GlobalLogic®

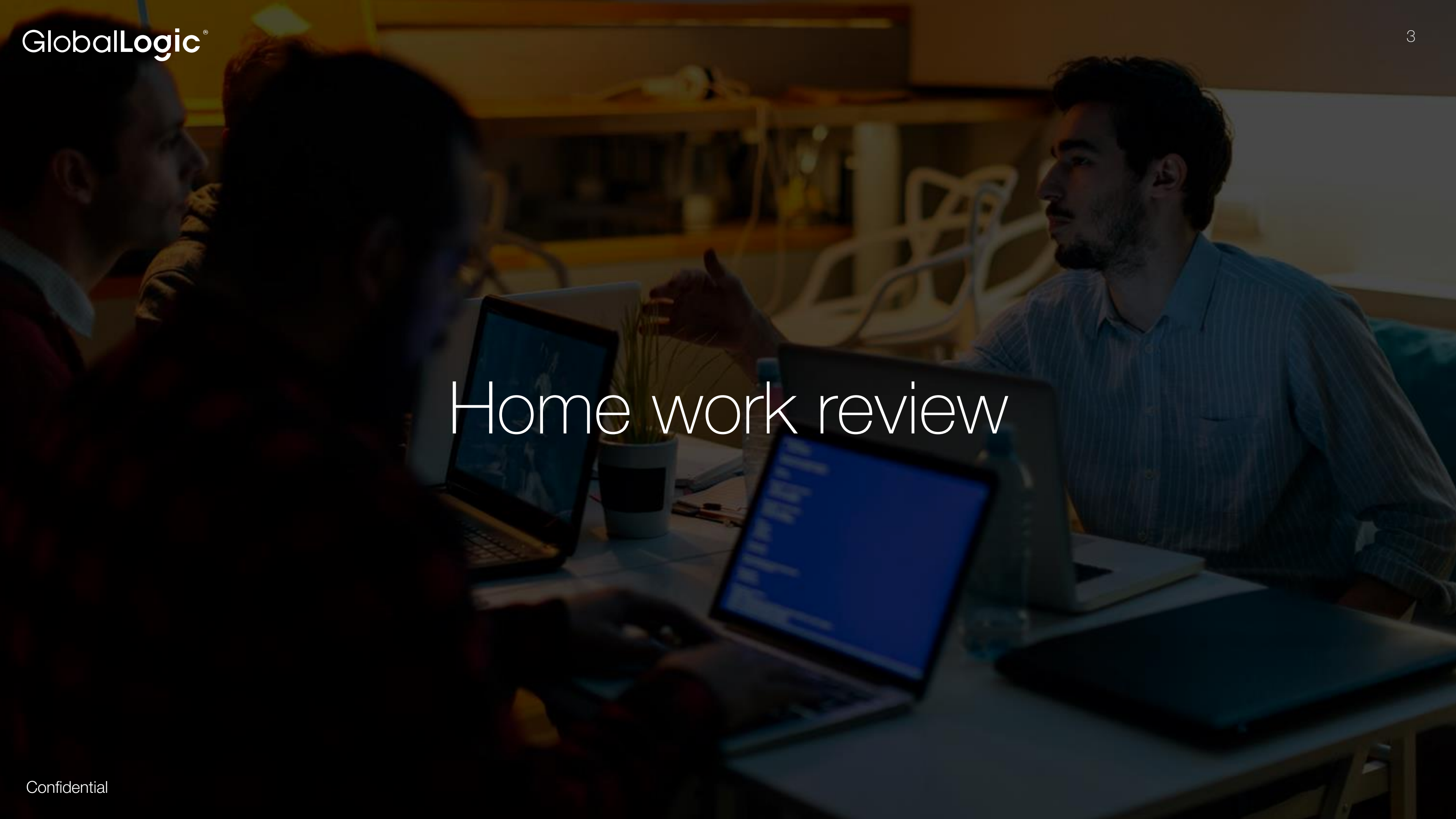
Basecamp: C/Embedded

3

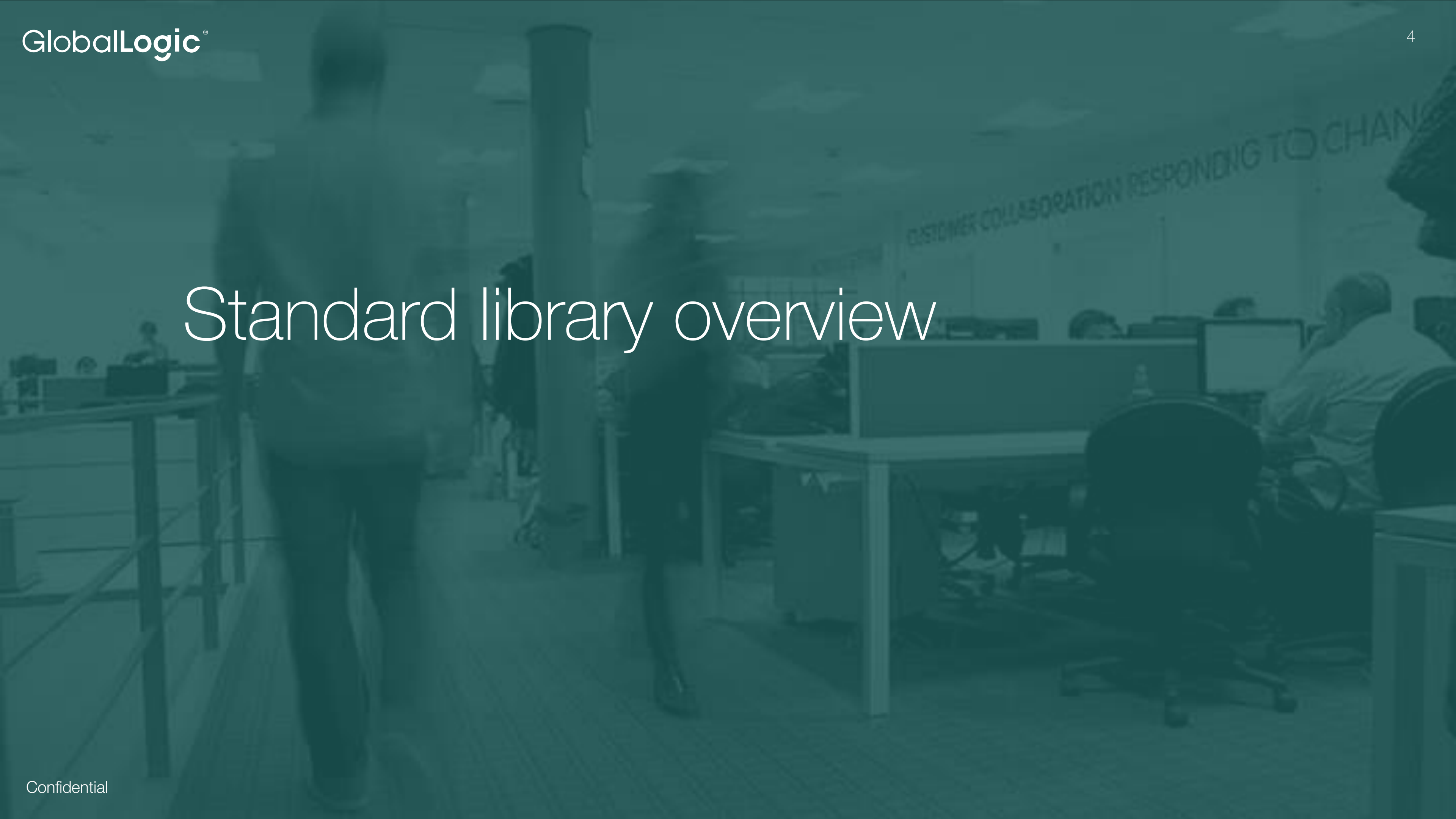
GlobalLogic®

Agenda

1. Homework review
2. Standard library
   - Short reference
3. Arithmetic and operators
   - Binary system
   - Hex
   - Other 2-base systems and conversions
   - OR, AND, NOT, XOR
   - Logic tables and composing logic tables



Home work review



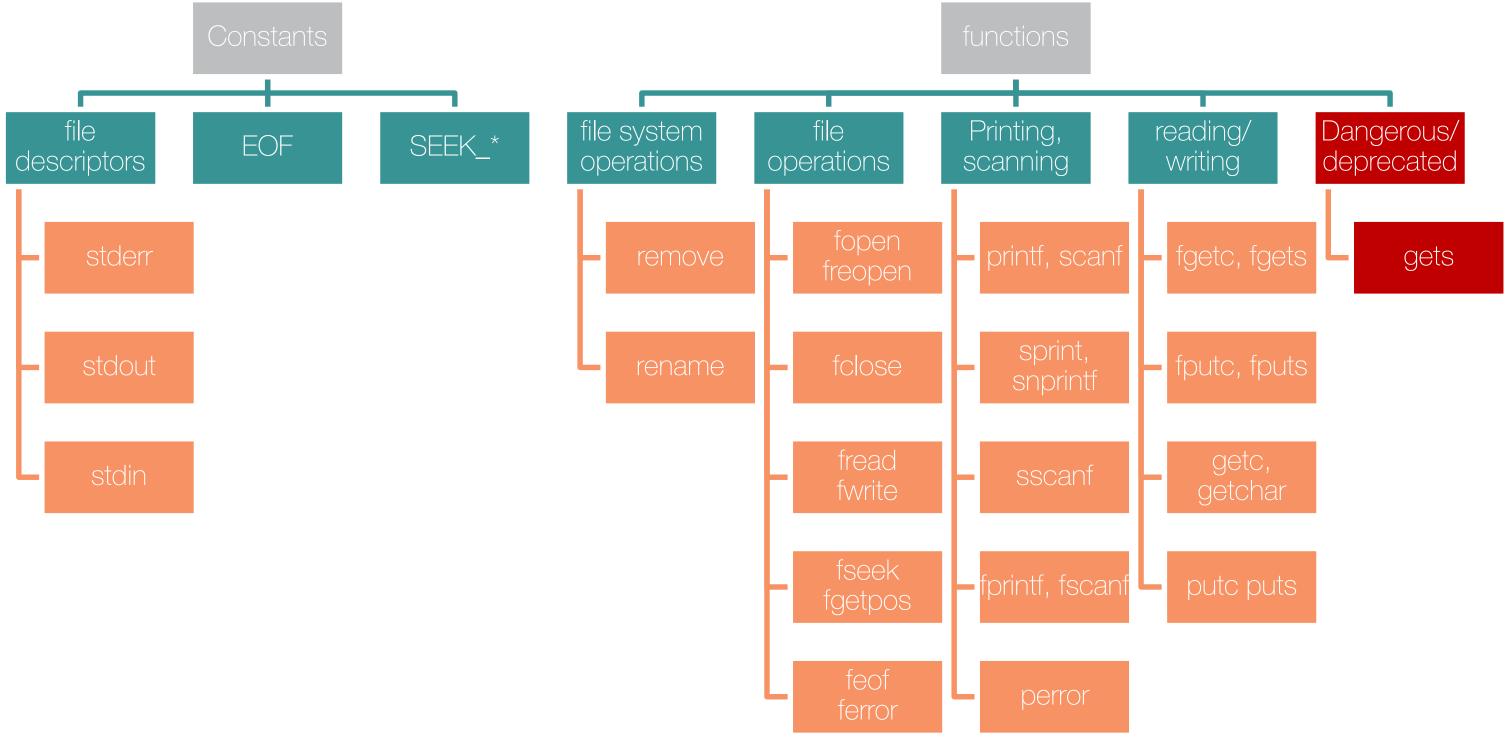
Standard library overview

GlobalLogic

C99 standard library

Types	Character and strings	OS and syscalls	Math	Other
☐ limits.h	☐ string.h	☐ errno.h	☐ math.h	☐ stdlib.h
☐ stdbool.h	☐ ctype.h	☐ signal.h	☐ complex.h	☐ stdarg.h
☐ stddef.h (default)	☐ locale.h	☐ stdio.h	☐ fenv.h	☐ assert.h
	☐ wchar.h	☐ time.h	☐ float.h	☐ iso646.h
	☐ wctype.h		☐ tgmath.h	☐ setjmp.h
	☐ inttypes.h			

<stdio.h>


Constants
file descriptors
stderr
stdout
stdin
EOF
SEEK_*
functions
file system operations
remove
rename
file operations
fopen freopen
fclose
fread fwrite
fseek fgetpos
feof ferror
Printing, scanning
printf, scanf
sprint, snprintf
sscanf
fprintf, fscanf
perror
reading/ writing
fgetc, fgets
fputc, fputs
getc, getchar
putc puts
Dangerous/ deprecated
gets

stdio printing

- Try to always use snprintf where applicable
    - remember, its return value differs from sprintf
    - It's done intentionally, check the example
- When specifying format descriptor for printf/sprint/snprintf, always try to specify most restricted format
    - Ex: `printf("%hd", short_value)` instead of `printf("%d", short_value)`
    - Format = % + modifier + width + type
    - width: h, hh, l, ll, z

```
/* use case: writing to possibly small buffer */
char buf[5];

int print_number(const int n)
{
    if (snprintf(buf, sizeof(buf), "%04d", n) > \
            sizeof(buf)) {
        fprintf(stderr, "Buffer overflow");
        return -1;
    }

    printf("%*s", (int)sizeof(buf), buf);
    return 0;
}

print_number(2);
print_number(10000);
```

stdio printing and scanning

- You used these functions already
- Mostly people forget, that
  - printf returns number of the written symbols
  - scanf returns number of the read tokens.
    - Always check this value.
  - snprintf returns number of the symbols, that would be written to the buffer, it it's big enough
- **C++/C:** people use itoa (originally appeared in K&R as an example) instead of printf.
  - Don't use it.
  - itoa is non-standard function, it's present mostly in Windows compilers
- **C++/C:** Sometimes people use atoi from stdlib.h
  - Don't use it.
  - It doesn't contain restrictions for the buffer size
  - It also doesn't check for errors

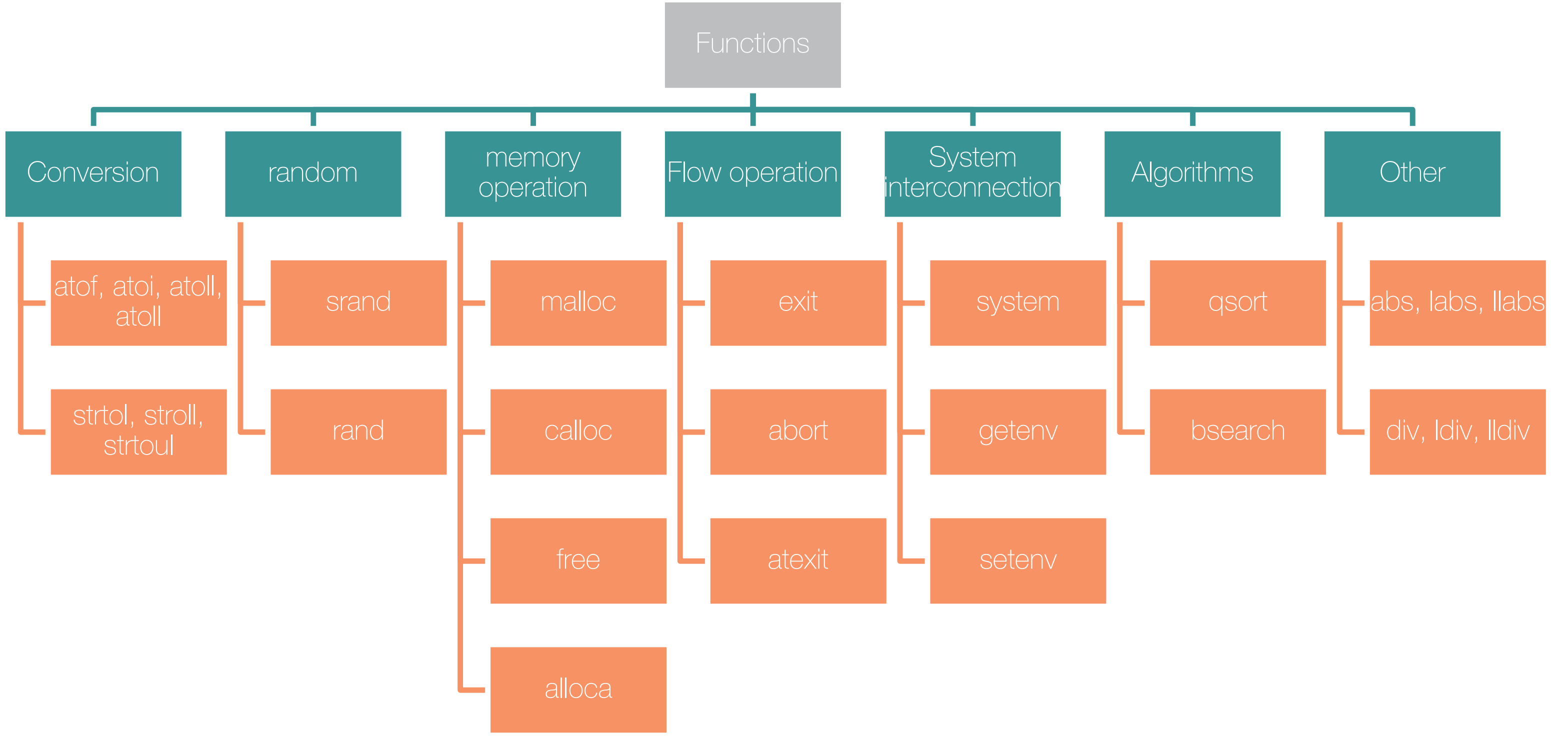
```
int printf(const char * restrict format, ...);

int scanf(const char * restrict format, ...);

int snprintf(char * restrict s, size_t n,
const char * restrict format, ...);

int sprintf(char * restrict s,
const char * restrict format, ...);

int sscanf(const char * restrict s,
const char * restrict format, ...);
```

stdio file operations

- files can be opened, read, written and closed with stdio.h file functions
- FILE * - a pointer to hidden structure
- Always close the file after writing
- Reading in text and binary formats returns different results because of compiler-defined handling of non-printable and whitespace characters

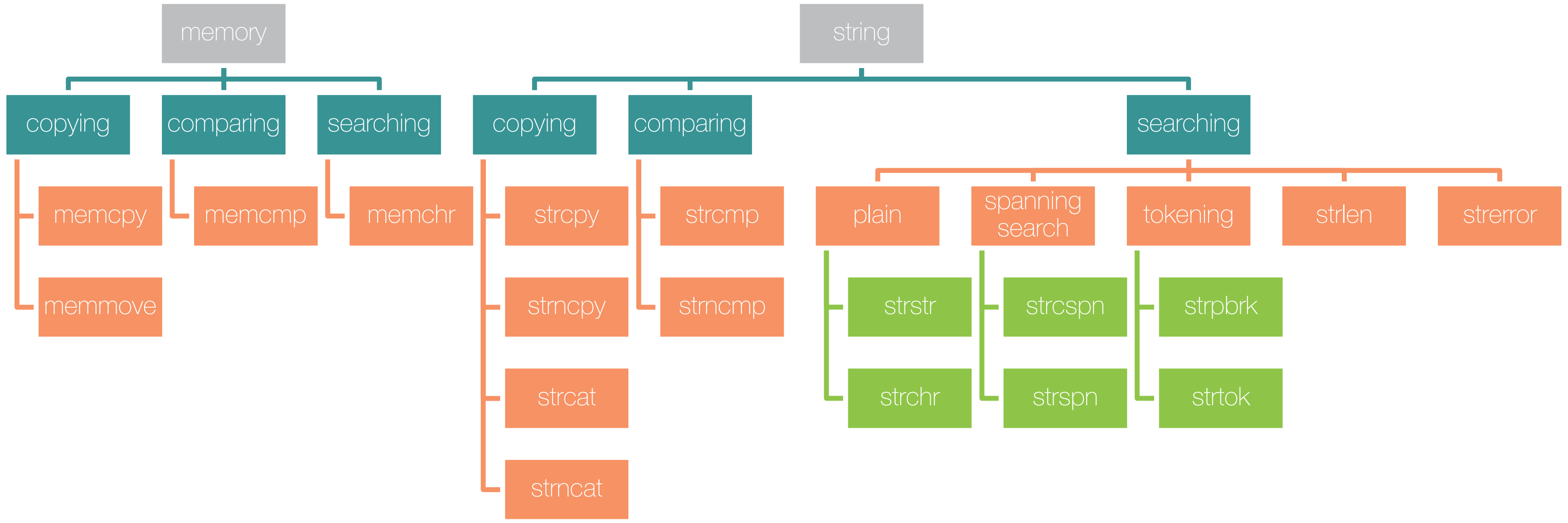
```
char buf[256];

FILE *f = fopen(“/etc/inittab”, “rt”);

fgets(buf, sizeof(buf), f);
printf(“%*s”, (int)sizeof(buf), buf);

fclose(f);
```

<stlib.h>


Functions
Conversion
atof, atoi, atoll, atoll
strtol, stroll, strtoul
random
srand
rand
memory operation
malloc
calloc
free
alloca
Flow operation
exit
abort
atexit
System interconnection
system
getenv
setenv
Algorithms
qsort
bsearch
Other
abs, labs, llabs
div, ldiv, lldiv

GlobalLogic

<string.h>

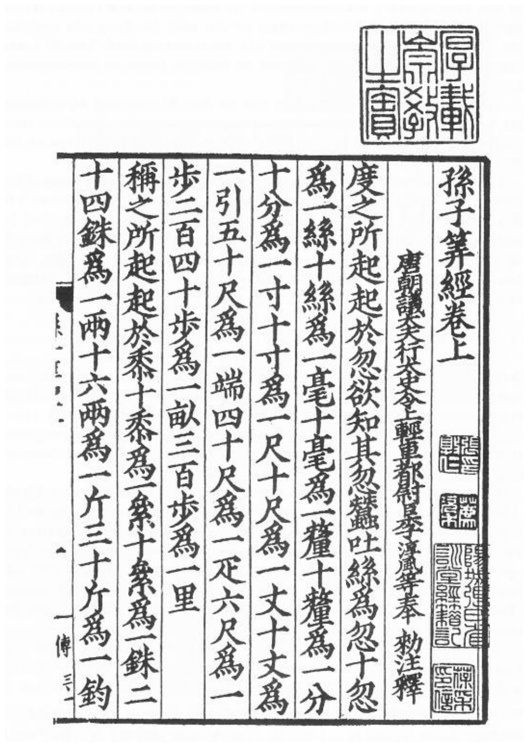


GlobalLogic

Calculus

Numeral systems

- Number itself doesn't depend on the numerical system and can be represented in different ways
- We are interested only in positional numeral systems
    - Historically there were other numeral systems, mostly non-positional, which are out of the scope
- What we are using usually is base-10 system
- Base-10 Hindu-Arabic system is probably of Chinese origin, and it was mainly graphical. It is successfully adopted by other cultures mainly because of the ease of arithmetic operations (comparing to non-positional systems)


孫子算經卷上
唐朝議大夫行太史令上輕車都尉臣李淳風等奉 勅注釋
度之所起起於忽欲知其忽蠶吐絲爲忽十忽
爲一絲十絲爲一毫十毫爲一氂十氂爲一分
十分爲一寸十寸爲一尺十尺爲一丈十丈爲
一引五十尺爲一端四十尺爲一疋六尺爲一
步二百四十步爲一畝三百步爲一里
稱之所起起於黍十黍爲一絫十絫爲一銖二
十四銖爲一兩十六兩爲一斤三十斤爲一鈞

Positional systems

- Positional numeral systems have the same theory: position of the digit defines its weight in the number
  - Position is a logarithm of the corresponding weight
- Positional number is the sum of the digits multiplied on their weight
- Non base-10 systems are the same, they have same arithmetic rules
  - non-base-10 numbers can be multiplied with the same tricks
  - we actually use ancient Chinese way of multiplication studied in the school

Position	3	2	1	0	-1	-2	…
Weight	b^3	b^2	b^1	b^0	b^{-1}	b^{-2}	…
Digit	a_3	a_2	a_1	a_0	a_{-1}	a_{-2}	…
Decimal weight	1000	100	10	1	0.1	0.01	…
Decimal digit	4	3	2	7	0	0	…

Binary system

- Binary system is very common for the hardware implementation
  - It's specifically useful in low level, as you have a direct mapping from 0/1 to different physical levels: high/low voltage, light is turned on/off
- Binary system if very long for human perception
- It's hard to do arithmetic operation in binary system, as you have to carry digits a lot
- Curious fact: binary system is not the most effective, but it's very close to it
  - The most effective radix for the information have to be *e* (see Information Entropy), but it's hard to operate with e-base numeric system

Position	3	2	1	0	-1	-2	…
Weight	b^3	b^2	b^1	b^0	b^{-1}	b^{-2}	…
Digit	a_3	a_2	a_1	a_0	a_{-1}	a_{-2}	…
Binary weight	8	4	2	1	0.5	0.25	…
Binary digit	1	0	1	1	0	0	…

Positional system sum comparison

Decimal weight	10000	1000	100	10	1	0
Number A	0	8	3	6	2	0
Number B	0	1	9	3	0	1
Sum	1	0	2	9	2	1

Binary weight	32	16	8	4	2	1	0
Number A	0	1	1	1	0	1	0
Number B	0	0	1	0	0	0	1
Sum	1	0	0	1	0	1	1

Hexadecimal system

- Hex is another useful numerical system
- It’s slightly more compact, than decimal system, and it’s easier for perception after some habit
  - 0xffffffff’ffffffff has 16 digits, while corresponding decimal has 20 digits
- As any other positional system it has the same rules as a base-10 or base-2 systems
- But the most important fact is that 16 is an exponent of 2, which makes conversion between them really easy

Position	3	2	1	0	-1	-2	…
Weight	b^3	b^2	b^1	b^0	b^{-1}	b^{-2}	…
Digit	a_3	a_2	a_1	a_0	a_{-1}	a_{-2}	…
Hex weight	1024	256	16	1	0.5	0.25	…
Hex digit	1	A	5	9	0	B	…

Numeral conversions

- In order to convert number to another system you need to present it as a sum of allowed digits in this numeric system with a corresponding weight
- Here should be example on the whiteboard

base-10	base-2	base-16
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	a
11	1011	b
12	1100	c
13	1101	d
14	1110	e
15	1111	f

2-base system conversions

- If radix of numeric system can be easily interpreted as an exponent of 2, than conversion between these systems is very simple
- General rule is the following:
  - Remember the representation of hex numbers in binary system
  - Start at the rightmost beginning of the number
  - Unite digits by the logarithm of radix with base 2
    - i.e. for base-2 => base-16 conversion, group them by 4 digits
    - convert these digits by groups
  - PROFIT!
- That’s why hex system is usually used for human representation of data saved in a binary system

1	0	0	1	1	1	1	0	1	0	0	1	0	0	0	0	0	1	0	1
9				e				9				0				5			

Usecase: octal system for file permissions

- POSIX file permissions are grouped by the target groups, for example
  - file can be read, written and executed by its owner
  - file can be read and executed by the user group, owner belongs
  - file can be read and executed by other users
- Such rights can be written in binary system
- While binary system is hard to read and write, octal system is much easier
  - Additionally octal system uses lesser digits, than decimal
  - And it can be directly converted from binary
- `chmod 755 myfile`

	user			group			others		
	r	w	x	r	w	x	r	w	x
permi-ssions	1	1	1	1	0	1	1	0	1
octal	7			5			5		

Logic and bitwise operations

- There are three binary logical operations:
  - AND: && - logical, & - binary
  - OR: || - logical, | - binary
  - XOR: ^ - binary
- There is one unary logical operation:
  - NOT: ! – logical, ~ - binary
- Tables to the right are called truth tables
- Pay attention to the operation you are using:
  - Logic: 0x2 && 0x1 == true && true == true
  - Bitwise: 0x2 & 0x1 == 0x0 == false

		AND
0	0	0
0	1	0
1	0	0
1	1	1

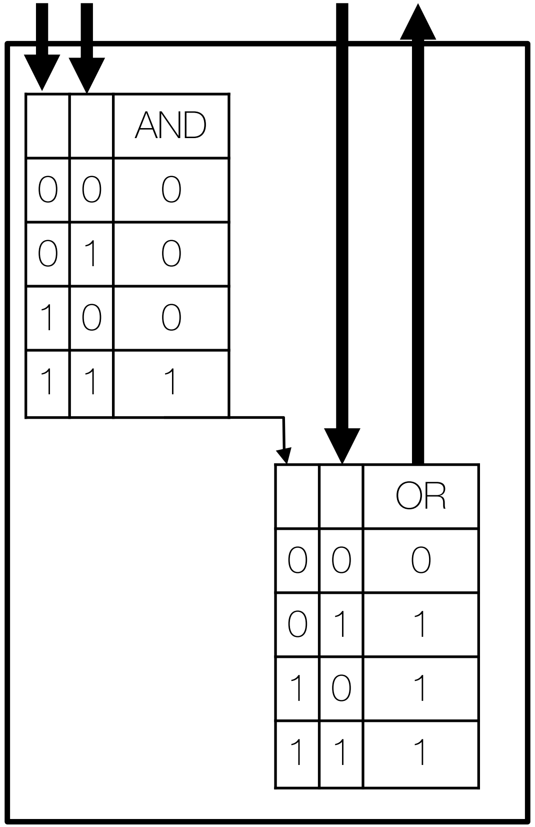
		OR
0	0	0
0	1	1
1	0	1
1	1	1

		XOR
0	0	0
0	1	1
1	0	1
1	1	0

	NOT
0	1
1	0

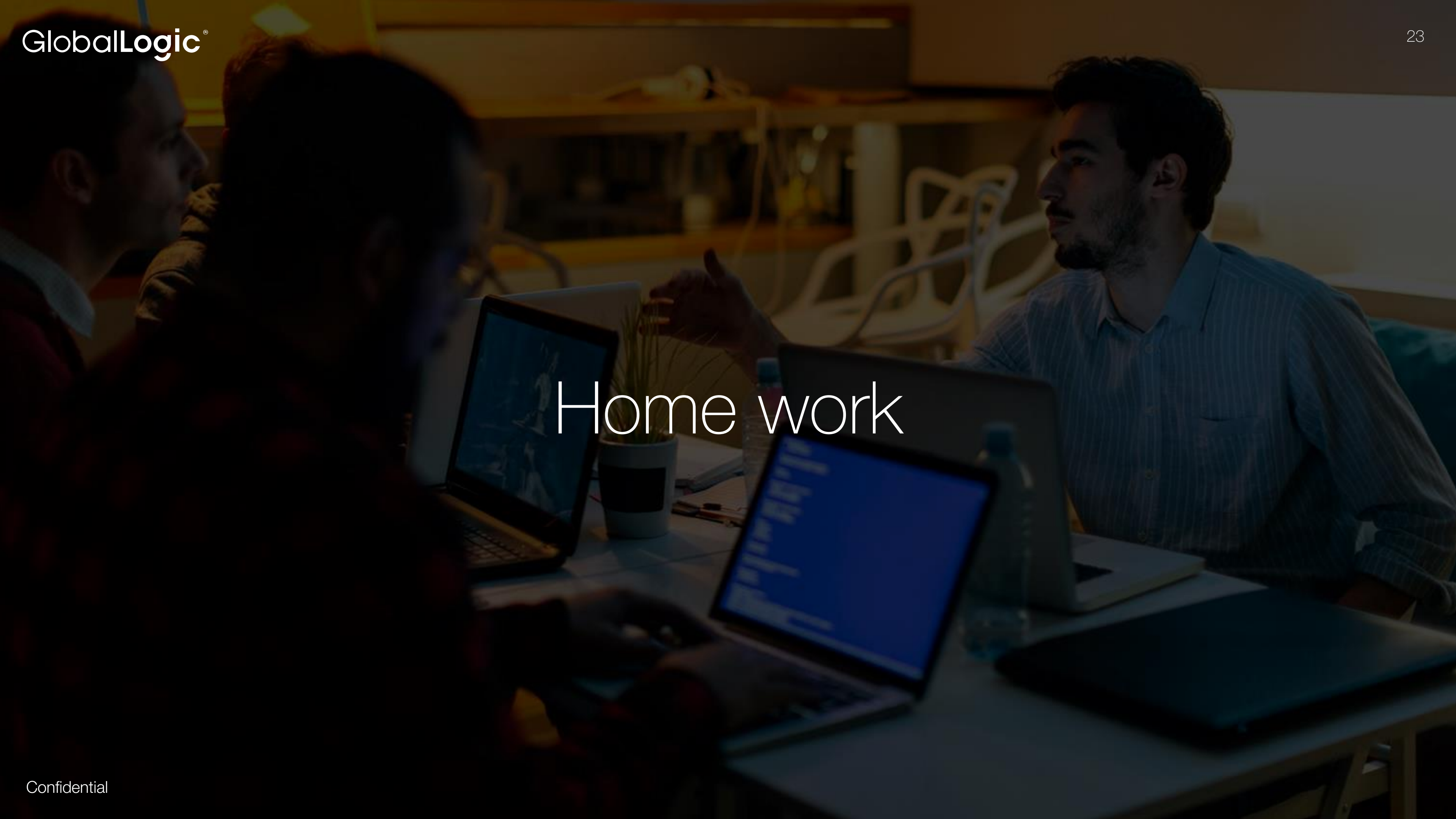
Usecase: FPGA

- Any complex logic expression can be evaluated as a truth table with various inputs number
- Any binary arithmetic can be represented as a logical expression, therefore it has truth table as well.
  - Check the (A & B) | C example to the right
- Truth tables can be encoded in the hardware, so it will evaluate any expression in very low operations (custom math)
- Arrays of such logic can be built as a separate module



		AND
0	0	0
0	1	0
1	0	0
1	1	1

		OR
0	0	0
0	1	1
1	0	1
1	1	1



Home work

Quiz

Fill in the blanks in each of the following statements:

a) The bases of the decimal, binary, octal and hexadecimal number systems are

______,______,______ and ______ respectively.

b) The positional value of the rightmost digit of any number in either binary, octal, decimal or hexadecimal is always

______.

c) The positional value of the digit to the left of the rightmost digit of any number in binary, octal, decimal or hexadecimal is always equal to ______.

State whether each of the following is *true* or *false*. If *false*, explain why.

a) A popular reason for using the decimal number system is that it forms a convenient notation for abbreviating binary numbers simply by substituting one decimal digit per group of four binary bits.

b) The highest digit in any base is one more than the base.

c) The lowest digit in any base is one less than the base.

Fill in the missing values in this chart of positional values for the rightmost four positions in each of the indicated number systems:

decimal	1000	100	10	1
hexadecimal	…	256	…	…
binary	…	…	…	…
octal	512	…	8	…

Do some conversion (manually, the purpose of this question is to practice in manual conversion)

- Convert binary 110101011000 to octal and to hexadecimal.
- Convert hexadecimal FADE to binary.
- Convert octal 7316 to binary.
- Convert hexadecimal 4BEF to octal. [Hint: First convert 4BEF to binary, then convert that binary number to octal.]
- Convert binary 1101010 to decimal.
- Convert octal 314 to decimal.
- Convert hexadecimal FED5 to decimal.
- Convert decimal 135 to binary, to octal and to hexadecimal.

Home task

1. Create a function that will test if bit with given number is set (i.e. equal to 1) in specified value. Omit any error checking (e.g. given bit number less/greather than max possible for given value).

2. Create a function that takes two parameters: val and mask and clears all bits in val which are set in mask.

3. In IP networking host (e.g. your PC) uses network address (na) and network mask (nm) from it's routing table to determine if given address (ip) is part of network or not. All zero bits in network mask are zero in network address too. Host applies some bitwise operation (you need to guess) to ip and nm in order to get network addres of this specific ip and if it equals to na then ip is part of network.

Implement function that takes IP address ip, network address na, network mask nm and returns 1 (one) if ip is part of network represented by na/nm and 0 (zero) otherwise.

For instance ip == 192.168.0.1, na == 192.0.0.0, nm == 255.0.0.0 should return 1; while ip == 10.0.0.1 with same na and nm should return 0.

Extend check for wrong na, when some of the bits in na are 1, while they are 0 in network mask nm; use xor instead of compare. For instance with arguments ip == 192.168.0.1, na == 192.168.1.1, nm == 255.255.0.0 should return 1 (one) and report for incorrect na.

GlobalLogic

Clean code is important

C doesn’t restricts you on the way you put spaces and the way you names the identifiers.

If the code is supposed to be read at least twice, you have to write it clean, especially if the code will be shared