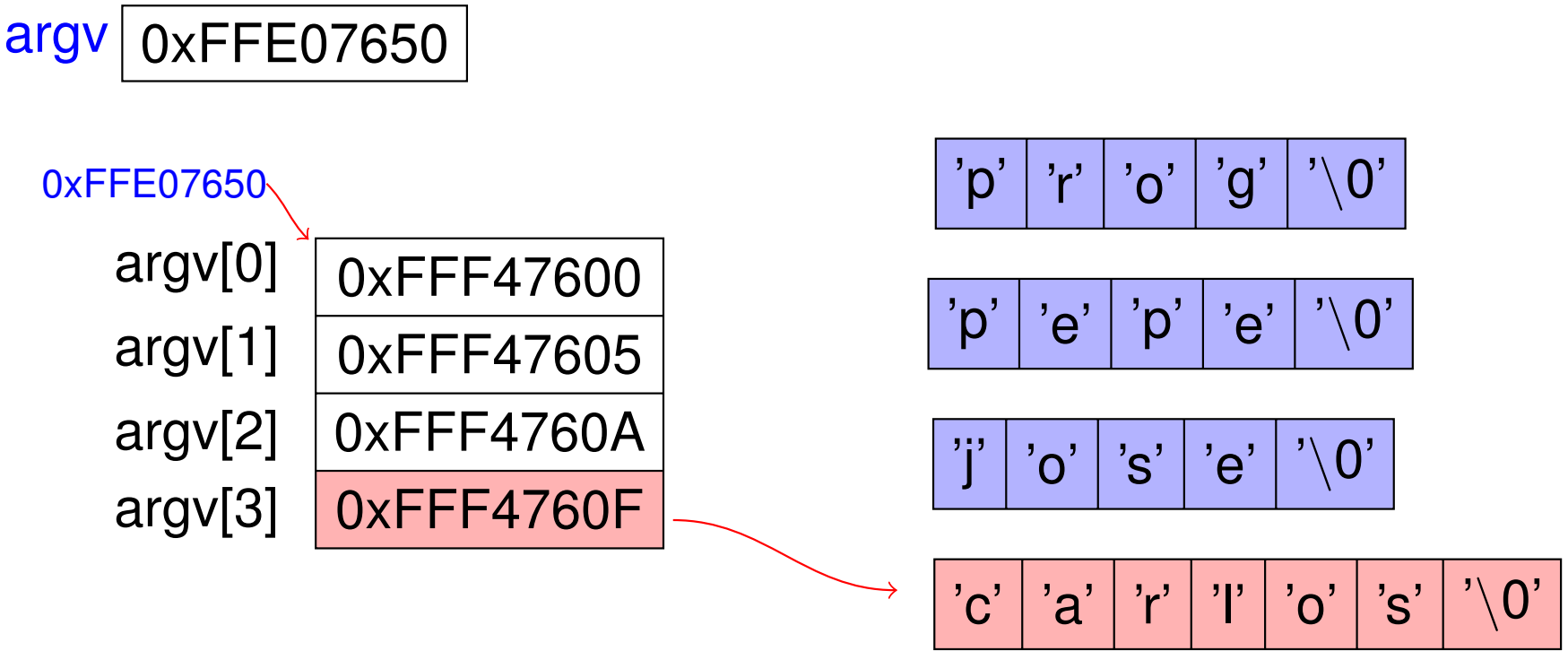
Ejemplo de *array de direcciones*

Arquitectura X86-32 bits

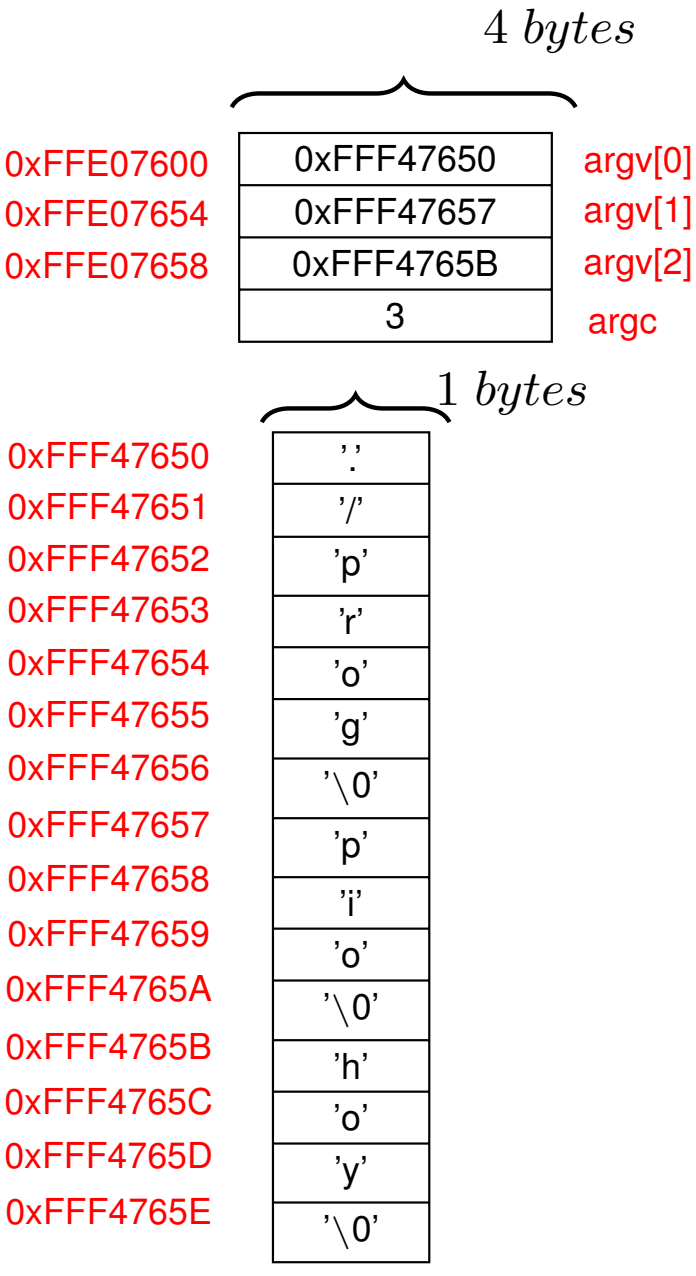


Ejemplo de *array de direcciones*

Arquitectura X86-32 bits



Argumentos por línea de comandos

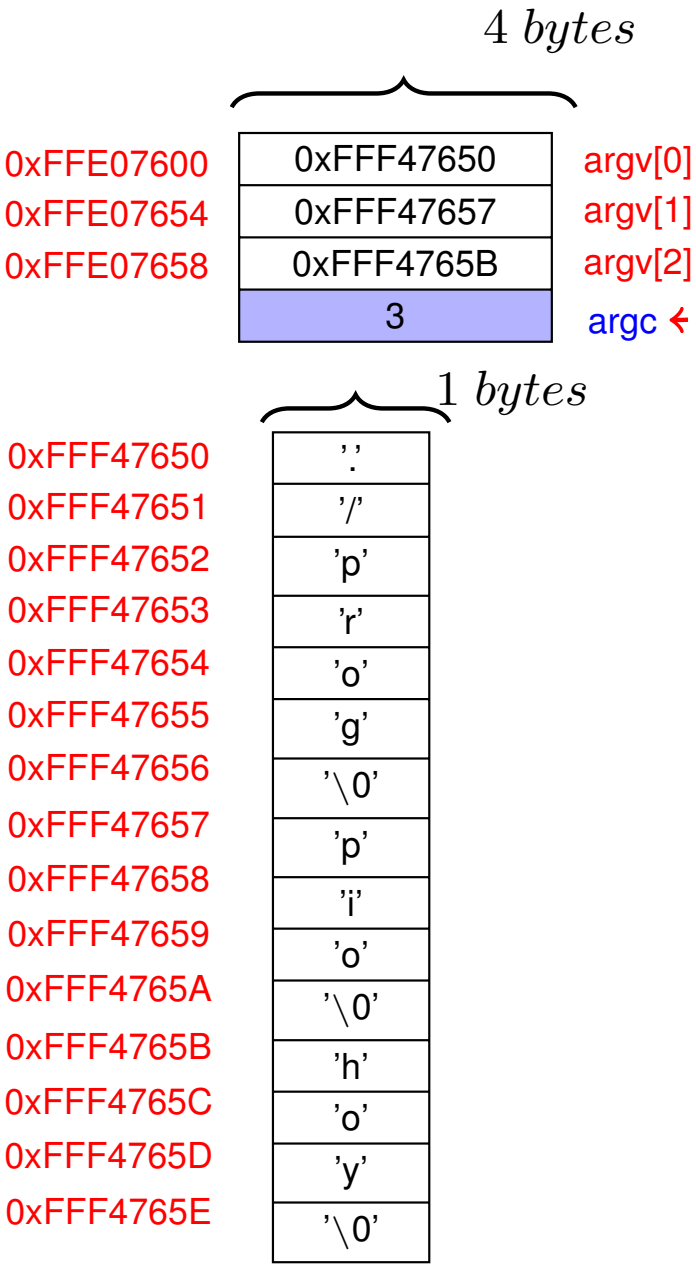


ejecutamos el programa

./prog pio hoy

```
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



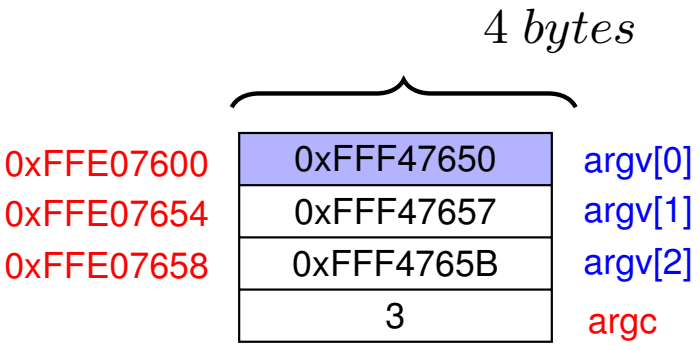
ejecutamos el programa

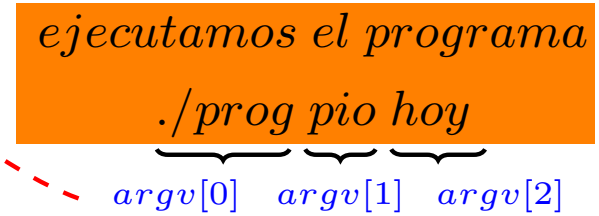
./prog pio hoy

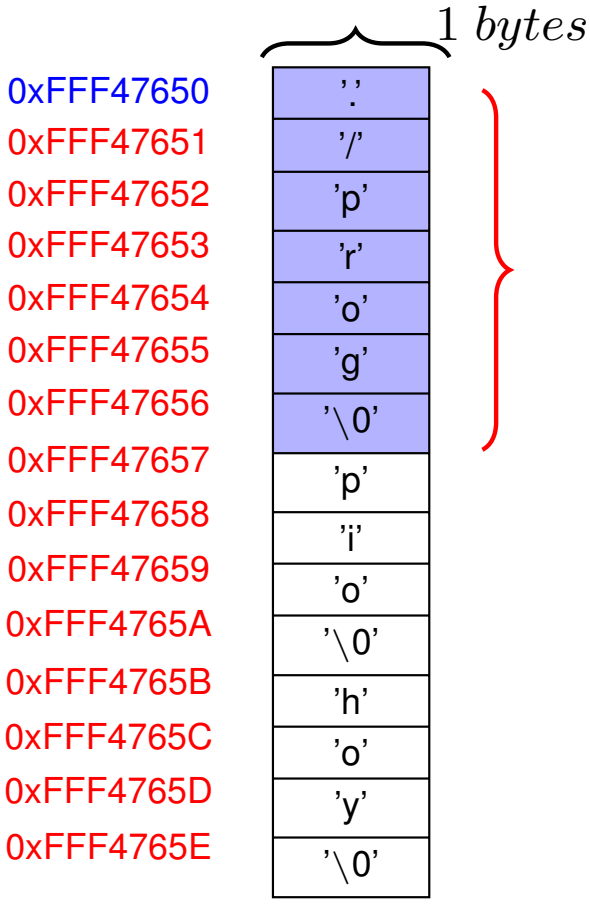
$argc = 3$

```
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

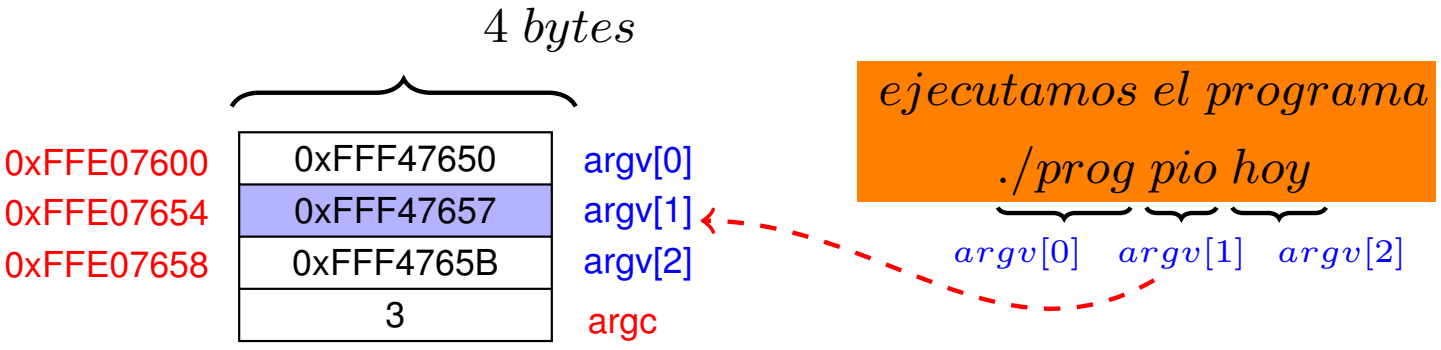


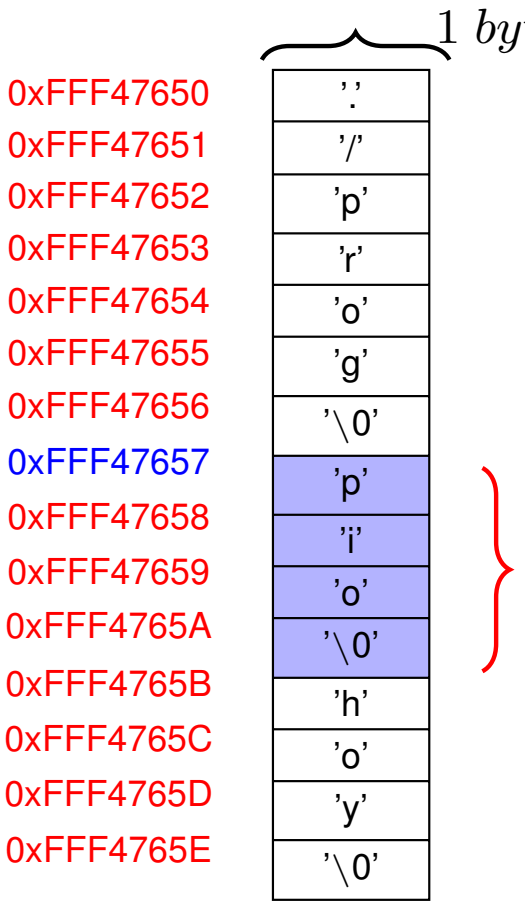





```
1  #include<stdio.h>
2
3
4  int main (int argc , char ** argv)
5  {
6      int      i;
7
8      printf("primer ejemplo\n");
9      printf("%d es argc\n",argc);
10     for ( i=0 ; i<argc ; i++ )
11     {
12         printf("%s \t" , *(argv+i) );
13     }
14 }
```

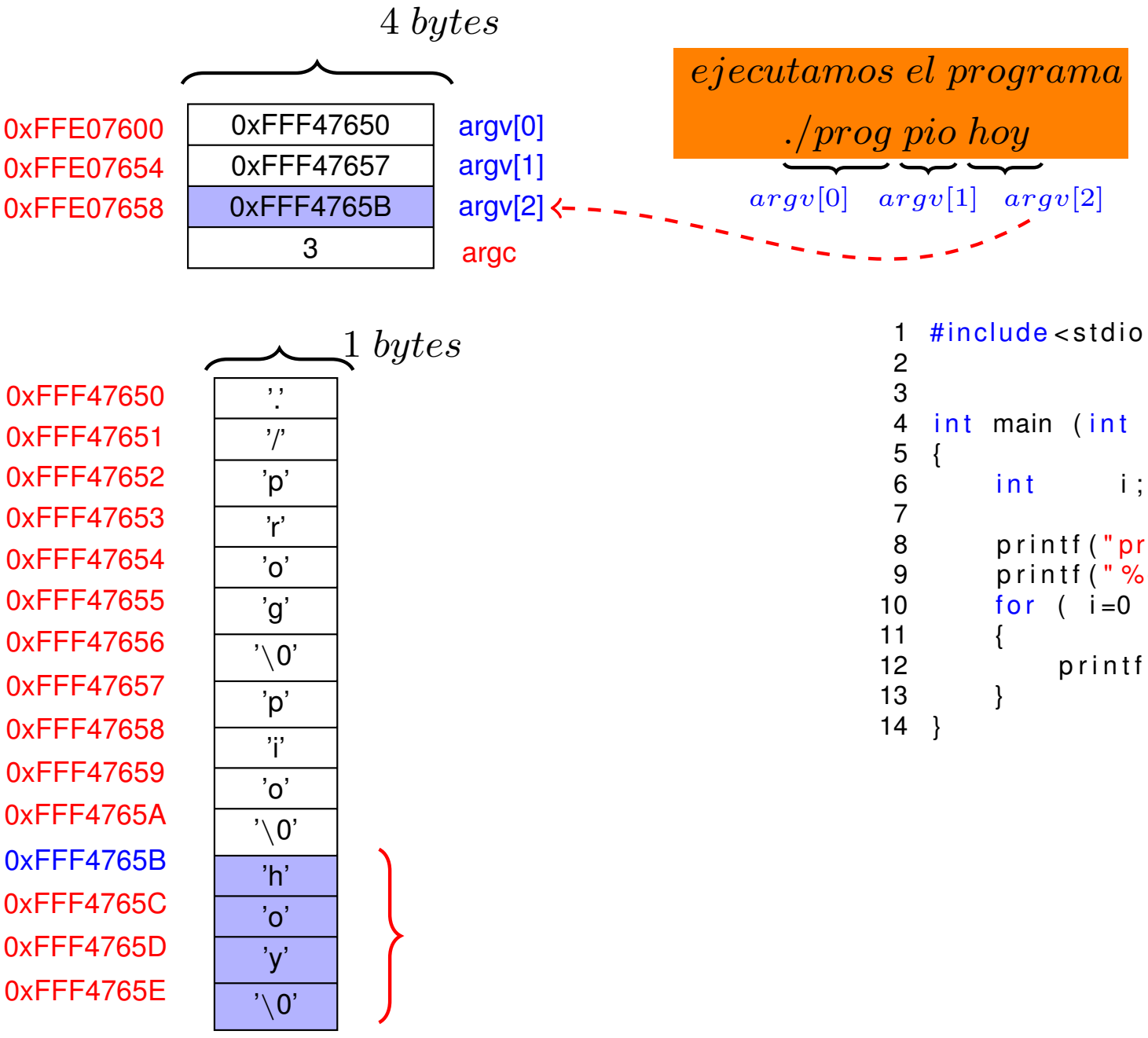
Argumentos por línea de comandos





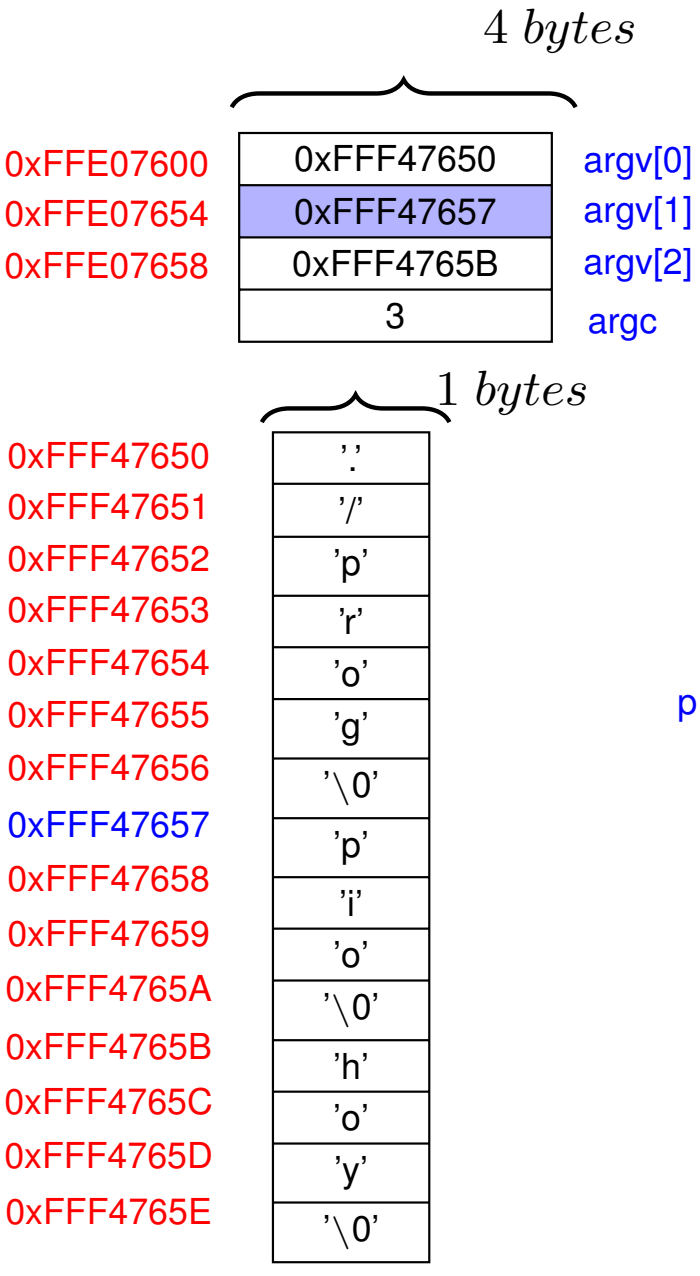

```
1  #include<stdio.h>
2
3
4  int main (int argc , char ** argv)
5  {
6      int      i;
7
8      printf("primer ejemplo\n");
9      printf("%d es argc\n",argc);
10     for ( i=0 ; i<argc ; i++ )
11     {
12         printf("%s \t" , *(argv+i) );
13     }
14 }
```

Argumentos por línea de comandos




```
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



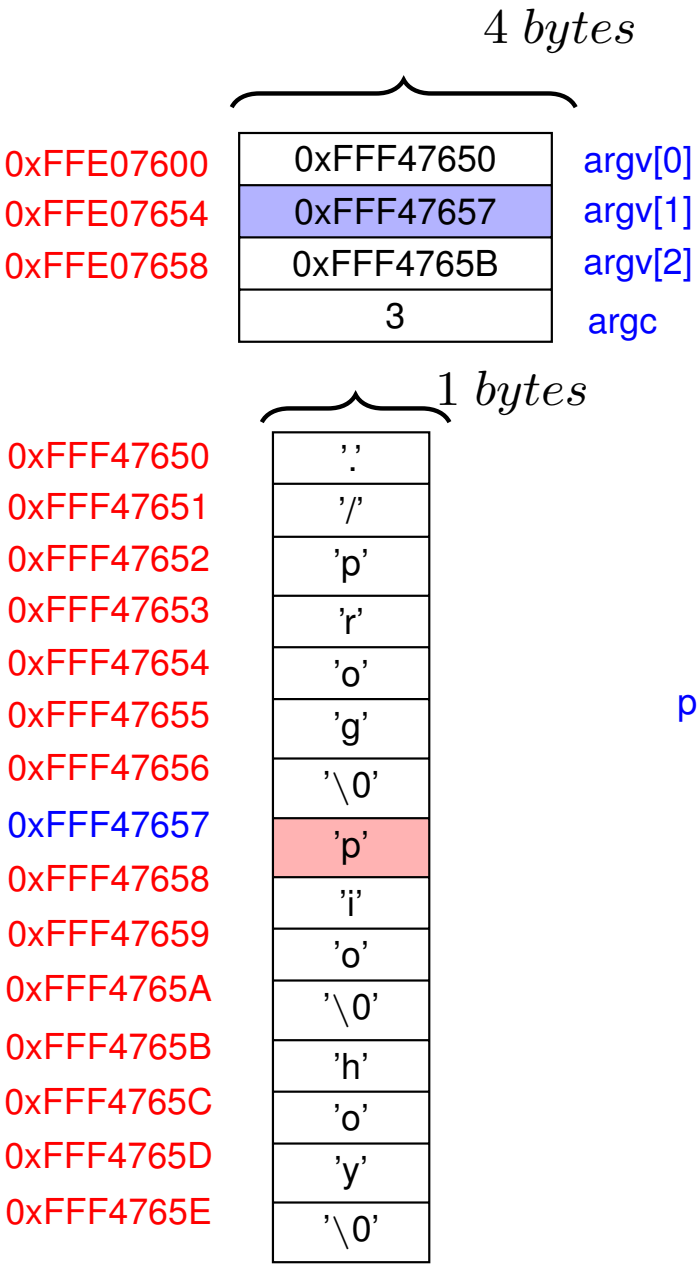
ejecutamos el programa

./prog pio hoy

p = 0xFFF47657

```
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

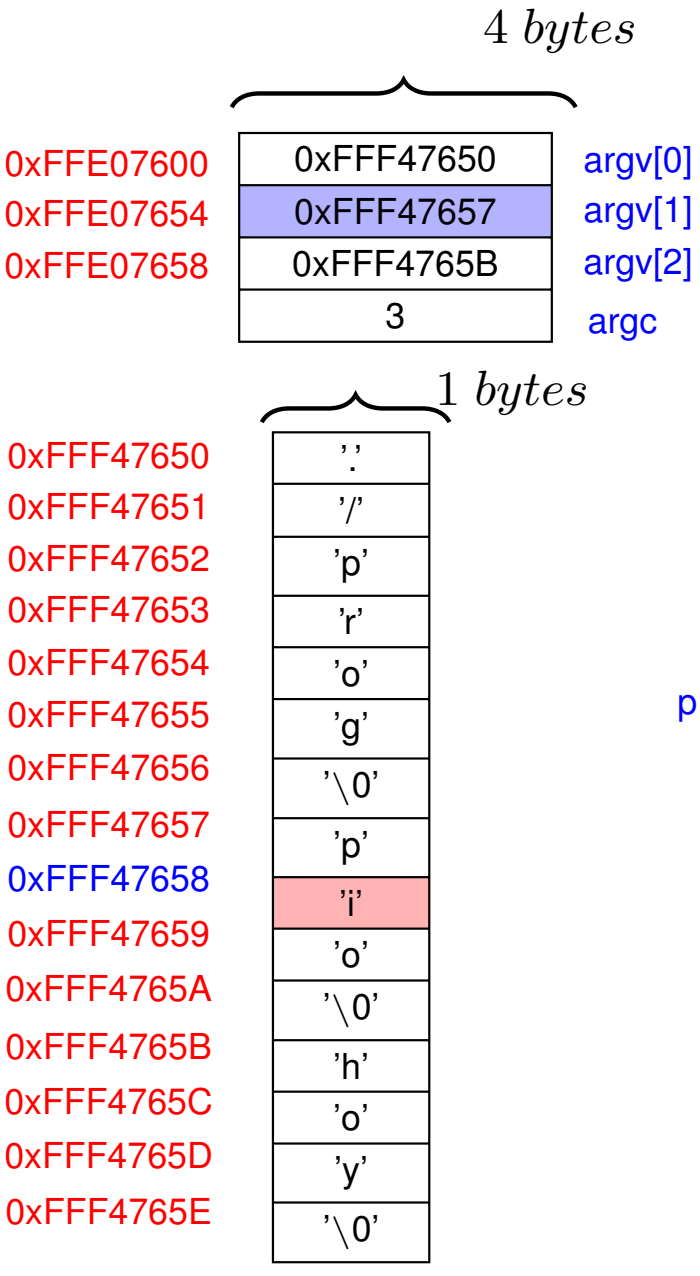
./prog pio hoy

p = 0xFFF47657

*(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

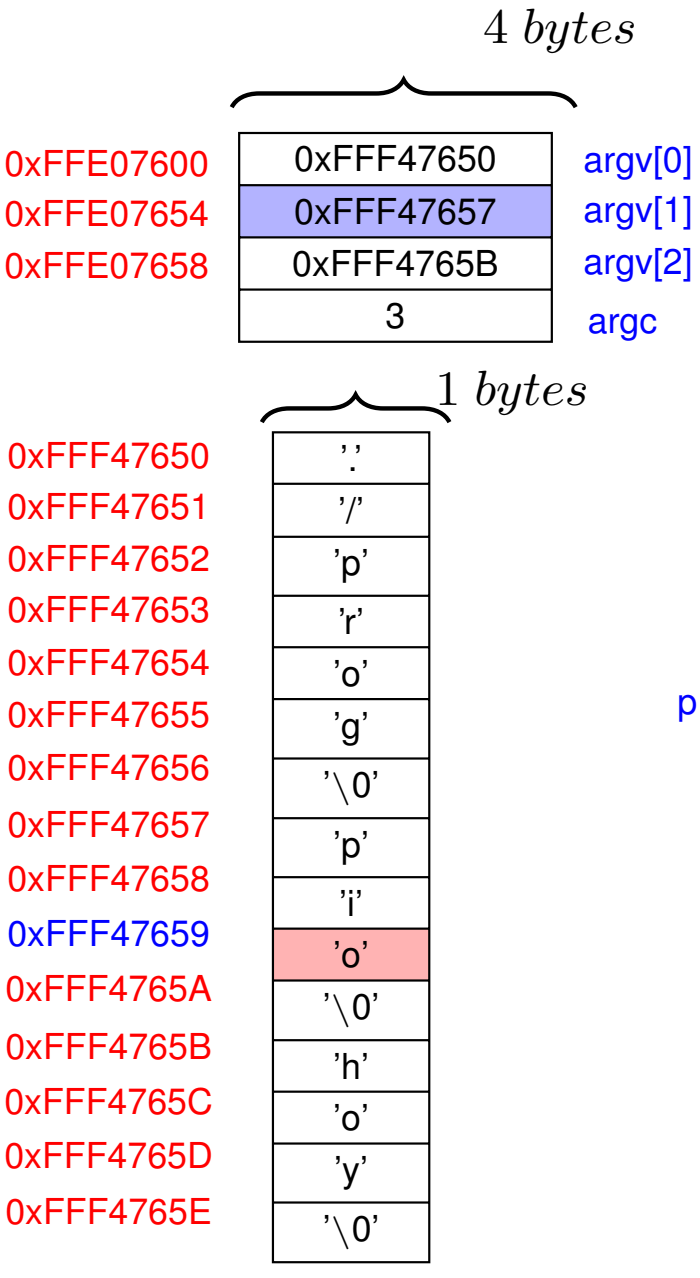
p = 0xFFF47657

*(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

p = 0xFFF47657

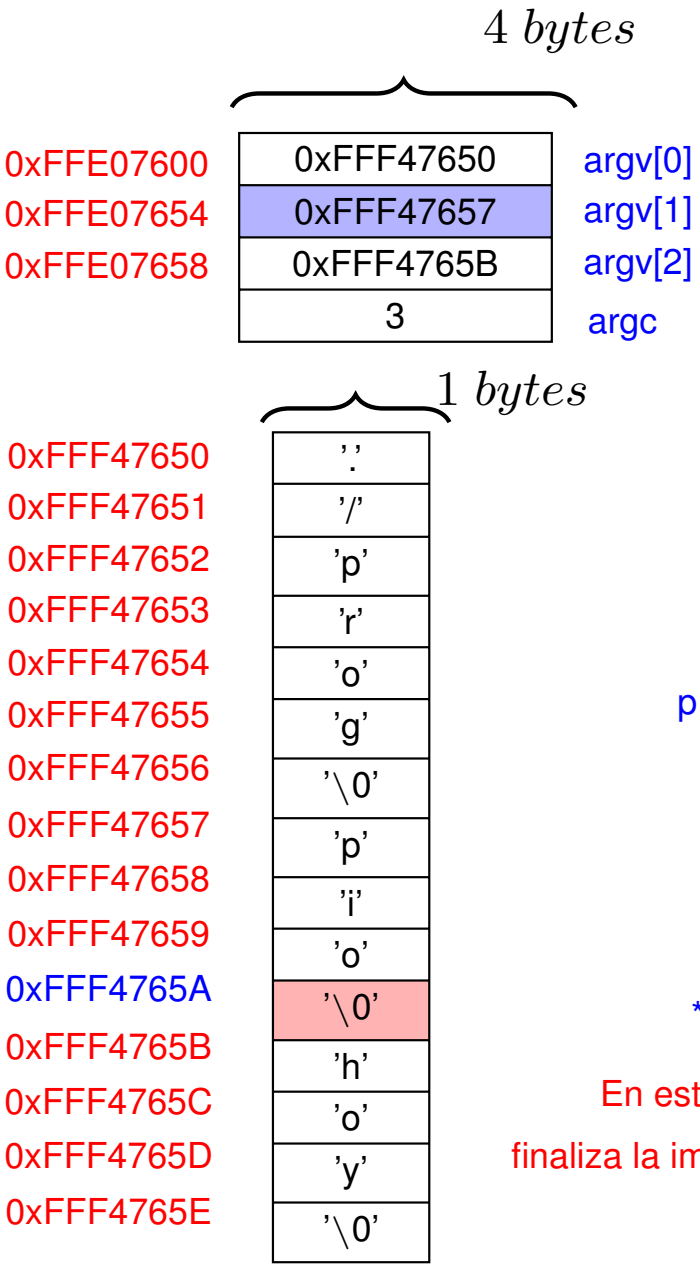
*(p+0) –> 'p'

*(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

```
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 }
```

p = 0xFFF47657

*(p+0) –> 'p'

*(p+1) –> 'i'

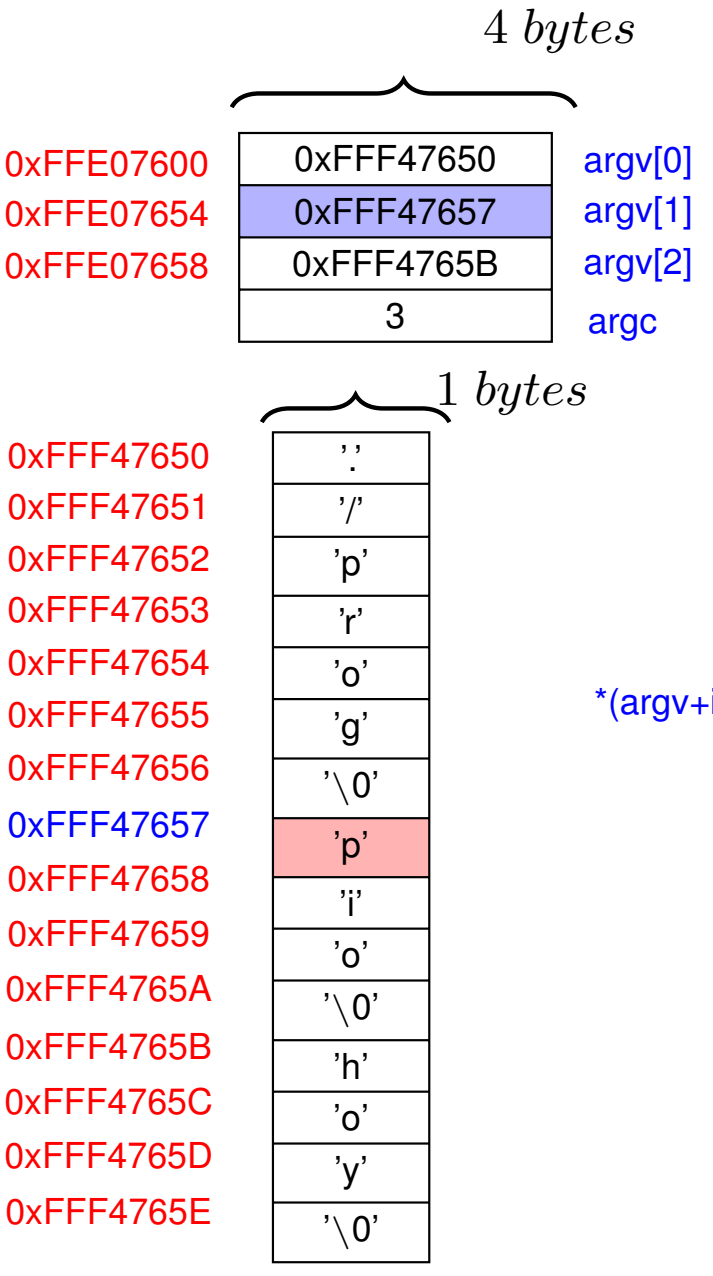
*(p+2) –> 'o'

*(p+3) –> '\0'

En este caso sale del for y

finaliza la impresion de los caracteres

Accediendo a los caracteres con una función



ejecutamos el programa

./prog pio hoy

*(argv+i) –> = 0xFFF47657

```
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 }
```