CS107 Reference

C Strings

```
size_t strlen(const char *str);
int    strcmp(const char *s, const char *t);
int    strncmp(const char *s, const char *t, size_t n);
char  *strchr(const char *s, int ch);
char  *strstr(const char *haystack, const char *needle);
char  *strcpy(char *dst, const char *src);
char  *strncpy(char *dst, const char *src, size_t n);
char  *strcat(char *dst, const char *src);
char  *strncat(char *dst, const char *src, size_t n);
size_t strspn(const char *s, const char *accept);
size_t strcspn(const char *s, const char *reject);
char  *strdup(const char *s);
int atoi(const char *s);
long strtol(const char *s, char **endptr, int base);
```

Memory

```
void *malloc(size_t sz);
void *calloc(size_t nmemb, size_t sz);
void *realloc(void *ptr, size_t sz);
void  free(void *ptr);
void *memcpy(void *dst, const void *src, size_t n);
void *memmove(void *dst, const void *src, size_t n);
void *memset(void *base, int byte, size_t n);
```

Search and Sort

```
void  qsort(void *base, size_t nelems, size_t width,
            int (*compar)(const void *, const void *));
void *bsearch(const void *key, const void *base, size_t nelems, size_t width,
              int (*compar)(const void *, const void *));
void *lfind(const void *key, const void *base, size_t *p_nelems, size_t width,
            int(*compar)(const void *, const void *));
void *lsearch(const void *key, void *base, size_t *p_nelems, size_t width,
              int(*compar)(const void *, const void *));
```

I/O

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

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
```

CS107 x86-64 Reference Sheet

Common instructions

mov	src, dst	dst = src
movsbl	src, dst	byte to int, sign-extend
movzbl	src, dst	byte to int, zero-fill
cmov	src, reg	reg = src when condition holds, using same condition suffixes as jmp
lea	addr, dst	dst = addr
add	src, dst	dst += src
sub	src, dst	dst -= src
imul	src, dst	dst *= src
neg	dst	dst = -dst (arith inverse)
imulq	S	signed full multiply R[%rdx]:R[%rax] <- S * R[%rax]
mulq	S	unsigned full multiply same effect as **imulq**
idivq	S	signed divide R[%rdx] <- R[%rdx]:R[%rax] mod S R[%rax] <- R[%rdx]:R[%rax] / S
divq	S	unsigned divide - same effect as **idivq**
cqto		R[%rdx]:R[%rax] <- SignExtend(R[%rax])
sal	count, dst	dst <<= count
sar	count, dst	dst >>= count (arith shift)
shr	count, dst	dst >>= count (logical shift)
and	src, dst	dst &= src
or	src, dst	dst |= src
xor	src, dst	dst ^= src
not	dst	dst = ~dst (bitwise inverse)
cmp	a, b	b-a, set flags
test	a, b	a&b, set flags
set	dst	sets byte at dst to 1 when condition holds, 0 otherwise, using same condition suffixes as jmp
jmp	label	jump to label (unconditional)
je	label	jump equal ZF=1
jne	label	jump not equal ZF=0
js	label	jump negative SF=1
jns	label	jump not negative SF=0
jg	label	jump > (signed) ZF=0 and SF=OF
jge	label	jump >= (signed) SF=OF
jl	label	jump < (signed) SF!=OF
jle	label	jump <= (signed) ZF=1 or SF!=OF
ja	label	jump > (unsigned) CF=0 and ZF=0
jae	label	jump >= (unsigned) CF=0
jb	label	jump < (unsigned) CF=1
jbe	label	jump <= (unsigned) CF=1 or ZF=1
push	src	add to top of stack Mem[--%rsp] = src
pop	dst	remove top from stack dst = Mem[%rsp++]
call	fn	push %rip, jmp to fn
ret		pop %rip

Condition codes/flags

ZF Zero flag
SF Sign flag
CF Carry flag
OF Overflow flag

Addressing modes

Example source operands to **mov**

Immediate

```
mov $0x5, dst
```

$val
source is constant value

Register

```
mov %rax, dst
```

%R
R is register
source in %R register

Direct

```
mov 0x4033d0, dst
```

0xaddr
source read from Mem[0xaddr]

Indirect

```
mov (%rax), dst
```

(%R)
R is register
source read from Mem[%R]

Indirect displacement

```
mov 8(%rax), dst
```

D(%R)
R is register
D is displacement
source read from Mem[%R + D]

Indirect scaled-index

```
mov 8(%rsp, %rcx, 4), dst
```

D(%RB,%RI,S)
RB is register for base
RI is register for index (0 if empty)
D is displacement (0 if empty)
S is scale 1, 2, 4 or 8 (1 if empty)
source read from:
Mem[%RB + D + S*%RI]

CS107 x86-64 Reference Sheet

Registers

%rip	Instruction pointer
%rsp	Stack pointer
%rax	Return value
%rdi	1st argument
%rsi	2nd argument
%rdx	3rd argument
%rcx	4th argument
%r8	5th argument
%r9	6th argument
%r10,%r11	Callee-owned
%rbx,%rbp, %r12-%15	Caller-owned

Instruction suffixes

b	byte
w	word (2 bytes)
l	long /doubleword (4 bytes)
q	quadword (8 bytes)

Suffix is elided when can be inferred from operands. e.g. operand `%rax` implies `q`, `%eax` implies `l`, and so on

Register Names

64-bit register	32-bit sub-register	16-bit sub-register	8-bit sub-register
%rax	%eax	%ax	%al
%rbx	%ebx	%bx	%bl
%rcx	%ecx	%cx	%cl
%rdx	%edx	%dx	%dl
%rsi	%esi	%si	%sil
%rdi	%edi	%di	%dil
%rbp	%ebp	%bp	%bpl
%rsp	%esp	%sp	%spl
%r8	%r8d	%r8w	%r8b
%r9	%r9d	%r9w	%r9b
%r10	%r10d	%r10w	%r10b
%r11	%r11d	%r11w	%r11b
%r12	%r12d	%r12w	%r12b
%r13	%r13d	%r13w	%r13b
%r14	%r14d	%r14w	%r14b
%r15	%r15d	%r15w	%r15b