Intro to C

Derrick

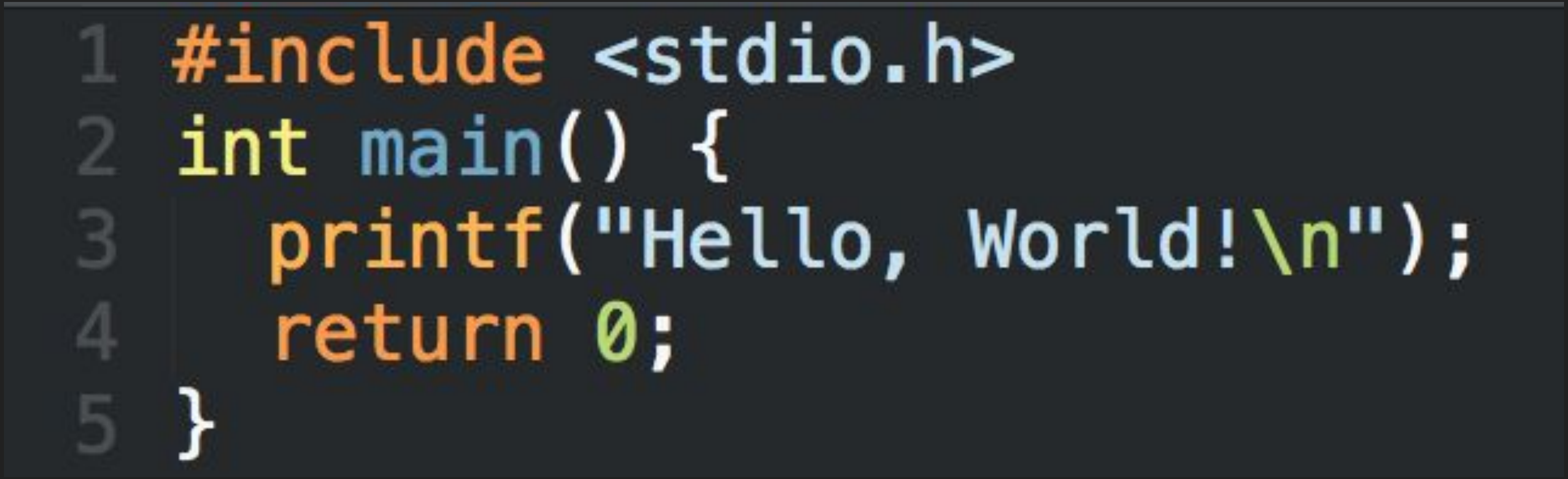
Course Objectives

- Be able to read and write C programs
- Understand “All” C language constructs
- Be able to use pointers
- Have a good overview of the Standard Library
- Be aware of some of C’s traps and pitfalls

Overview of C

- Developed by Brian Kernighan & Dennis Ritchie of AT&T Bell Labs in 1972
- C can be thought of as a “high level assembler”
- Designed for maximum processor speed
- THE System programming language
- Has a “write only” reputation

Simple C program



```c
#include <stdio.h>
int main() {
  printf("Hello, World!\n");
  return 0;
}
```

Sample Code

- Every full C program begins inside a function called “**main**”
- To access the standard functions that comes with your compiler, you need to include a header with **#include** directive.
- The #include is a “**preprocessor**” directive that tells the compiler to put code from the header called **stdio.h** into our program before actually create the executable. -- printf()
- The **printf** function is the standard C way of displaying output on the screen.

Header files

- The files that are specified in the include section is called as header file.
- These are precompiled files that has some functions defined in them.
- We can call those functions in our program by supplying parameters.
- Header file is given an extension .h
- C source file is given an extension .c

Main Functions

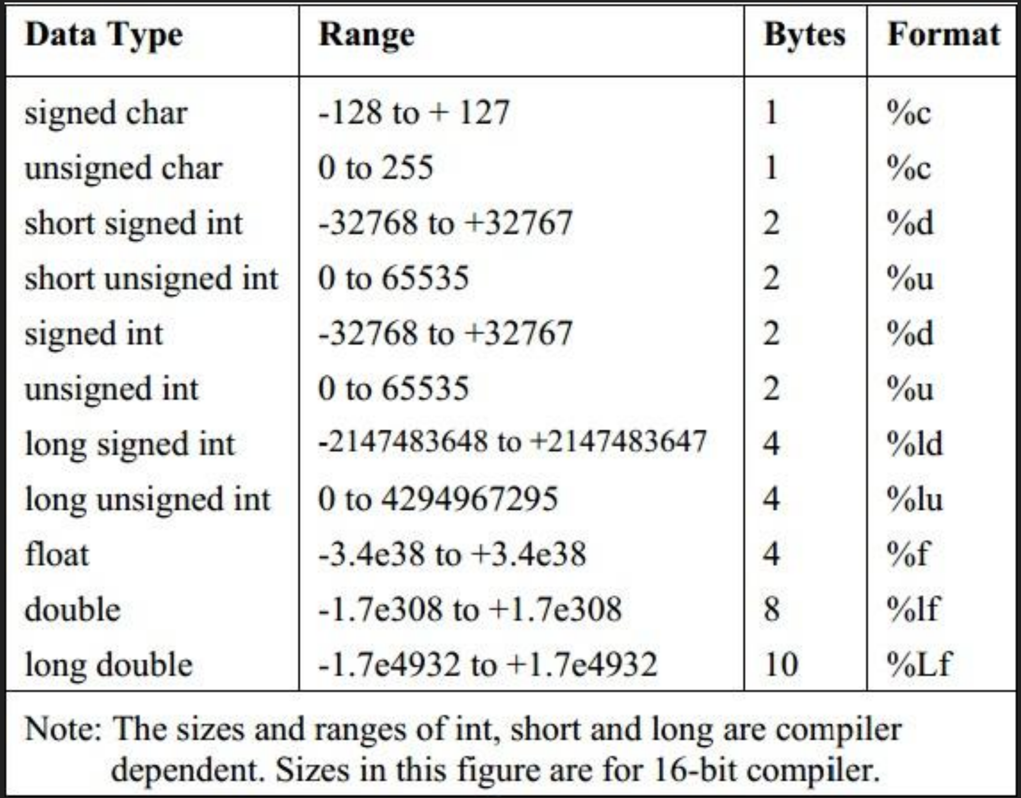
- This is the entry point of a program.
- When a file is executed, the start point is the main function.
- There may or may not be other functions written in a program.
- Main function is mandatory for any C program.

Running a C program

- Compile the program
  - `$ gcc hello.c -o Hello`

- Run the program
  - `$ ./Hello`

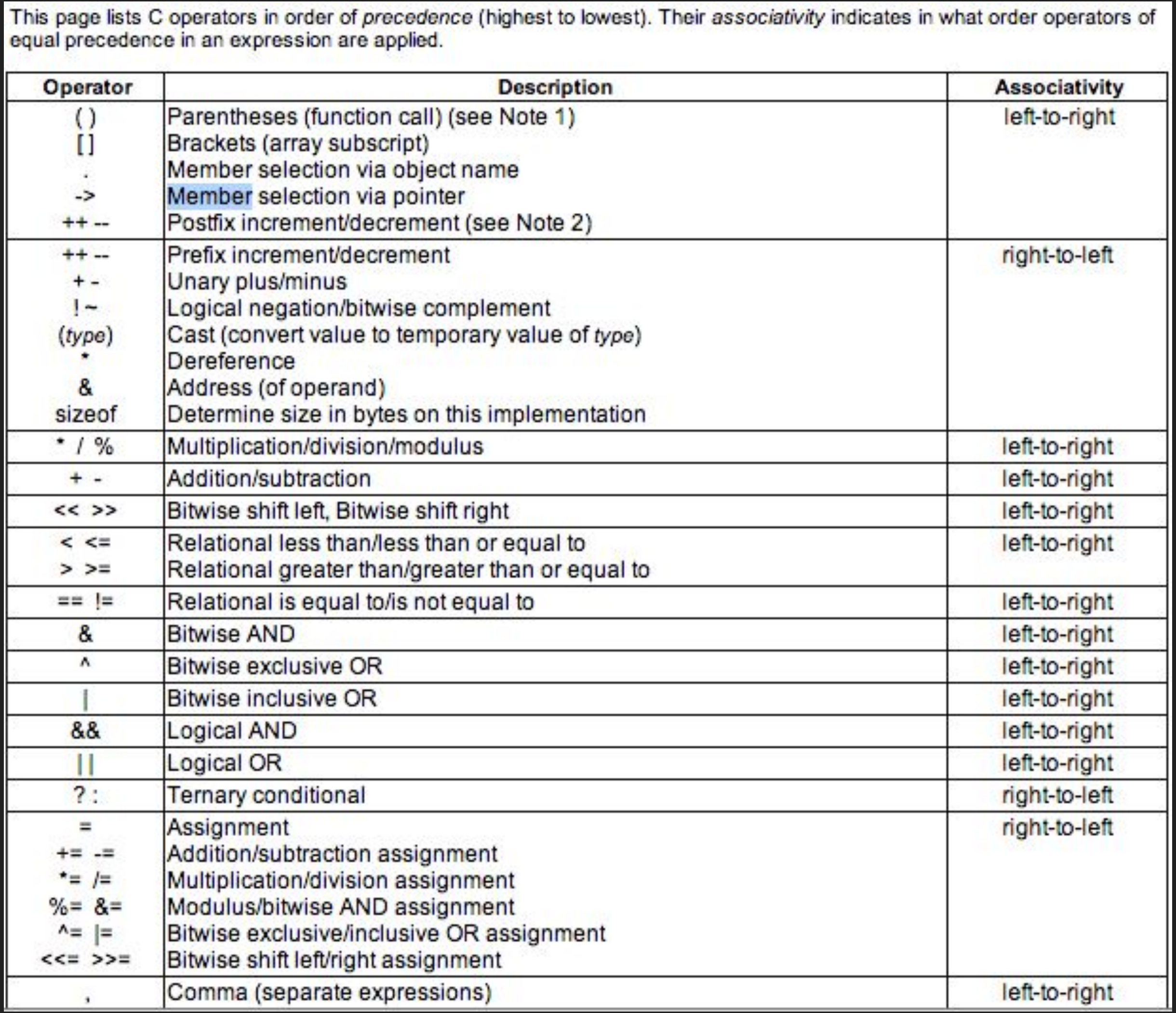
Data Types and Sizes



Data Type	Range	Bytes	Format
signed char	-128 to + 127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32768 to +32767	2	%d
short unsigned int	0 to 65535	2	%u
signed int	-32768 to +32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to +2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to +1.7e4932	10	%Lf

Note: The sizes and ranges of int, short and long are compiler dependent. Sizes in this figure are for 16-bit compiler.

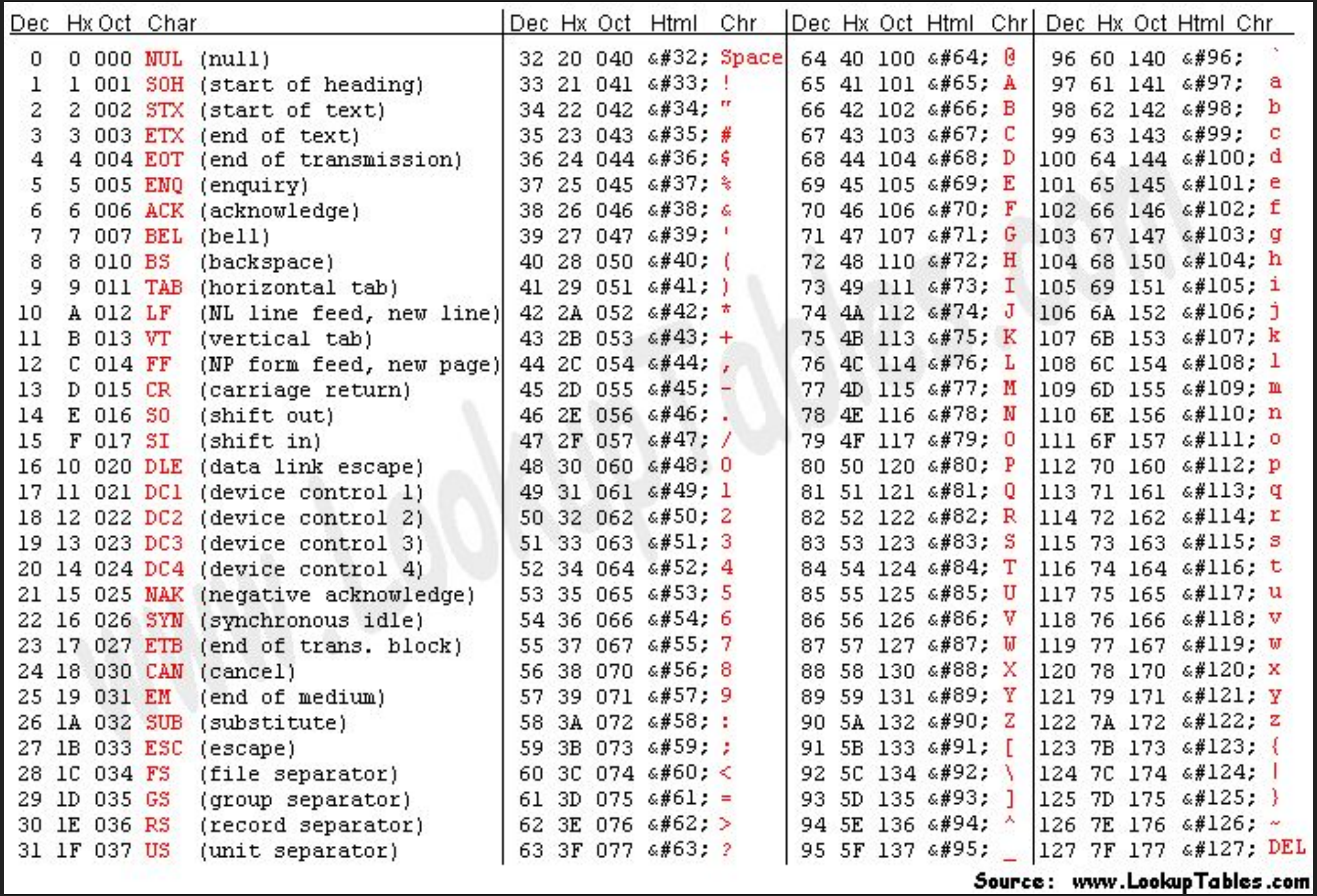
Operators



This page lists C operators in order of *precedence* (highest to lowest). Their *associativity* indicates in what order operators of equal precedence in an expression are applied.

Operator	Description	Associativity
( )	Parentheses (function call) (see Note 1)	left-to-right
[ ]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++ --	Postfix increment/decrement (see Note 2)	
++ --	Prefix increment/decrement	right-to-left
+ -	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(*type*)	Cast (convert value to temporary value of *type*)	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes on this implementation	
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= |=	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	
,	Comma (separate expressions)	left-to-right

ASCII Table



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Constants

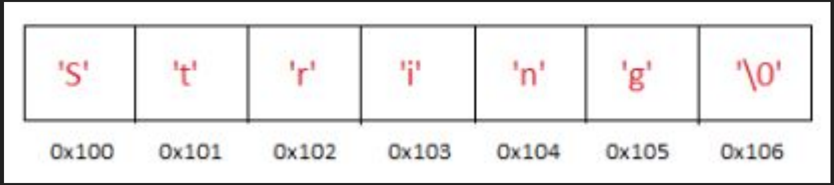
- long : `123456789l` or `123456789L`
- Unsigned: `123456789u` or `123456789U`
- Unsigned Long: `123456789ul` or `123456789UL`
- Float: `123.4f` or `123.4F`
- Long double: `123.4l` or `123.4L`
- A leading `0` (Octal base 8) and a leading `0x` (Hexadecimal base 16)
- A character constant: `‘x’` use single quote, mapped to an int value.
  - `‘0’ -> 48`
- Certain characters can be represented in character and string constants by escape sequences like `‘\n’` (represents one char)
  - `‘\t’ (tab), \”, \’, \\,` etc.

Symbolic Constant

- A ***#define*** directive (macro expansion) defines a **symbolic name** or **symbolic constant** to be a particular string of characters
- ***#define name replacement_text***
- Any occurrence of ***name*** will be replaced by the corresponding ***replacement_text***.
- The ***name*** has the same form as a variable name.
- The ***replacement_text*** can be any sequence of chars

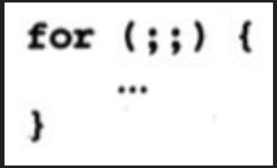
String Constant (Literal)

- A sequence of zero or more characters surrounded by double quotes.
  - `“I am a string”`
- Technically, a string constant is an array of chars with a null character `‘\0’` at the end.
  - The physical storage required is one more than the number of characters written between the quotes.
  - The standard library function **`strlen(s)`** returns the length of a string **s** excluding `‘\0’` `<string.h>`
  - `“X”` and `‘X’` are different.



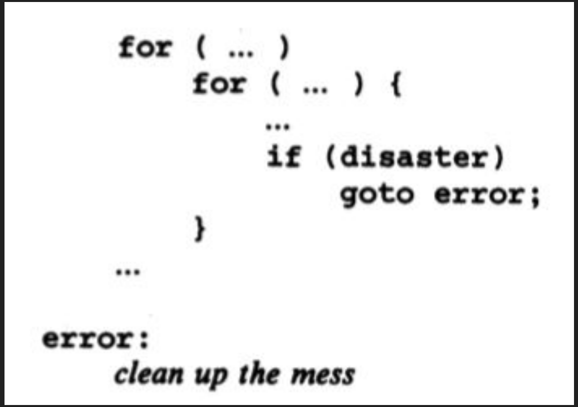
Control flows

- If - else, switch
- for loop, while loop, do-while
- All the same…



```
for (;;) {
     ...
}
```

- Goto and Labels (NO)

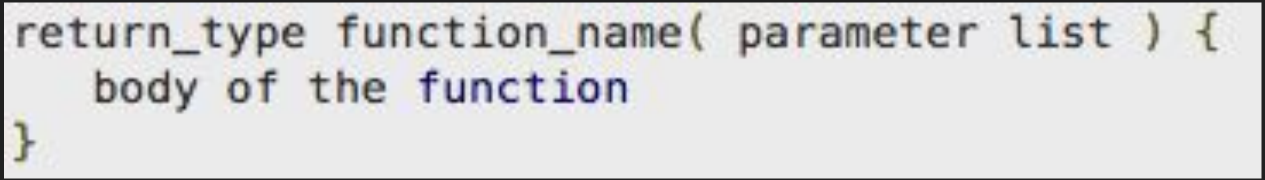


```
    for ( ... )
        for ( ... ) {
            ...
            if (disaster)
                goto error;
        }
    ...

error:
    clean up the mess
```

Functions

- A group of statements that together perform a task.
- Every C program has at least one function. (`main()`)



```
return_type function_name( parameter list ) {
   body of the function
}
```

putchar

- The C library function `int putchar(int char)` writes a character (`an unsigned char`) specified by the argument char to stdout.

-
  ```
  #include <stdio.h>

  int main () {
     char ch;

     for(ch = 'A' ; ch <= 'Z' ; ch++) {
        putchar(ch);
     }

     return 0;
  }
  ```

man

[RTFM]