- `char * strchr(const char *s, int c);`
  The `strchr` function locates the first occurrence of `c` (converted to a char) in the string pointed to by `s`. It returns a pointer to the located character, or `NULL` if the character does not appear in the string.

- `char * strdup(const char *s1);`
  The `strdup` function allocates sufficient memory for a copy of the string `s1`, does the copy, and returns a pointer to it. The pointer may subsequently be used as an argument to the function `free`. If insufficient memory is available, `NULL` is returned.

- `size_t strlen(const char *s);`
  The `strlen` function computes the length of the string `s` and returns the number of characters that precede the terminating `\0` character.

- `int strcmp(const char *s1, const char *s2);`
  `int strncmp(const char *s1, const char *s2, size_t n);`
  The `strcmp` and `strncmp` functions lexicographically compare the null-terminated strings `s1` and `s2`. The `strncmp` function compares not more than `n` characters. Because `strncmp` is designed for comparing strings rather than binary data, characters that appear after a `\0` character are not compared. These functions return an integer greater than, equal to, or less than 0, according as the string `s1` is greater than, equal to, or less than the string `s2`. The comparison is done using unsigned characters.

- `char *strstr(const char *s1, const char *s2);`
  The `strstr` function locates the first occurrence of the null-terminated string `s2` in the null-terminated string `s1`. If `s2` is an empty string, `s1` is returned; if `s2` occurs nowhere in `s1`, `NULL` is returned; otherwise a pointer to the first character of the first occurrence of `s2` is returned.

- `void * malloc(size_t size);`
  The `malloc` function allocates `size` bytes of memory and returns a pointer to the allocated memory.

- `void * realloc(void *ptr, size_t size);`
  The `realloc` function creates a new allocation of `size` bytes, copies as much of the old data pointed to by `ptr` as will fit to the new allocation, `free`s the old allocation, and returns a pointer to the allocated memory. If `ptr` is `NULL`, `realloc` is identical to a call to `malloc` for `size` bytes.

- `void free(void *ptr);`
  The `free` function deallocates the memory allocation pointed to by `ptr`. If `ptr` is a `NULL` pointer, no operation is performed.

- `int fgetc(FILE *stream);`
  The `fgetc` function obtains the next input character (if present) from the stream pointed at by `stream`, or `EOF` if `stream` is at end-of-file.

- `char * fgets(char *  str, int size, FILE *  stream);`
  The `fgets` function reads at most one less than the number of characters specified by `size` from the given `stream` and stores them in the string `str`. Reading stops when a newline character is found, at end-of-file or error. The newline, if any, is retained. If any characters are read and there is no error, a `\0` character is appended to end the string. Upon successful completion, it returns a pointer to the string. If end-of-file occurs before any characters are read, it returns `NULL`.

- `ssize_t getline(char ** linep, size_t * linecapp, FILE * stream);`
  The getdelim() function , delimited by the character delimiter. The `getline` function reads a line from stream, which is ended by a newline character or end-of-file. If a newline character is read, it is included in the string. The caller may provide a pointer to a `malloc`ed buffer for the line in `*linep`, and the capacity of that buffer in `*linecapp`. These functions expand the buffer as needed, as if via `realloc`. If `linep` points to a `NULL` pointer, a new buffer will be allocated. In either case, `*linep` and `*linecapp` will be updated accordingly. This function returns the number of characters written to the string, excluding the terminating `\0` character. The value -1 is returned if an error occurs, or if end-of-file is reached.

- `void * memcpy(void * dst, const void *src, size_t n);`
  The `memcpy` function copies `n` bytes from memory area `src` to memory area `dst`.