terminal vars
code output

Getting Started

need file w/extension .c
compile w/ command : cc hello.c
makes executable : a.out

Special function : main( )
program starts @ main( ) → all programs need one

`#include <stdio.h>`
include info about standard input & output

parantheses of f^n hold argument list

statements of a f^n are enclosed in braces { }

function calls can't go over multiple lines

within string constant:

printf( )
doesn't make new line for each new call

\n	→ new line
\t	→ tab
\b	→ backspace non destructively moves cursor hello! \b k → hellok
\"	→ double quote
\\	→ backslash

Variables and Arithmetic Expressions

wrap comments around : /* comment */

all variables must be declared before used
↳ announce properties of variables
↳ typename list of vars

int	integer
float	floating point number
char	character - single byte
short	short int
long	long int
double	double - precision float

assignment statements set vars to initial values

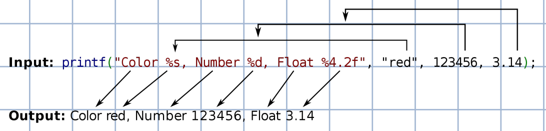
while loops repeats once per output line

condition in parentheses tested
true → do body of loop statement
false → nah

```
int i =0
int j =5
while ( i < j)
{
    i += 1
}
```

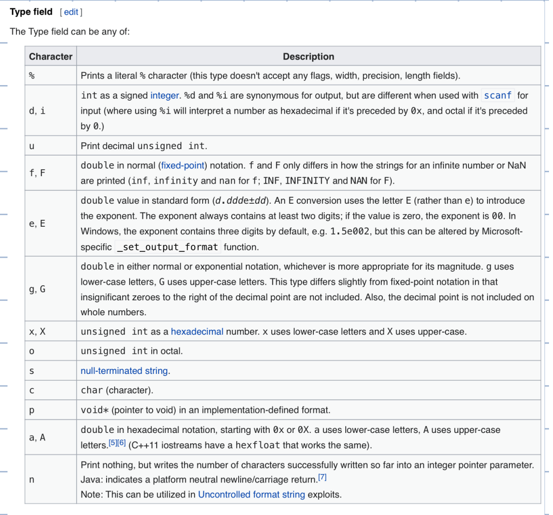
can use % as format specifier, splicing something into string

print f(" %3.4 d " , var)
↳ optional width (3) & optional digits after decimal (4)



Input: `printf("Color %s, Number %d, Float %4.2f", "red", 123456, 3.14);`

Output: Color red, Number 123456, Float 3.14



Type field [edit]

The Type field can be any of:

Character	Description
%	Prints a literal % character (this type doesn't accept any flags, width, precision, length fields).
d, i	`int` as a signed integer. `%d` and `%i` are synonymous for output, but are different when used with `scanf` for input (where using `%i` will interpret a number as hexadecimal if it's preceded by `0x`, and octal if it's preceded by `0`.)
u	Print decimal `unsigned int`.
f, F	`double` in normal (fixed-point) notation. `f` and `F` only differs in how the strings for an infinite number or NaN are printed (`inf`, `infinity` and `nan` for `f`; `INF`, `INFINITY` and `NAN` for `F`).
e, E	`double` value in standard form (*d.ddde±dd*). An E conversion uses the letter E (rather than e) to introduce the exponent. The exponent always contains at least two digits; if the value is zero, the exponent is `00`. In Windows, the exponent contains three digits by default, e.g. `1.5e002`, but this can be altered by Microsoft-specific `_set_output_format` function.
g, G	`double` in either normal or exponential notation, whichever is more appropriate for its magnitude. `g` uses lower-case letters, `G` uses upper-case letters. This type differs slightly from fixed-point notation in that insignificant zeroes to the right of the decimal point are not included. Also, the decimal point is not included on whole numbers.
x, X	`unsigned int` as a hexadecimal number. `x` uses lower-case letters and `X` uses upper-case.
o	`unsigned int` in octal.
s	null-terminated string.
c	`char` (character).
p	`void*` (pointer to void) in an implementation-defined format.
a, A	`double` in hexadecimal notation, starting with `0x` or `0X`. `a` uses lower-case letters, `A` uses upper-case letters.[5][6] (C++11 iostreams have a `hexfloat` that works the same).
n	Print nothing, but writes the number of characters successfully written so far into an integer pointer parameter. Java: indicates a platform neutral newline/carriage return.[7] Note: This can be utilized in Uncontrolled format string exploits.

scanf ( ) is similar, but it reads input

when working w/float, operating on an int will turn int into a float

The For Statement

① initialization
② test or condition
③ increment

```
for ( init = # ; test ; increment)
{
        body
}
```

Symbolic Constants

don't use magic numbers

`# define    VARIABLE        value`

can define a symbolic constant ~ global constant

Character Input and Output

text input & output dealt w/ as streams of characters

text stream – sequence of characters divided into lines
—▷ each line consists of 0 or more characters followed by newline of characters

standard library has many fⁿ's for reading/writing 1 char at a time –▷ 2 simple ones

getchar ( )	reads next input character
putchar (var)	print contents of var

need a integer to read next integer & another to state integer

file copying
eg to copy files

```
read char
while(char not end of file)
        output char
        read char
```
→ defined in stdio.h

```
int c
```
→ declare as int b/c c must be type big enough to hold EOF & any char

```
c = getchar()
while (c != EOF)
{
    putchar(c)
    c = getchar()
}
```

after assignment, c = getchar() is value of c

can rewrite test to get rid of line & reassignment:

```
while( (c = getchar()) != EOF)
```

character counting

$$i = i+1 \xleftrightarrow{\text{same}} i \mathrel{+}= 1 \xleftrightarrow{\text{same}} i{+}{+} \xleftrightarrow{\text{same}} {+}{+}i$$

```
double nc ;

for (   nc = 0 ; getchar() != EOF; ++nc)
    ;
printf("%.0f\n", nc) ;
```
→ empty body . isolated semicolon → null statement

line counting

```
int   c , n1;

n1 = 0
while (getchar()   != EOF)

     if   ( c == '\n' )
          n1 ++ ;

printf("%d \n", n1) ;
```
equal comparison

if statement!

character constant – character written between single quotes represents an integer value equal to numerical value of character

'A' → 65

'\n' → 10
"\n" → string constant

Word Counting

```
#define IN  1
#define OUT 0

int c, nw;
state = OUT;
 nw = 0
while ((c = getchar()) != EOF)
    if (c == ' ' || c = '\n' || c = '\t')
        state = OUT ;
    else if (state = OUT)
        state = IN
        nw += 1
```

OR operator

&& → AND operator

Arrays

`int ndigit[10]` → array of 10 elements of type integers

0 indexed

characters described by numbers.
the chars '0', '1', ..., '9' are consecutive

'0' - '0' → 0
'5' - '0' → 5

ndigits['7' - '0']
↳ ndigits[7]

```
if (condition1)
    statement1
else if (condition2)
    statement2
:
else
    stament n
```

Functions

has the form:

```
return-type function_name (parameter declarations, if any)
{
    declarations
    statements
}
```

can appear in any order in file

```
int power(int base, int n)
    int i, p
    p = 1
    for ( i = 0; i <= n; ++i)
        p *= base
    return p
```

→ declare param types & names & return type
Will produce error if not

→ doesn't need a return statement

Arguments - Call by Value

in C, function args. are "passed by value"

→ the called f^n is given values of arg by temporary vars intead of originals

can only alter local, private copy

if you loop over a var, doesn't change var

to change var, need to specify address of var

for arrays, it doesn't make a copy → value passed is location of array

Character Arrays

most common array: character arrays

supply size of array to set aside storage

returning no value? return type is void