Useful information on some functions in C language. You may also refer to this site for more information: http://www.cplusplus.com/reference/clibrary/

`sizeof()` returns the number of bytes allocated for a buffer, variable, or data type. For example, `sizeof("hello")` returns 6 (5 + string terminator character).

`strlen()` returns the length (number of characters not including the terminating null-character) of the string. For example, `strlen("hello")` returns 5. Do not confuse this with the size of the array that holds the string.

`strcpy(dst,src)` copies the string pointed by `src` into the array pointed by `dst`, including the terminating null character (and stopping at that point). To avoid overflows, the size of the array pointed by `dst` shall be long enough to contain the same string as `src` (including the terminating null character), and should not overlap in memory with `src`.

`strncpy(dst,src,num)` copies the first *num* characters of `src` to `dst`. If the end of the `src` string (which is signaled by a null-character) is found before *num* characters have been copied, `dst` is padded with zeros until a total of `num` characters have been written to it. No null-character is implicitly appended at the end of `dst` if `src` is longer than *num*. Thus, in this case, `dst` shall not be considered a null terminated string.

`strcat(dst,src)` Appends a copy of the `src` string to the `dst` string. The terminating null character in `dst` is overwritten by the first character of `src`, and a null-character is included at the end of the new string formed by the concatenation of both in `dst`.

`strncat(dst,src,num)` Appends the first *num* characters of `src` to `dst`, plus a terminating null-character. If the length of the string in `src` is less than *num*, only the content up to the terminating null-character is copied. The `dst` must be large enough to contain the concatenated resulting string, including the additional null-character.

`malloc(size)` allocates a block of `size` bytes of memory and returns a pointer to the beginning of the block. If the function failed to allocate the requested block of memory, a NULL pointer is returned.

`free(ptr)` deallocates or frees a block of memory previously allocated by a call to `malloc()`, `calloc()`, or `realloc()` functions, making it available again for further allocations. This function does not change the value of pointer `ptr` itself, hence it still points to the same (now invalidated) location.