#include<stdio.h>
#include<string.h>

int main(void)
{
    char arr[4];                           // for accommodating 3 characters and one null '\0' byte
    char *ptr = "abc";                     // a string containing 'a', 'b', 'c', '\0'

    //reset all the bytes so that none of the byte contains any junk value
    memset(arr, '\0', sizeof(arr));

    strncpy(arr, ptr, sizeof("abc"));     // Copy the string "abc" into the array arr
    printf("\n %s \n",arr);                // print the array as string
    arr[0] = 'p';                          // change the first character in the array
    printf("\n %s \n",arr);                // again print the array as string
    return 0;
}
Dynamic Memory Functions

- Can be found in the stdlib.h library:
  - To allocate space for an array in memory you use
    - `calloc()`
  - To allocate a memory block you use
    - `malloc()`
  - To de-allocate previously allocated memory you use
    - `free()`

- Each function is used to initialize a pointer with memory from free store (a section of memory available to all programs)
The function `malloc()` will allocate a block of memory that is size bytes large. If the requested memory can be allocated a pointer is returned to the beginning of the memory block.

- **Note:** the content of the received block of memory is not initialized.

### Usage of `malloc()`:

- `void * malloc ( size_t size );`

### Parameters:

- Size of the memory block in bytes.

### Return value:

- If the request is successful then a pointer to the memory block is returned.
- If the function failed to allocate the requested block of memory, a null pointer is returned.

### Example


### Another example:

```c
#include <stdlib.h>
int *ptr = malloc( sizeof( int ) );
set ptr to point to a memory address of size int
int *ptr = malloc( sizeof(*ptr) );
is slightly cleaner to write malloc statements by taking the size of the variable pointed to by using the pointer directly
float *ptr = malloc( sizeof(*ptr) );
float *ptr;
/* hundreds of lines of code */
ptr = malloc( sizeof(*ptr) );
```
FYI... the exit function

Syntax:

```c
#include <stdlib.h>
void exit( int exit_code );
```

Description:

The `exit()` function stops the program. `exit_code` is passed on to be the return value of the program, where usually zero indicates success and non-zero indicates an error.

Example:

Static and Dynamic Arrays

- Static arrays are used when we know the amount of bytes in array at **compile time**.
  - Static arrays are ones that reside on the stack
  - `char arr[10];`
- A dynamic array is used where we come to know about the size on **run time**.
  - Dynamic arrays is a popular name given to a series of bytes allocated on the heap.
  - `char *ptr = (char*) malloc(10);`
  - allocates a memory of 10 bytes on heap and we have taken the starting address of this series of bytes in a character pointer `ptr`.
  - Fine if know number of characters, but what if don’t?
    - Read in one char/byte at a time until the user presses the enter key
- **malloc** (memory allocation) is used to dynamically allocate memory at run time. Possible uses for this function are:
  - Read records of an unknown length.
  - Read an unknown number of database records.
  - Link lists.
Usage of `calloc()`:

`void * calloc ( size_t num, size_t size );`

Parameters:

- Number of elements (array) to allocate and the size of elements.

Return value:

- Will return a pointer to the memory block. If the request fails, a NULL pointer is returned.

Example:

- `note: ptr_data = (int*) calloc ( a,sizeof(int) );`
The free function returns memory to the operating system.

```c
free(ptr);
```

After freeing a pointer, it is a good idea to reset it to point to 0.

**NOTE:** When 0 is assigned to a pointer, the pointer becomes a null pointer...in other words, it points to nothing. By doing this, when you do something foolish with the pointer (it happens a lot, even with experienced programmers), you find out immediately instead of later, when you have done considerable damage.