The C programming Language

The C Programming Language: *classes of lexical atoms*

Any source text is composed of tokens *(lexical atoms)*

C tokens:

- **reserved keywords**
- literal constants *(including string constants)*
- identifiers *(variable and function names as well as labels)*
- separators *(including operators)*

```
main()
{
char c;
while((c=getchar()) != -1)
       putchar(c);
}
```

Structural components of C programs

- preprocessor directives
- declarations
  - of variables
  - of functions
- definitions of functions
- comments

```
/*  copy input to output, in upper case  */
#include <stdio.h>

char to_upper(char);

main()
{
char c;
while((c=getchar()) != -1)   /*read character*/
        putchar(to_upper(c));  /*convert & print it */
}
```

The C Programming Language

PROGRAM = DATA + ALGORITHM

Data: *classification*

As operands:

- constants
  - - literal constants: `-15, 0 , 123, 45.67, 'a', "Timisoara"`
  - - symbolic constants: `PI, EOF, TRUE, FALSE`
- variables *(identifiable storage areas in memory, which hold some values)*

Data: *classification*

by *(internal)* **representation:**

- **integers**
  - - unsigned *(plain binary)*
  - - signed *(2's complement)*
- **reals** *(floating point)*

Data: *classification*

by interpretation

- numbers
- characters
- addresses

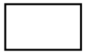
E.g.

```
65+10
putchar(65);
*0x6510 = 1;
```

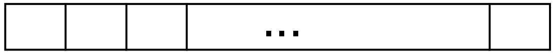
Data: *classification*

by aggregation scale

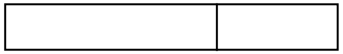
- scalar



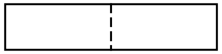
- aggregate:
  - arrays



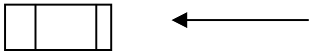
  - structures



  - unions



  - bit-fields



byte/multiple of bytes

Data: *classifications*

special data:

- character strings: “Timisoara” “9876”

‘T’	‘i’	‘m’	‘i’	‘s’	‘o’	‘a’	‘r’	‘a’	‘**\0**’	‘9’	‘8’	‘7’	‘6’	‘**\0**’	

- enumerations:

```
enum colors { BLACK, RED, GREEN, BLUE, WHITE};
```

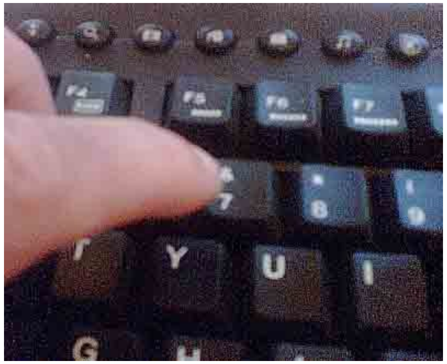
BLACK		=0
RED	=BLACK+1	=1
GREEN	=RED+1	=2
BLUE	=GREEN+1	=3
WHITE	=BLUE+1	=4

Data: *representation*

Two different views:

- the human user → external representation

- the computer → internal representation



Data: *internal representation*

- **2's complement**, for (signed) ***integers***
- **floating point**, for ***reals***

Number of bits: 4

Binary configuration	Value as unsigned	Value as signed	
0000	0	0	>= 0
0001	1	1	
0010	2	2	
0011	3	3	
0100	4	4	
0101	5	5	
0110	6	6	
0111	7	7	
1000	8	-8	< 0
1001	9	-7	
1010	10	-6	
1011	11	-5	
1100	12	-4	
1101	13	-3	
1110	14	-2	
1111	15	-1	

See more about 2’s complement here

Data: *internal representation of reals*

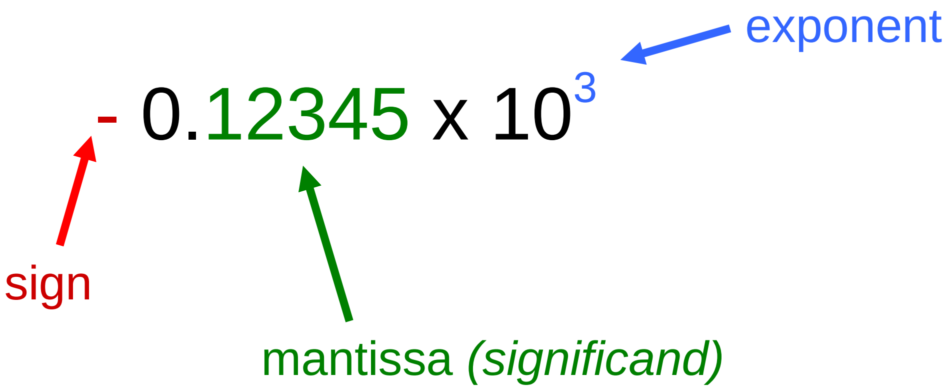
- **floating point**, for ***reals***

Based on separate representation of:

- sign
- digits
- magnitude (exponent)

E.g.

$N = -123.45 \quad = -\ 1234.5 \times 10^{-1} = -\ 12.345 \times 10^{1} \quad \ldots = -\ 0.12345 \times 10^{3} \ldots$



exponent

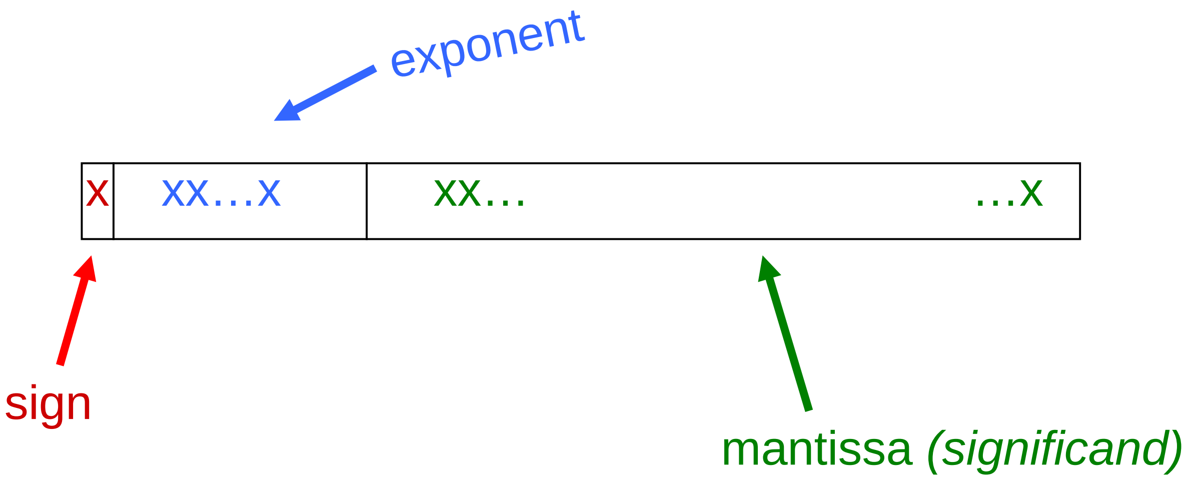
$-\ 0.12345 \times 10^{3}$

sign

mantissa *(significand)*

Data: *internal representation of reals*

- **floating point**, for ***reals***



Some references

The IEEE standard for floating point arithmetic

IEEE-754 References

Floating-Point Number Tutorial

Data Types: *tentative definitions*

Data type: **range of values and the set of admitted operations**

E.g.

[0, 255], {+, -, *, /, %, &, I, ^, ~, …}

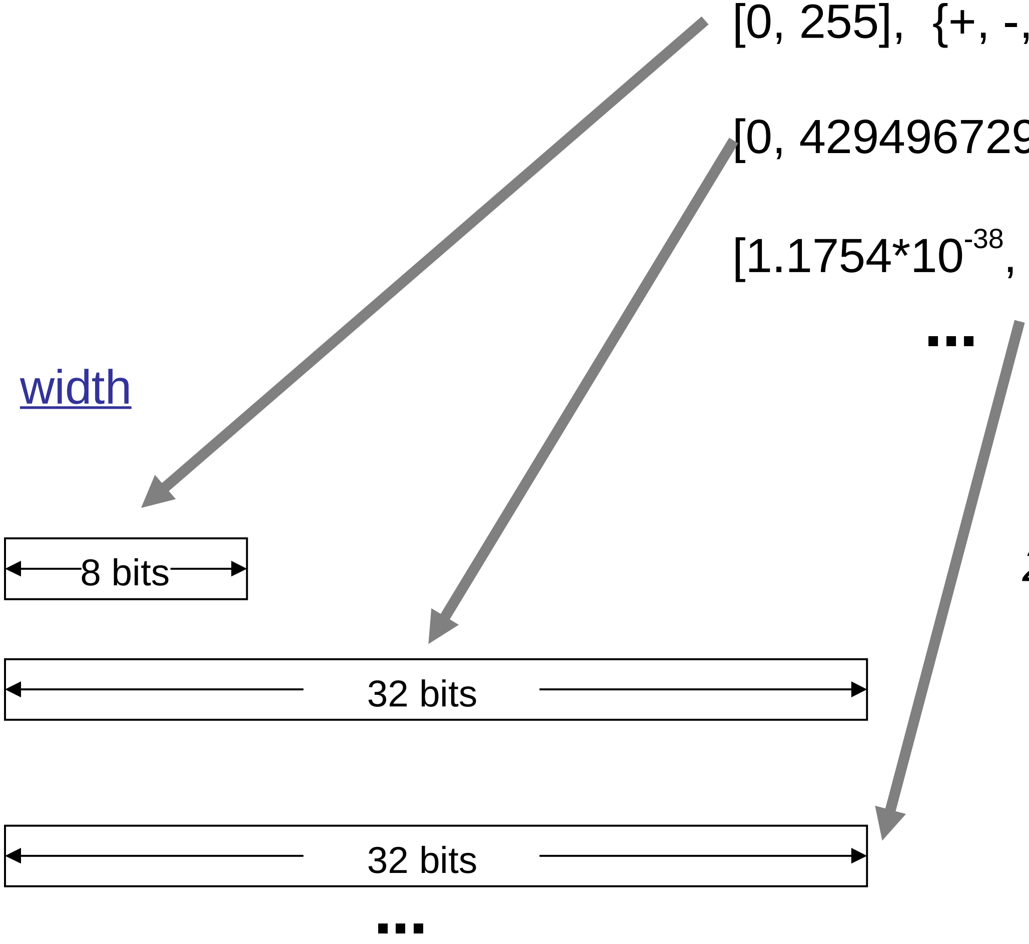
[0, 4294967295], {+, -, *, /, %, &, |, ^, ~, …}

[1.1754*10^{-38}, 3.4028*10^{38}], {+, -, *, /, …}

…

width

representation



2’s complement *(unsigned)*

2’s complement *(unsigned)*

floating point

Data: *C data types*

	representation	width	range		name
			unsigned	signed	
• integer	*2's complement*	8 bits	[0, 255] *$[0, 2^{8}-1]$*	[-128, 127] $[-2^{7} , 2^{7}-1]$	**char**
		16 bits	[0, 65535] *$[0, 2^{16}-1]$*	[-32768, 32767] $[-2^{15} , 2^{15}-1]$	**short** int
					int (implem. defined)
		32 bits	[0,4294967295] *$[0, 2^{32}-1]$*	[-2147483648, 2147483648] $[-2^{31} , 2^{31}-1]$	**long** int
• real	*floating point*	32 bits	$[1.1754*10^{-38}, 3.4028*10^{38}]$		**float**
		64 bits	...		**double**
		80 bits	...		**long double**

Data: *type specification*

- **literal constants**

  - implicit/explicit by their value and some suffixes

- **variables**

  - through declarations

Data: *integer constants*

integer-constant:

decimal-constant *integer-suffix*$_{opt}$

octal-constant *integer-suffix*$_{opt}$

hexadecimal-constant *integer-suffix*$_{opt}$

decimal-constant:

nonzero-digit

decimal-constant *digit*

octal-constant:

0

octal-constant *octal-digit*

hexadecimal-constant:

hexadecimal-prefix *hexadecimal-digit*

hexadecimal-constant *hexadecimal-digit*

hexadecimal-prefix:

one of **0x 0X**

integer-suffix:

unsigned-suffix *long-suffix*$_{opt}$

long-suffix *unsigned-suffix*$_{opt}$

nonzero-digit:

one of **1 2 3 4 5 6 7 8 9**

octal-digit:

one of **0 1 2 3 4 5 6 7**

hexadecimal-digit:

one of **0 1 2 3 4 5 6 7 8 9**

a b c d e f

A B C D E F

unsigned-suffix:

one of **u U**

long-suffix:

one of **l L**

Examples

`123, 50000`

`50000U, 1UL`

`0, 00123, 0177`

`0x1, 0xA16C, 0XFF`

Data: *type of integer constants*

The **type** of an integer constant is the *first* of the corresponding list in which its value can be represented. (see ISO/IEC 9899:1990, pp27; ISO/IEC 9899:1999, pp 56)

Suffix	DecimalConstant	Octal or Hexadecimal Constant

none	`int`
	`long int`
		`int`
		`unsigned int`
		`long int`
		`unsigned long int`

`u` or `U`	`unsigned int`
	`unsigned long int`
		`unsigned int`
		`unsigned long int`

`l` or `L`	`long int`	`long int`
		`unsigned long int`

Both `u/U or l/L`	`unsigned long int`	`unsigned long int`

Data: *character constants*

Either:

‘*c_character*’

or

simple-escape-sequence
octal-escape-sequence
hexadecimal-escape-sequence

e.g.
‘a’ ‘b’ ‘A’ ‘0’ ‘9’ ‘+’ ‘*’ ‘&’
~~‘\’~~ ~~‘‘’~~

escape sequences:

simple-escape-sequence: one of
**\' \" \? **
\a \b \f \n \r \t \v

e.g.
‘\’ ‘\\’ ‘\n’ ‘\t’

octal-escape-sequence:
**** *octal-digit*
**** *octal-digit octal-digit*
**** *octal-digit octal-digit octal-digit*

e.g.
‘\0’ ‘\60’ ‘\142’

hexadecimal-escape-sequence:
\x *hexadecimal-digit*
hexadecimal-escape-sequence hexadecimal-digit

e.g.
‘\x1’ ‘\x41’ ‘\x30’

An character-constant has type **`int`**.

Data: *floating point constants*

floating-constant:

decimal-floating-constant:

fractional-constant exponent-part$_{opt}$ *floating-suffix*$_{opt}$
digit-sequence exponent-part floating-suffix$_{opt}$

fractional-constant:

digit-sequence$_{opt}$ **.** *digit-sequence*
digit-sequence **.**

digit-sequence:

digit
digit-sequence digit

exponent-part:

e *sign*$_{opt}$ *digit-sequence*
E *sign*$_{opt}$ *digit-sequence*

sign:

one of **+ -**

floating-suffix:

one of **f l F L**

Examples

```
3.14
0.1
 .15
123.
1.5 E2
1.5 e+2
1E-6
1.5F
1.5L
```

An unsuffixed floating constant has type **double**.
If suffixed by the letter **f** or **F**, it has type **float**.
If suffixed by the letter **l** or **L**, it has type **long double**.

DATA: *variables...*

Terms:

object,
lvalue,
declaration of variables,
storage class of a variable,
scopes of an identifier
visibility of a variable,
duration of a variable,
linkage of an identifier