CS107 Autumn 2000

Handout 05 September 29, 2000

C Cheat Sheet

Compiled by Nick Parlante.

Precedence		Associativity
`function-call() [] -> .`		L to R
`& ! ~ ++ -- + - *(ptr) sizeof`	(remember:all unary ops are the same)	**R to L**
`* / %`	(binary arithmetic ops are next)	L to R
`+ -` L to R		
`< <= > >=`		L to R
`== !=`		L to R
and then in order: `& ^ | && ||`		L to R
`=` and all its variants `+=, -=,` etc.		**R to L**
`,` (The lowest precedence. An operator only a nerd could love.)		L to R

A combination which never works right without parens: `*structptr.field`

stdio.h

`FILE *fopen(const char *fname, const char*mode);` "r"read,"w" write,"a"append, NULL on err.
`int fclose(FILE *file);` Returns EOF on error.
`int fgetc(FILE *in);` Returns next char, or EOF token if no more.
`int fputc(int ch, FILE *out);` Writes char to file, returns EOF on err.
`int ungetc(int ch, FILE *in);` Push one fgetc char back, may not ungetc EOF, EOF on err.

printf

`%d` int, `%s` char*, `%f` double, `%e` double scientific notation.
standard already-open FILE*'s: `stdin, stdout, stderr`
printf goes to stdout by default, use `fprintf(FILE *out,...)` with FILE *param to send elsewhere.

ctype.h

Macros for testing the type of a character: `isalpha(ch)` alphabetic upper or lower, `islower(ch), isupper(), isspace(ch)` tab space newline, `isdigit(ch), ispunct(ch)`
`tolower(ch), toupper(ch)` These conversions work on any char, no need to test first.

string.h

None of the string routines allocate memory, the client is responsible for making sure there is enough space.
Most of these return the string just written which is often not all that useful.
`size_t strlen(const char* string);` return num chars, doesn't count null. `strlen("abc") --> 3`
`char *strcpy(char *dest, const char* source);` Copy a string. Remember: copies R to L like assign.
`char *strncpy(char *dest, const char* source, int n);` Copy at most n chars.
`char *strcat(char *dest, const char* source);` Append src to end of dest.
`int strcmp(const char *a, const char *b);` Return neg. if a<b, pos. if a>b, 0 if a==b. Is case-sensitve.
`char *strchr(const char* str, char ch);` ret pointer to first occurence of ch in str, NULL if none.

`void* memcpy(void *dest, const void *source, size_t n)` Copy non-overlapping bytes.
`void* memmove(void *dest, const void *source, size_t n)` Works even if bytes overlap.
These two are likely to be very efficiently implemented on whatever machine you are on.

stdlib.h

`int rand(void);` returns pseudo random numbers (non-negative).
`void srand(unsigned int seed);` Set seed for rand. Often use `time(NULL)` from `<time.h>` as seed.
`void *malloc(size_t size);` allocate heap block, returns NULL on fail. `size_t` is an `unsigned long.`
`void *realloc(void *block, size_t size);` resize block to new size, returns *possibly changed ptr* to use.
`void free(void *block);` Return a malloc or realloc block to the heap.
`void exit(int status);` Halt the program. Pass EXIT_SUCCESS or EXIT_FAILURE.

`void *bsearch(const void *key, const void *base, size_t n, size_t elem_size, cmpfn below)`
returns address of found element, or NULL if not found
`void qsort(void *base, size_t n, size_t elem_size, cmpfn below);`
var name conventions: "len" or "n" = number of elements in array, "size" = number of bytes
Comparator has return value like strcmp: `int cmp(const void *key, const void *x);`