

C Reference Card

2004-06-21

substitutable parameters shown in *italics*

Compilation

```
gcc -flags program.c
dcc -flags program.c (CSE labs only)
-c          Compile only, output to file.o
-o file     Executable output to file
-g         Generate debugging info for gdb
-Wall      Turn on all warnings
```

Lexical structure, preprocessor

```
/* comment */
// comment to end of line
#include <libmodule.h>
#include "usermodule.h"
#define NAME replacement-text
#define NAME(args...) replacement-text
```

Program structure:

Header files: declarations only (#includes, #defines, function prototypes)

Implementation files: #includes, #defines, prototypes for local functions, function definitions

Main program file: as for implementation, must have main:

```
int main(int argc, char **argv)
```

Identifiers start with a letter, followed by any number of letters, digits or underscores

Identifiers starting with `_` reserved for system use

Reserved words (can't use as identifiers):

```
auto break case char const continue default
do double else entry enum extern float for
goto if int long register return short
signed sizeof static struct switch typedef
union unsigned void volatile while
```

Literals (examples)

```
123 -4 0xAf0C 057 integers (int)
3.14159265 1.29e-23 reals (double)
'x' '\t' '\033' characters (char)
"hello" "abc\\n" "" strings (char *)
```

Character and string escapes

symbol	represents	symbol	represents
<code>\t</code>	tab	<code>\ddd</code>	ASCII value (octal)
<code>\n</code>	newline	<code>\'</code>	single quote
<code>\r</code>	carriage-return	<code>\"</code>	double quote
<code>\0</code>	null character	<code>\\</code>	backslash

Declarations (examples)

```
int i, length;
char *str, buf[BUFSIZ], prev;
double x, values[MAX];
typedef enum { FALSE, TRUE } Bool;
typedef struct {
    char *key;
    int val;
} KeyValType;
type funcname (type param1, type param2 ...);
```

More types

```
short (int) long (int, double)
unsigned (int, char)
```

Storage classes (common)

```
static local to file, or var saved across function calls
extern accessible everywhere
```

Initialisation (examples)

```
int c = 0;
char prev = '\n';
char *mssg = "hello";
int seq[MAX] = { 1, 2, 3 };
KeyValType keylist[] = {
    "NSW", 0, "Vic", 5, "Qld", -1 };
```

Operators (decreasing precedence down and across)

<code>() [] . -></code>	Brackets, array, struct, pointer-struct
<code>++ -- - ! *</code> <code>& ~ sizeof</code> <i>(typename)</i>	Incr/decrement, unary minus, logical NOT, pointer deref., address-of, 1's complement, size in bytes, cast ♦
<code>* / %</code> <code>+ -</code>	Binary arithmetic operators
<code><< >></code>	Bitwise left shift/right shift
<code>< <= > >=</code>	Relational operators
<code>== !=</code> <code>&</code>	(In)equality operators; bitwise AND
<code>^</code>	Bitwise exclusive OR, inclusive OR
<code>&& ?:</code>	Logical AND and OR; conditional ♦
<code>= += -= *=</code> <code>/= %= etc</code>	Assignment (with optional arithmetic operation) ♦
<code>,</code>	Comma (sequential) operator

Left-associative except for ♦ (right associative)

Statements

```
expression ;
{ statements... }
if (expression) statement
if (expression) statement else statement
switch (expression) {
    case constant : statements... break;
    case constant : statements... break;
    default : statements
}
while (expression) statement
for (initialiser; condition; increment) statement
do statement while (expression);
break;          terminate loop or switch
continue;       resume next iteration of loop
return expr;    return value from function
goto identifier; transfer to label (rare)
```

C library functions (and other objects)

Parameter name implies type: **c** char
n int **l** long **s** string(char *)
b buffer (char array) **p** pointer (void *)
d double **fh** file handle (FILE *)

stdlib.h

atoi(s)	atof(s)	string to int or double
malloc(n)	calloc(n)	allocate n bytes
free(p)		recycle memory
exit(n)		terminate with status n
abs(n)	labs(l)	absolute value

stdio.h

stdin	stdout	stderr	FILE * variables
BUFSIZ	EOF	NULL	constants

fopen(s, mode) open file, returns **fh**
mode is one or more of "r", "w", "a" "b" "+"
fclose(fh) close file
fgetc(fh) getchar() read char, **EOF** if none
fgets(b, n, fh) read line, **NULL** if none
fputc(c, fh) putchar(c) write char
fputs(s, fh) write line
fread(p, size, nel, fh) read into binary buffer,
return number of elements read
fwrite(p, size, nel, fh) write from binary buffer

Formatted output:

fprintf(fh, format, list)	formatted output to fh
printf(format, list)	fmt output to stdout
sprintf(b, format, list)	formatted output to string

format items **%width.precision code**

negative *width* left-justifies. *code* is one of

d decimal	o octal	x hexadecimal
f fixed point	g general	e exponential (scientific)
c character	s string	p pointer
% literal '%' character		

Formatted input:

fscanf(fh, format, list)	formatted input from fh
scanf(format, list)	fmt input from stdin
sscanf(s, format, list)	formatted input from string

format codes similar to printf, *list* has addresses

ctype.h

toupper(c)	tolower(c)	case mapping
isupper(c)	islower(c)	case testing
isalpha(c)	isalnum(c)	alpha(betic numeric)
isdigit(c)	isxdigit(c)	decimal or hex digit
isspace(c)	isprint(c)	white space, printable

string.h

strlen(s)	length (excluding '\0')
strcpy(sd, ss)	copy ss to sd , return sd
strcat(sd, ss)	append ss to sd , return sd
strcmp(s1, s2)	compare, return <0 ==0 >0
strncpy(sd, ss, n)	strncat(sd, ss, n)
strncmp(s1, s2, n)	max n chars processed
strchr(s, c)	return ptr to first c in s
strrchr(s, c)	return ptr to last c in s
strstr(s, sp)	return ptr to first sp in s
strpbrk(s, set)	return ptr to first of any in set
strspn(s, set)	length of prefix of any in set
strcspn(s, set)	length of prefix all <i>not</i> in set

math.h (all parameters are double)

sin(d)	cos(d)	tan(d)	trigonometry (radians)
asin(d)	acos(d)	atan(d)	inverse (radians)
	atan2(y, x)		= tan ⁻¹ (y/x)
sinh(d)	cosh(d)	tanh(d)	hyperbolic
exp(d)	log(d)	log10(d)	exponential, logarithm
pow(x, y)	sqrt(d)		x ^y , square root
floor(d)	ceil(d)		integral bounds
fabs(d)	fmod(x, y)		absolute value, x % y

Common programming patterns

Read input a character at a time:

```
int c;
while ((c = getchar()) != EOF) {
    putchar(c); // or some other use of c
}
```

Read input a line at a time:

```
char buf[BUFSIZ];
while(fgets(buf, BUFSIZ, stdin) != NULL)
    process_line(buf);
```

Opening a file named on the command line:

```
FILE *fp;

fp = fopen(argv[1], "r");
if (fp == NULL) {
    fprintf(stderr, "can't open %s\n",
            argv[1]);
    exit(1);
}
```

Same, but file name is optional, default standard input:

```
if (argc == 1)
    process(stdin);
else {
    // open fp as above
    process(fp);
}
```

Print array of real numbers:

```
for (i=0; i < MAX; i++)
    printf("%14.6f\n", sample[i]);
```

Function to find the length of a string:

```
int mystrlen(char *s)
{
    int len = 0;
    char *sp;

    for (sp=s; *sp != '\0'; sp++)
        len++;
    return len;
}
```