terminal vars
code output

declarations : list variables, type, & initialize w/values
operators : what's done to the variables
expressions : combine variables & constants to produce new values

Variable Names

don't begin w/ underscores
there is case sensitivity

first 31 chars of name are significant

be smart

Data Types & Sizes

char	single byte, capable of holding 1 character
int	integer, typically reflecting size of ints on machine
float	single-precision floating part
double	double-" "

some modifiers : short or long
↓ ↓
16 bits 32 bits

short int counter

unsigned → positive or 0

unsigned char 8bits 0 → 255
signed char -128 → 127

Constants

long constant terminates w/ an l or L 123456789L
unsigned? → ul or UL

float constant terminates w/ f or F

octal leading zero on an integer constant
hexadecimal 0x or 0X

decimal 31 → octal: 031 hex: 0x1F or 0X1F

character constant is an int, written as 1 character w/ single quotes

'0' ~~ASCII~~ → 48

can use some escape sequences

\a	alert character	\\	backslash
\b	backspace	\?	question mark
\f	form feed	\'	single quotation mark
\n	new line	\"	double " "
\r	carriage return	\ooo	octal #
\t	horizontal tab	\xhh	hex #
\v	vertical tab		

'\0' → null

constant expressions - expression only using constants

```
#define MAXLINE 1000
char line[MAXLINE + 1]
```

string constant - sequence of 0 or more characters surrounded by " "

are concatenated @ compiling

defined as array of characters, terminated by '\0'

'x' → integer used to produce numeric value of letter x

"x" → array of chars containing x & '\0'

enumeration constant

enumeration - list of constant integer values

```
enum boolean { NO, YES }
```

if not all values are specified, continue from last specified values

```
enum months { Jan = 1, Feb, March } → Feb = 2 & March = 3
```

names must be distinct, values not necessarily

Declarations

type & list of vars of that type

const double e = 2.718

↳ won't change value

Arithmetic Operators

\+ - * / %

↳ truncates w/ ints

Relational and Logical Operators

> >= < <= == !=

&& → and || → or

read from left to right

relational expression → 0 or 1

! operator nonzero → 1 & zero → 1

if (! valid) ← preferred over if (valid == 0)

Type Conversions

only automatic conversion is 'narrow' to 'wide' w/ losing info.

long → short can cause warning, not illegal

char is just a small int, can be used for arithmetic expressions

<ctype.h> has type fn that are useful

Specify unsigned or signed when converting char → int

Order:

if there's a long double → other to long double

Otherwise if either is a double → other to double

" " float → " " float

" " convert char & short → int

if either is long → other to long

int i
char c

i = c → int i → char i
c = i → char c → char i value stays same

conversions occur when passing args into a f^n to what f^n expects

to cast: (type-name) var

Increment and Decrement Operators

++n increment, then operate
n++ operate, then increment

--n
n--

only goes on vars, not expressions

Bitwise Operators

&	bitwise AND
|	bitwise inclusive OR
^	" " exclusive OR
<<	left shift
>>	right shift
~	unary

Assignment Operators and Expressions

works w/ most binary operators

+=
op=

operator =

+=	*=	%=	>>=	^=
-=	/=	<<=	&=	|=

Conditional Expressions

expression1 ? expression2 : expression3

if expression1 is nonzero/true, 2 is executed. else, 3 is executed

max of a & b

→ g = (a > b) ? a : b