

C Quick Reference

Data types

type	value	printf/scanf
int	42	%d
char	'd'	%c
float	4.3, 5.6e7	%f
double	4.3, 5.6e7	%lf
char *	"a string"	%s

Pointer types

Declare a pointer to int with

```
int *myptr;
```

Set myptr to point to a with

```
myptr = &a;
```

and use it with

```
*myptr = 42; // same as a = 42
```

Operators and relations

Highest precedence first, bars separate precedences

++	--	increment, decrement	a++
()		function call	mean(5.6,44)
[]		array subscript	array[7]
.		struct field	p.x
->		struct field via ptr	p_ptr->x
-		unary minus	-a
!		not	!b
(type)		cast	(double)4.3
*		dereference	*p_ptr
&		address of	&p
* / %		multn, divn, mod	a * 4.2
+ -		addn, subtracn	a+3
< <= > >=		gtr/less than/or equal	a <= b
== !=		(not) equal	a + 3 == b/7
&&		logical and	a && b
		logical or	a b
=		assignment	a = b+3

Control flow

Assignment: a = b+3;

Conditionals:

```
if ( a > 3 ) {  
    b++;  
} else {  
    b--;  
}
```

Iteration:

```
while ( i < n ) {  
    ...  
}  
for ( i = 0; i < n; i++) {  
    ...  
}
```

break; terminates a for or while loop.

continue; skips the rest of the loop body this time round

Switch statement:

```
switch ( i ) {  
    case 0:  
        ...  
        break;  
    case 1: case 2:  
        ...  
        break;  
    default:  
        ...  
}
```

Functions

Prototype:

```
char nthcharof(char *, int);
```

Definition:

```
char nthcharof(char *str, int n) {  
    return str[n];  
}
```

Use:

```
char *mystr = "a boring string";  
char c;  
c = nthcharof(mystr,7);
```

Structs

Declaring a struct type with typedef:

```
typedef struct {  
    int n;  
    char *str;  
} mytype;
```

Using structs:

```
mytype x;  
x.n = 42;  
x.str = "forty-two";
```

Pointers and structs:

```
mytype *xptr = &x;  
xptr->n = 43;  
xptr->str = "forty-three";
```

Enums

typedef enum { FIRST, SECOND } num_t; Enums are ints; start from zero by default. Can do:

```
typedef enum { APPLE = 3, PEAR, ORANGE } fruit_t  
to start from 3, or
```

```
typedef enum { LEEK = 5, TOM = 8, CARROT = 10 } veg_t  
for arbitrary values
```

Strings

are null-terminated arrays of chars.

Useful functions (#include <string.h>):

```
char *s, *s1, *s2;
```

```
strlen(s) length of s, excluding final null
```

```
strcpy(s1,s2); copy contents of s2 into s1
```

```
strcmp(s1,s2); return -1, 0, 1 as s1 is <,,> s2 in  
lexicographic order
```

```
sprintf(s,...) print into s
```

```
scanf(s,...) read from s
```

Arrays

Declaring:

```
int myarray[10];
```

Using

```
myarray[i+1] = 2*myarray[i];
```

Arrays and pointers: `myarray[i]` is the same as `*(myarray+i)`, and `&(myarray[i])` is the same as `myarray+i`

Arrays and strings: after

```
char achar[] = { 'a', 'b', 'c', 'd', 0 };
```

```
char *mystr = achar;
```

`mystr` is the string "abcd", and `achar[i] == mystr[i]`

Basic i/o

Printing formatted strings:

```
int n; double f; char c; char *str;
printf("n is %d, f is %f, c is %c, str is %s\n",
       n,f,c,str);
```

Reading variables from input:

```
scanf("%d %lf %c %s",&n,&f,&c,str)
```

(N.B. no `&` for `%s`)

`scanf` skips white space before numbers or strings, and when there is a space in the format string.

Skipping a value: `"%*s"`

Useful `printf` format modifiers:

`%3d` pad with blanks on left to 3 columns

`%03d` pad with zeros on left to 3 columns

`%.3f` print to 3 decimal places

Character i/o:

```
int c; /* N.B. int NOT char */
```

```
c = getchar();
```

```
putchar(c);
```

File i/o

Input:

```
char *filename = "foo.txt";
```

```
FILE *infile;
```

```
infile = fopen(filename,"r");
```

```
fscanf(infile,"%d",&n);
```

```
char c = fgetc(infile);
```

Output:

```
FILE *outfile;
```

```
outfile = fopen(filename,"w");
```

```
fprintf(outfile,"n is %d\n",n);
```

```
fputc(c,outfile);
```

```
fclose(outfile);
```

Character identification

```
#include <ctype.h>
```

provides

```
int isalpha(int c); and similarly isdigit, isupper,
```

```
islower, isspace
```

```
and int toupper(int c); and similarly tolower.
```

Descartes quick reference

Types: `point_t`, `lineSeg_t`, `rectangle_t`.

General functions:

```
void OpenGraphics(void); Opens and initialises the graphics window
```

```
void Clear(void); Clears the entire graphics window to white
```

```
void CloseGraphics(void); Closes the graphics window
```

Functions about points:

```
point_t GetPoint(void); Waits until the user clicks the mouse, then returns the point that the user is indicating.
```

```
point_t Point(int a, int b); Creates a point with given coordinates.
```

```
int XCoord(point_t p); Returns the x-coordinate of the point given as argument.
```

```
int YCoord(point_t p); Returns the y-coordinate of the point given as argument.
```

Functions about lines:

```
lineSeg_t LineSeg(point_t p1, point_t p2); Creates a line segment with given endpoints.
```

```
point_t InitialPoint(lineSeg_t l); Returns one endpoint of a line segment ...
```

```
point_t FinalPoint(lineSeg_t l); ...returns the other endpoint.
```

```
double Length(lineSeg_t l); Returns the length of a line segment.
```

```
void DrawLineSeg(lineSeg_t l); Draws a line segment.
```

Functions about rectangles:

```
rectangle_t Rectangle(point_t bl, point_t tr); Creates a rectangle with the given bottom left and top right.
```

```
point_t BottomLeft(rectangle_t r); Extract the bottom left of a rectangle.
```

```
point_t TopRight(rectangle_t r); Extract the top right of a rectangle.
```

```
void FillRectangle(rectangle_t r); Fills a rectangle.
```

```
void ClearRectangle(rectangle_t r); Clears a rectangle to white.
```

Writing and compiling Descartes programs

The program must have the header line

```
#include "descartes.h"
```

To compile `myprog.c` that uses Descartes, do

```
gcc -Wall myprog.c descartes.o -lSDL -lm
```