

CS107 Exam C Reference Sheet

C Strings

`size_t strlen(const char *str);`
returns the length of `str`

`int strcmp(const char *s, const char *t);`
Compares `s` to `t`. Returns `<0` if `s < t`, `0` if `s == t`, `>0` if `s > t`

`int strncmp(const char *s, const char *t, size_t n);`
Compares `s` to `t` up to `n` characters. Returns `<0`, `0`, or `>0` as `strcmp` does

`char *strchr(const char *s, int ch);`
Returns a pointer to the first occurrence of `ch` in `s`, or `NULL` if not found

`char *strstr(const char *haystack, const char *needle);`
Returns a pointer to the first occurrence of `needle` in `haystack`, or `NULL` if not found

`char *strcpy(char *dst, const char *src);`
Copies `src` string to `dst` string. Strings must be properly null-terminated. Caller is responsible for assuring there is space for `src` in `dst`. Returns `dst`

`char *strncpy(char *dst, const char *src, size_t n);`
Similar to `strcpy`, but only copies at most `n` bytes. If there is no null byte in the first `n` bytes of `src`, the string placed into `dst` will not be null-terminated.

`char *strcat(char *dst, const char *src);`
Appends `src` to the end of `dst`, overwriting the original null-terminator in `dst`, and adding a null-terminator to the end of the resulting string. Caller is responsible for assuring there is space in `dst` for the appended string.

`char *strncat(char *dst, const char *src, size_t n);`
Similar to `strcat`, but only uses at most `n` bytes from `src`. As with `strcat()`, the resulting string in `dst` is always null-terminated. If `src` contains `n` or more bytes, `strncat()` writes `n+1` bytes to `dst` (`n` from `src` plus the terminating null byte). Therefore, the size of `dst` must be at least `strlen(dst)+n+1`.

`size_t strspn(const char *s, const char *accept);`
calculates the length (in bytes) of the initial segment of `s` which consists entirely of bytes in `accept`.

`size_t strcspn(const char *s, const char *reject);`
calculates the length of the initial segment of `s` which consists entirely of bytes not in `reject`.

`char *strdup(const char *s);`
returns a pointer to a new string which is a duplicate of the string `s`.
Memory for the new string is obtained with `malloc`, and should be freed with `free`.

`char *strndup(const char *s, size_t n);`
similar to `strdup`, but copies at most `n` bytes. If `s` is longer than `n`, only `n` bytes are copied, and a terminating null byte (`'\0'`) is added.
Memory for the new string is obtained with `malloc`, and should be freed with `free`.

`int atoi(const char *s);`
converts the initial portion of the string pointed to by `s` to `int`

`long strtol(const char *s, char **endptr, int base);`
converts the initial part of the string in `s` to a long integer value according to the given base. If `endptr` is not `NULL`, `strtol()` stores the address of the first invalid character in `*endptr`.

Memory

`void *malloc(size_t sz);`
allocates `size` bytes and returns a pointer to the allocated memory.
The memory is not initialized.

`void *calloc(size_t nmemb, size_t sz);`
allocates memory for an array of `nmemb` elements of `size` bytes each and returns a pointer to the allocated memory. The memory is set to zero.

`void *realloc(void *ptr, size_t sz);`
changes the size of the memory block pointed to by `ptr` to `size` bytes. The contents will be unchanged in the range from the start of the region up to the minimum of the old and new sizes. If the new size is larger than the old size, the added memory will not be initialized.

`void free(void *ptr);`
frees the memory space pointed to by `ptr`, which must have been returned by a previous call to `malloc()`, `calloc()`, or `realloc()`.

`void *memcpy(void *dst, const void *src, size_t n);`
copies `n` bytes from memory area `src` to memory area `dst`.
The memory areas must not overlap. Returns `dst`.

`void *memmove(void *dst, const void *src, size_t n);`
copies `n` bytes from memory area `src` to memory area `dst`. Returns `dst`.

`void *memset(void *s, int c, size_t n);`
fills the first `n` bytes of the memory area pointed to by `s` with the constant byte `c`.

Search and sort

```
void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
```

The `qsort()` function sorts an array with `nmemb` elements of size `size`. The `base` argument points to the start of the array. The contents of the array are sorted in ascending order according to a comparison function referenced by `compar`.

```
void *bsearch(const void *key, const void *base, size_t nmemb, size_t size,
              int (*compar)(const void *, const void *));
```

The `bsearch()` function searches an array of `nmemb` objects, the initial member of which is pointed to by `base`, for a member that matches the object pointed to by `key`. The size of each member of the array is specified by `size`. The contents of the array should be in ascending sorted order according to the comparison function referenced by `compar`.

I/O

```
char *fgets(char buf[], int buflen, FILE *fp);
```

`fgets()` reads in at most `buflen - 1` characters from `fp` and stores them into the buffer addressed by `buf`. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A terminating null byte ('\0') is stored after the last character in the buffer. `fgets()` returns `buf` on success, and `NULL` on error or when end of file occurs while no characters have been read.

Defined Constants

```
#define CHAR_MIN -128
#define CHAR_MAX 127
#define UCHAR_MAX 255

#define SHRT_MIN -32768
#define SHRT_MAX 32767
#define USHRT_MAX 65535

#define INT_MIN -2147483648
#define INT_MAX 2147483647
#define UINT_MAX 4294967295

#define LONG_MIN -9223372036854775808
#define LONG_MAX 9223372036854775807
#define ULONG_MAX 18446744073709551615
```

Decimal / Hex / Binary values, 0-15

decimal	hex	binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	a	1010
11	b	1011
12	c	1100
13	d	1101
14	e	1110
15	f	1111