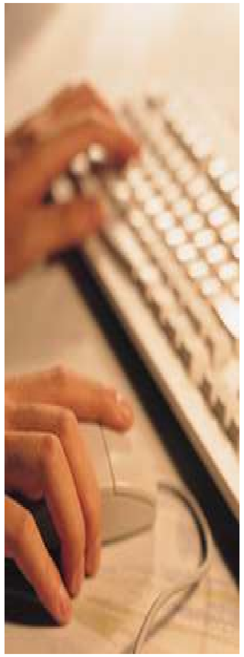
Table of contents

- History
- Algorithms
- Nassi-Shneiderman-Diagrams
- Programming Tools
- Variables
- Preferences / Attributes of C
- Constant Values
- **Operators and Operations**
- Functions
- Classes of Data Types
- Local and Global Variables

Operators and Operations



Priority	Operator	associativity
1	() [] -> .	left
2	! ~ ++ -- - (data type) * & sizeof	right
3	* / %	left
4	+ -	left
5	<< >>	left
6	< <= > >=	left
7	== !=	left
8	&	left
9	^	left
10	|	left
11	&&	left
12	||	left
13	? :	right
14	= += -= *= /= %= &= ^= != <<= >>=	right
15	,	left

Operators

- operators are applied to operands
- operators and operands form operations
- every operation has a result
- an operation generates a value
- be careful: there are side effects / spin-offs

Side effects are often the origin of semantic errors.

You cannot find any syntactical errors there!

Punctuation Marks

- symbol that is no operation
- independent syntactical and semantical meaning
- punctuation marks in C
  - [] () {}
  - * , :
  - = ; …
  - #

() = parenthesis; [] = squared brackets

{} = braces

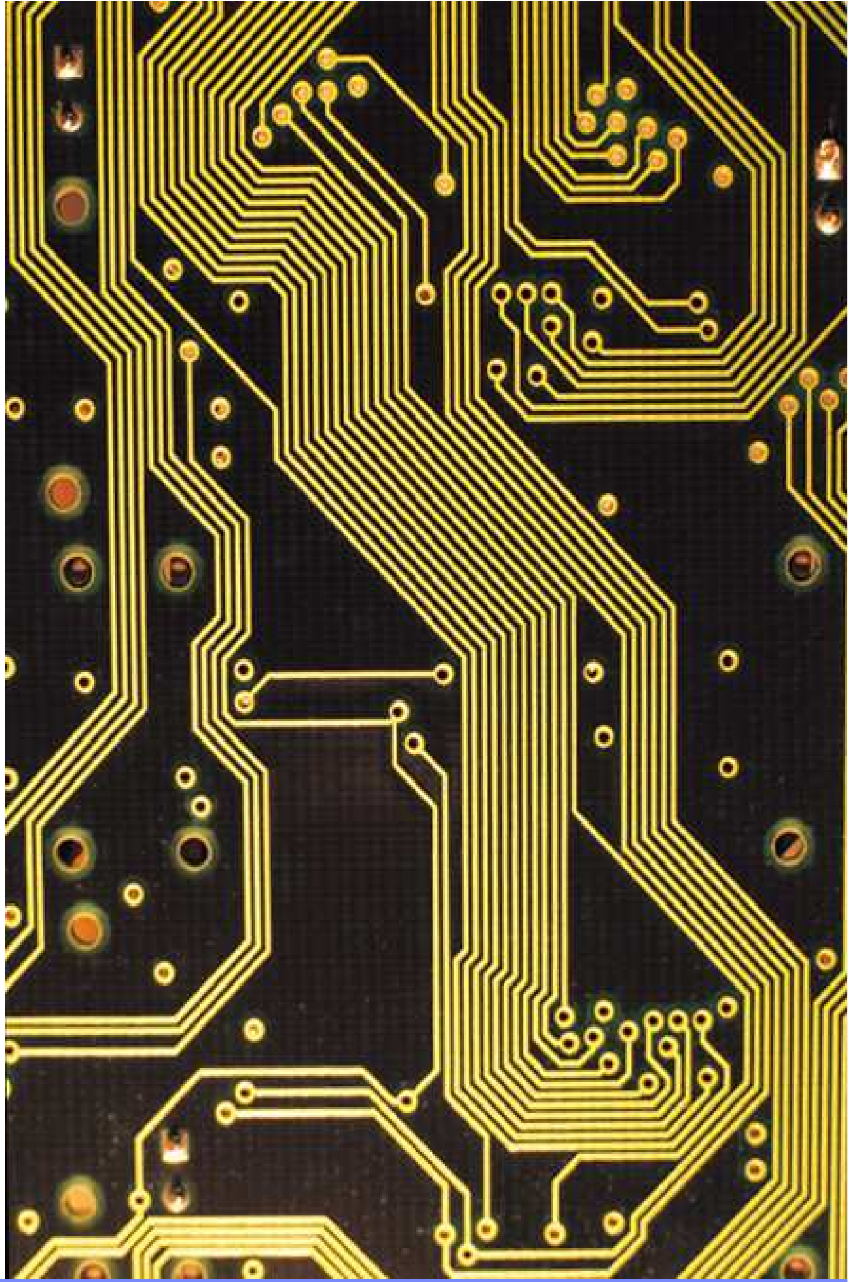
Table of contents

Functions in C

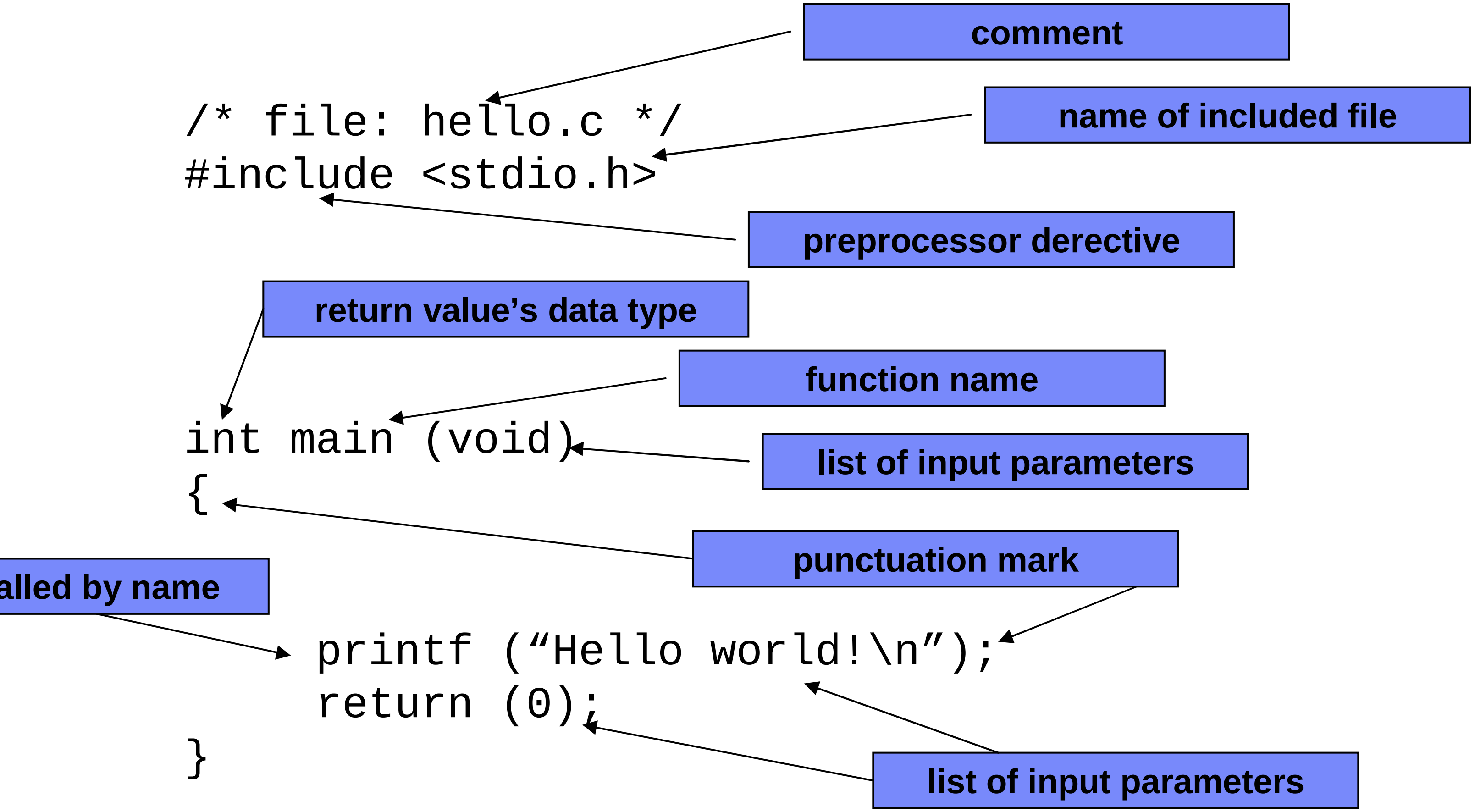
- C only has functions.
- There aren’t any procedures.
- A function can call another function.
- A function cannot be defined within another function.
- A function can call itself.
- A function has a return value.
  → This is the major characteristic of a function.

Program main

- The main function is the MAIN function in every program.
- The main function is executed first.
- The main function is started at the program’s start.
- You need at least the main function.
- Without a main function you are coding, oh!, programming a library.



The famous program “Hello Word!”



comment

name of included file

preprocessor derective

return value’s data type

function name

list of input parameters

punctuation mark

function called by name

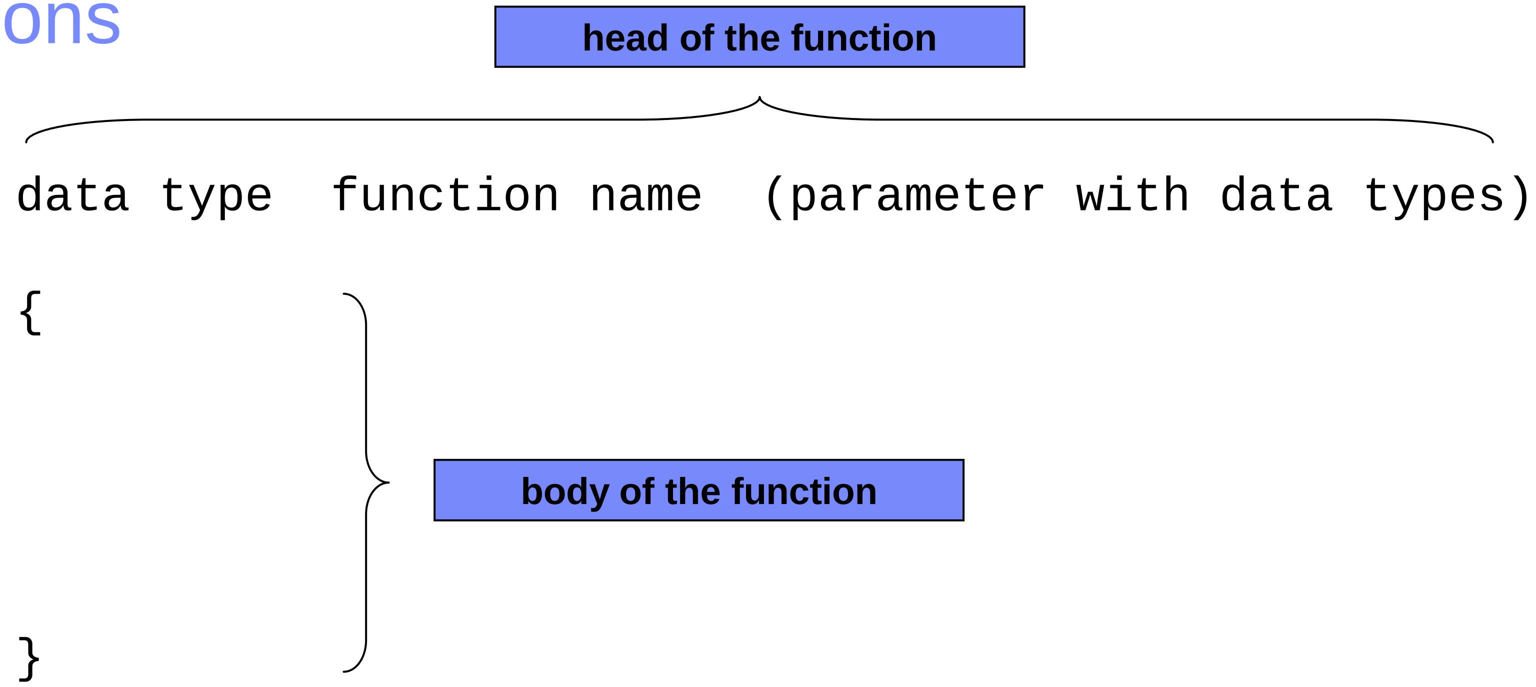
list of input parameters

```
/* file: hello.c */
#include <stdio.h>

int main (void)
{
    printf (“Hello world!\n”);
    return (0);
}
```

This is the most famous program code I know, even in Pascal.
Pascal, what the hell is that?   … and this confusion

Functions



head of the function

```
data type  function name  (parameter with data types)

{



}
```

body of the function

Functions you need – the first step

```
int printf (const char * string, …);

int getchar (void);

int scanf (const char * format, …);
```

You will need more than only these functions above in the future, but now it is enough to code some simple programs in C.

Table of contents

History

Algorithms

Nassi-Shneiderman-Diagrams

Programming Tools

Variables

Preferences / Attributes of C

Constant Values

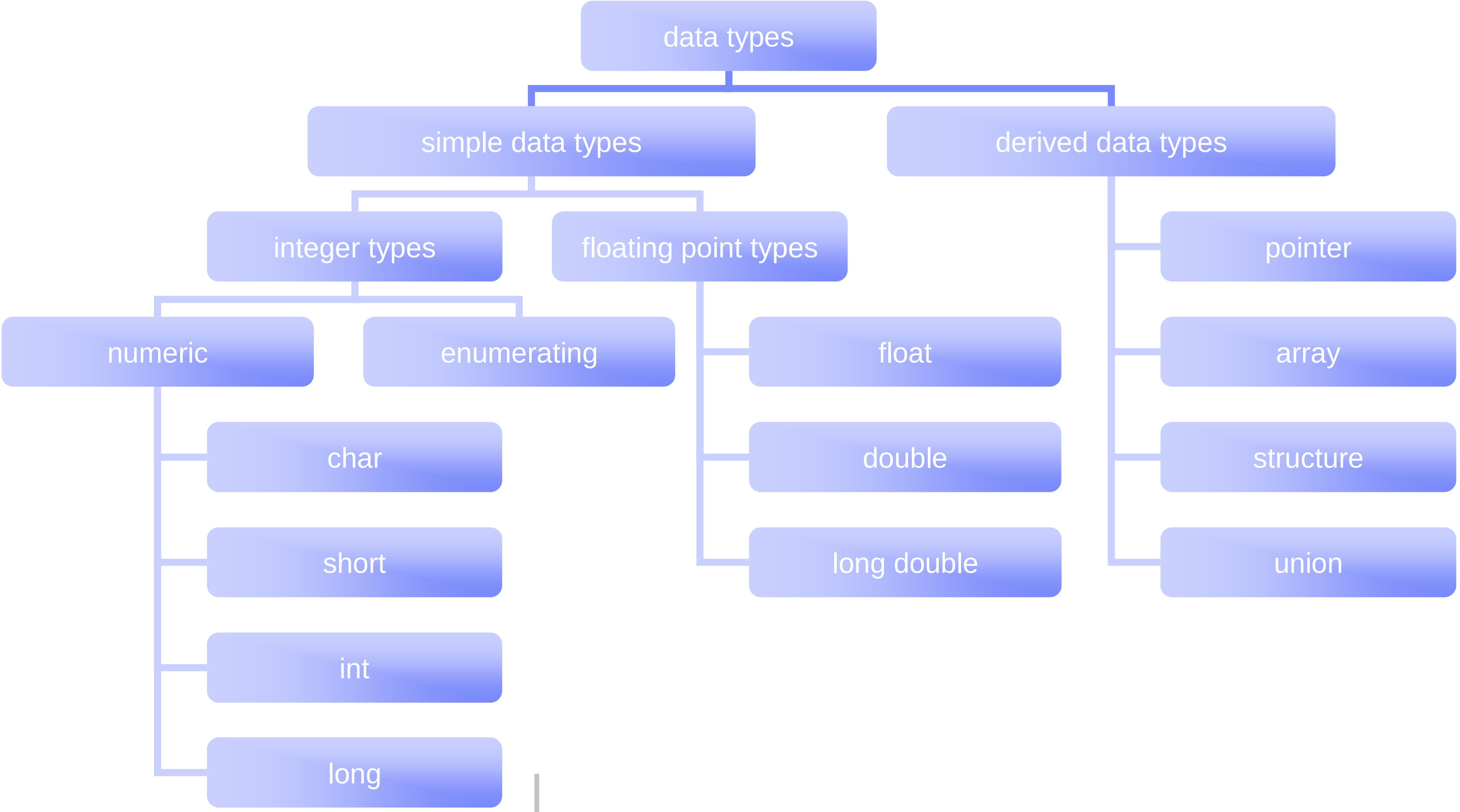
Operators and Operations

Functions

Classes of Data Types

Local and Global Variables

Classes of Data Types



data types

simple data types

derived data types

integer types

floating point types

numeric

enumerating

char

short

int

long

float

double

long double

pointer

array

structure

union

C has no data type for characters and strings!
Characters are represented by numeric values.

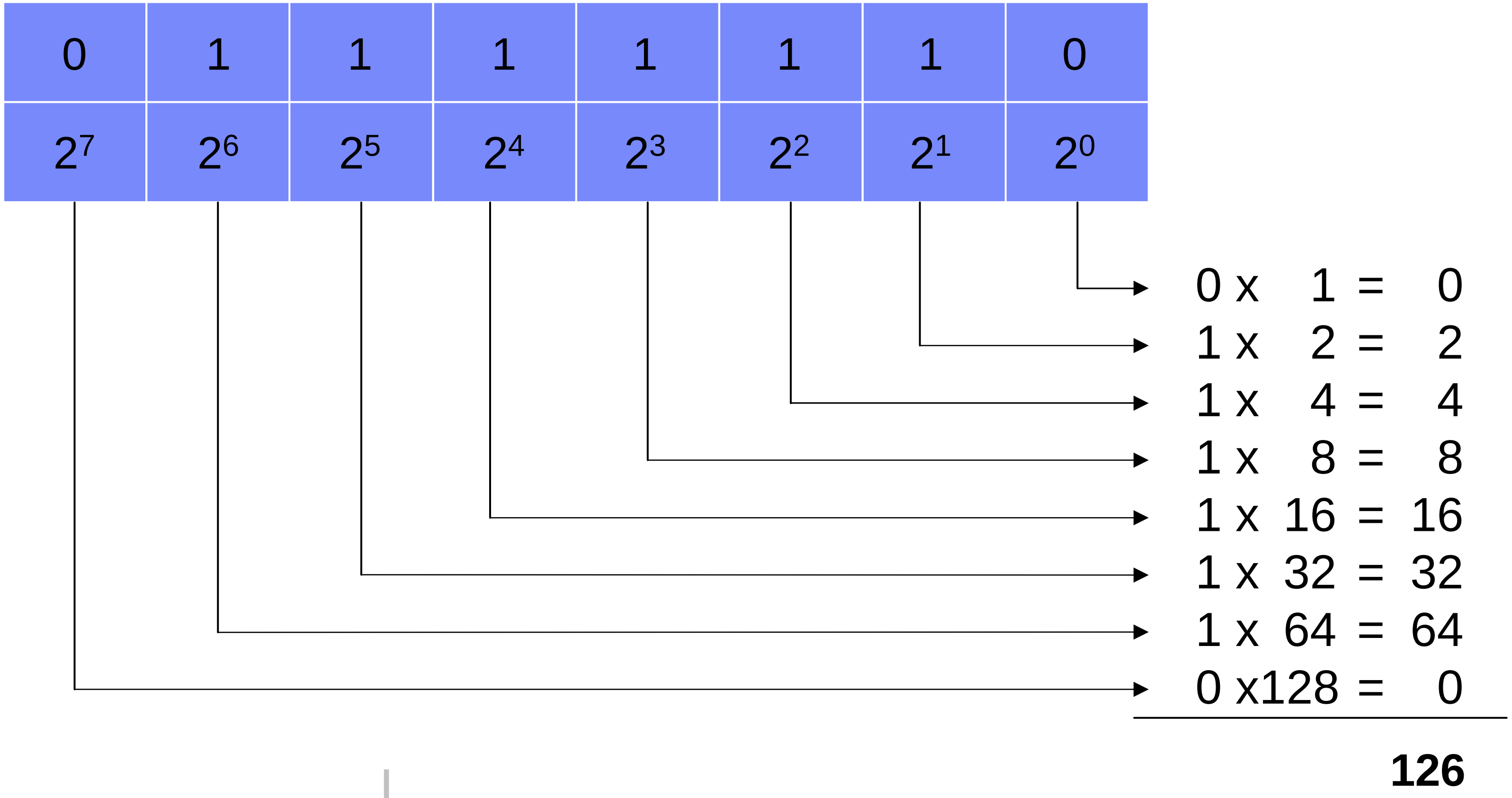
Modifier

- The range of the data type is predetermined by the modifier
  - signed
  - unsigned
- The data type `char` is either `signed char` or `unsigned char`. This depends on the implementation.
- All other data types are normally implemented as unsigned data types.

Classes of Data Types

		range	
	length	min.	max.
signed char	1 byte	-128	+ 127
unsigned char		0	+ 255
signed short	2 bytes	- 32.768	+ 32.768
unsigned short		0	+ 65.535
signed int	2 bytes	-32.768	+32.767
	4 bytes	-2.1147.483.648	+2.147.483.647
unsigned int	2 bytes	0	+65.535
	4bytes	0	+4.294.967.295
signed long	4 bytes	-2.147.483.648	+2.147.483.648
unsigned long		0	+4.294.967.295

Data type **`char (unsigned char)`**



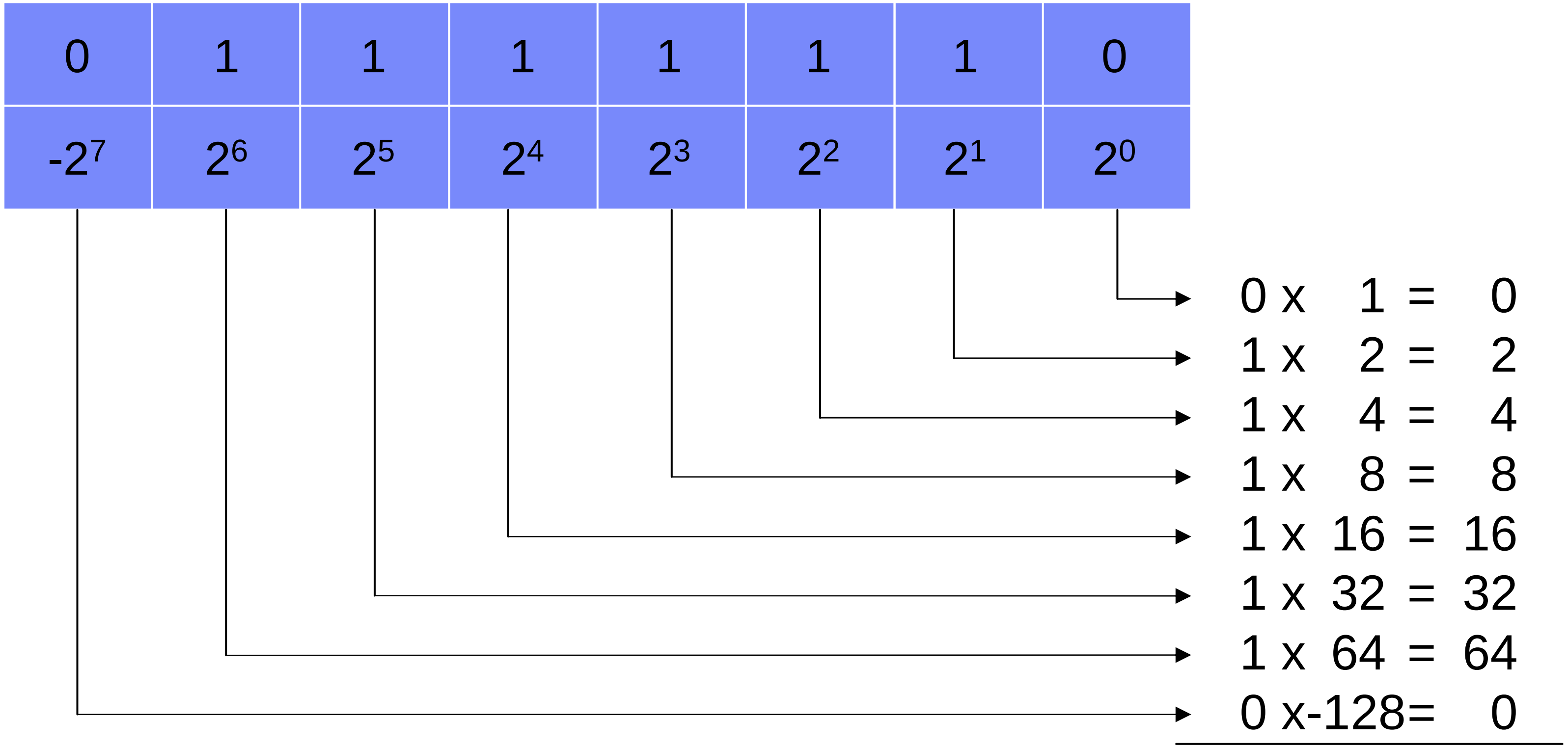
0	1	1	1	1	1	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

0 x 1 = 0
1 x 2 = 2
1 x 4 = 4
1 x 8 = 8
1 x 16 = 16
1 x 32 = 32
1 x 64 = 64
0 x128 = 0

126

The data type char is used to represent a character from the ASCII character table.

Data type **char (signed char)**



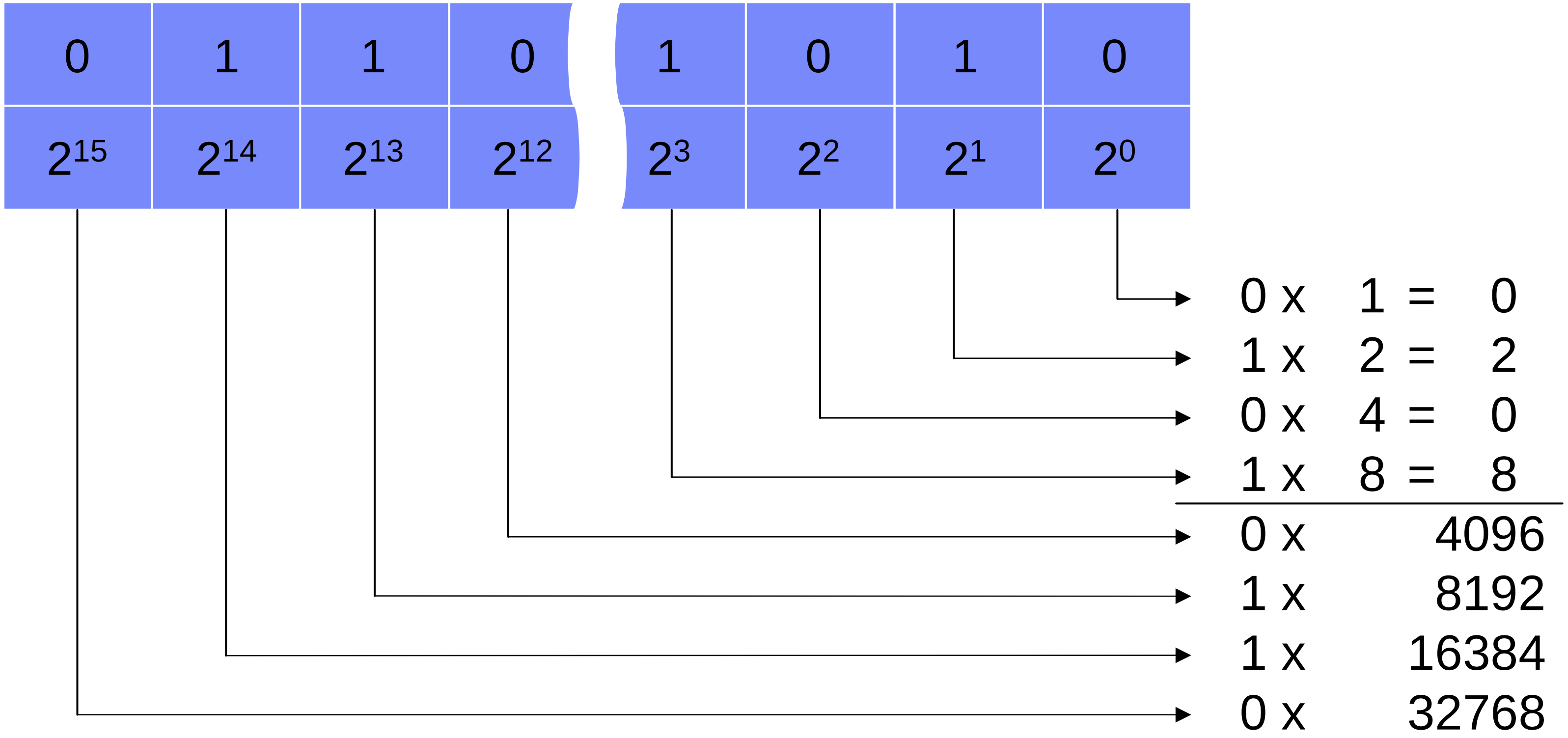
0	1	1	1	1	1	1	0
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

0 x 1 = 0
1 x 2 = 2
1 x 4 = 4
1 x 8 = 8
1 x 16 = 16
1 x 32 = 32
1 x 64 = 64
0 x-128= 0

126

In this example the bit sequence 0111 1110 represents the same value in signed and unsigned.

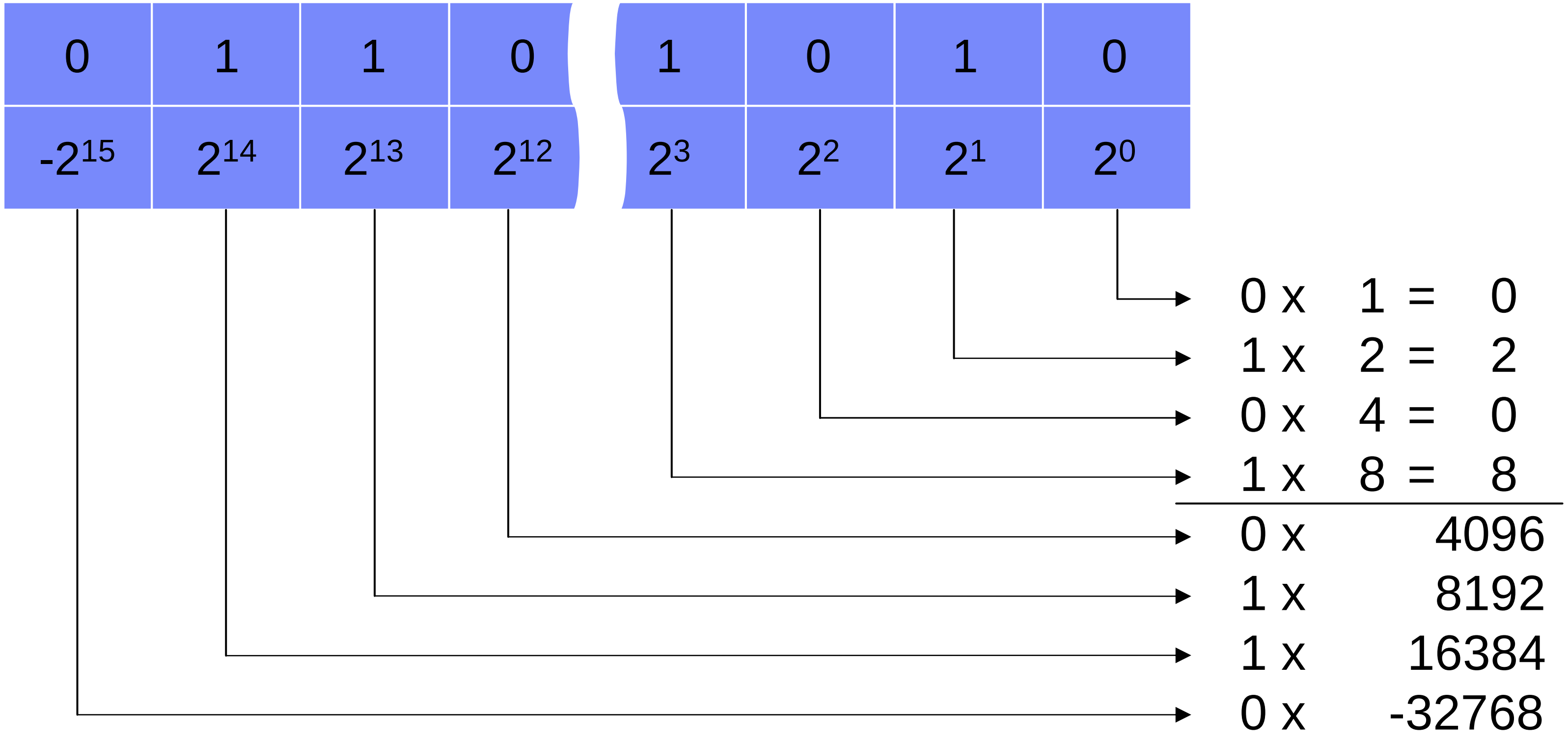
Data type **`short`** **`(unsigned short)`**



0	1	1	0	1	0	1	0
2^{15}	2^{14}	2^{13}	2^{12}	2^{3}	2^{2}	2^{1}	2^{0}

0 x 1 = 0
1 x 2 = 2
0 x 4 = 0
1 x 8 = 8
0 x 4096
1 x 8192
1 x 16384
0 x 32768

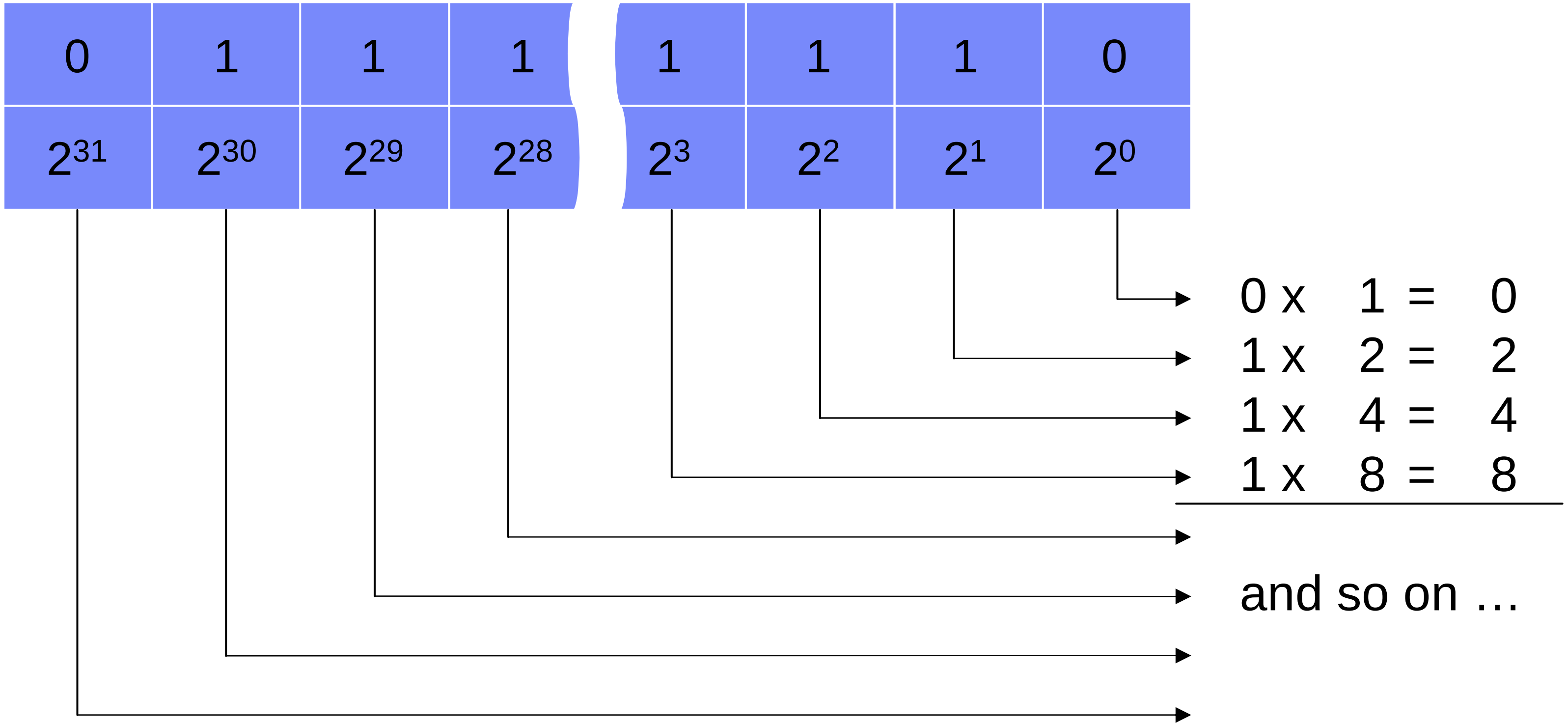
Data type `short (signed short)`



0	1	1	0	1	0	1	0
-2^{15}	2^{14}	2^{13}	2^{12}	2^{3}	2^{2}	2^{1}	2^{0}

0 x 1 = 0
1 x 2 = 2
0 x 4 = 0
1 x 8 = 8
0 x 4096
1 x 8192
1 x 16384
0 x -32768

Data type `int (unsigned int)`



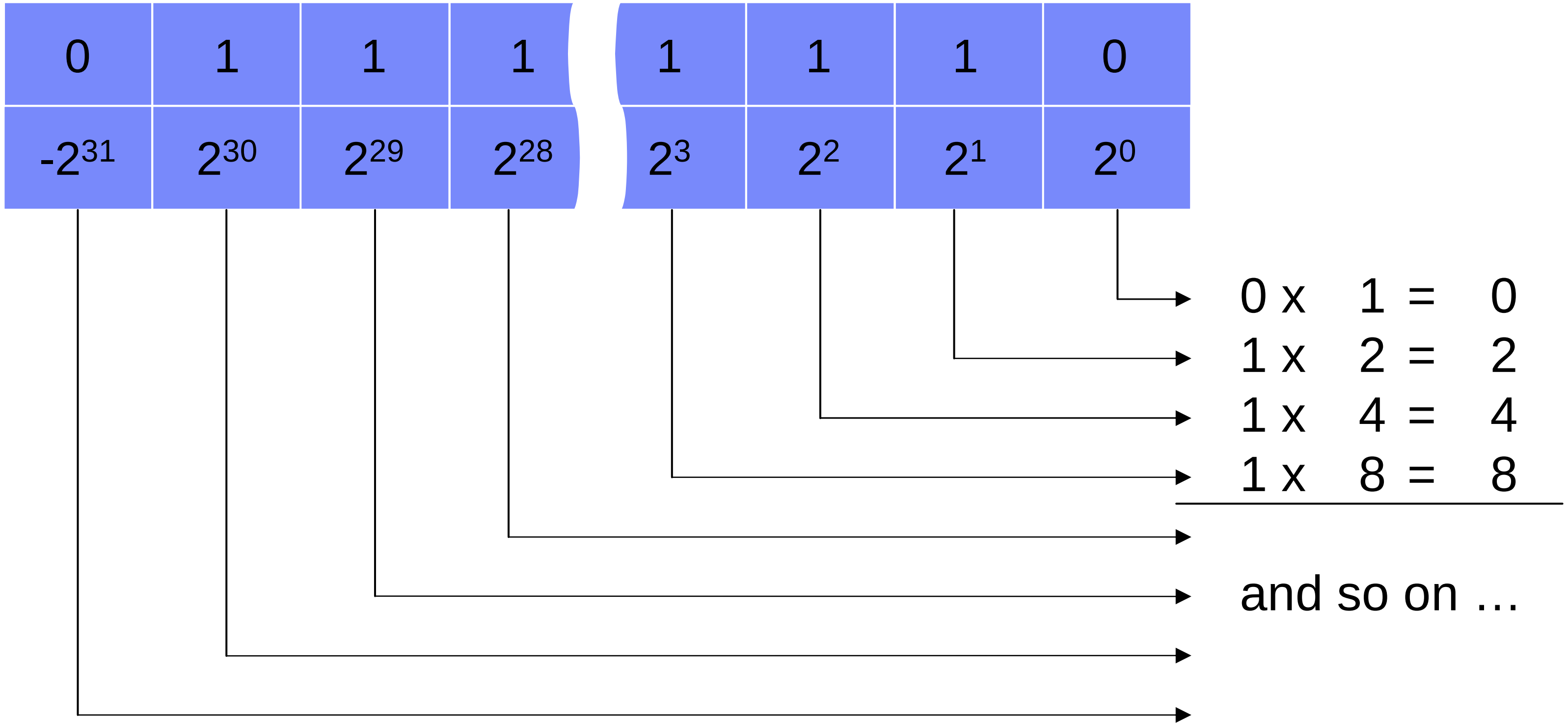
0	1	1	1	1	1	1	0
2^{31}	2^{30}	2^{29}	2^{28}	2^{3}	2^{2}	2^{1}	2^{0}

0 x 1 = 0
1 x 2 = 2
1 x 4 = 4
1 x 8 = 8

and so on …

We assume that the data type int is implemented with 4 bytes / 32 bits.

Data type `int (signed int)`



0	1	1	1	1	1	1	0
-2^{31}	2^{30}	2^{29}	2^{28}	2^{3}	2^{2}	2^{1}	2^{0}

0 x 1 = 0
1 x 2 = 2
1 x 4 = 4
1 x 8 = 8

and so on …

We assume that the data type int is implemented with 4 bytes / 32 bits.