Structured programs – Arithmetic types

Basics of Programming 1

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

G. Horváth, A.B. Nagy, Z. Zsóka, P. Fiala, A. Vitéz

September 13, 2023

Contents

Chapter 1

Structured programming in C

Sequence in C

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Forming a sequence is listing instructions one after eachother

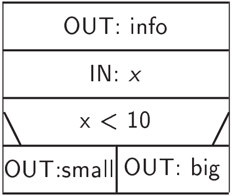
```c
/* football.c -- football fans */
#include <stdio.h>
int main()
{
  printf("Are you");   /* no new line here */
  printf(" blind?\n"); /* here is new line */
  printf("Go Bayern, go!");
  return 0;
}
```

link

```
Are you blind?
Go Bayern, go!
```

Selection control in C – the `if` statement

Let's write a program, that decides if the inputted integer number is small (< 10) or big (≥ 10)!




```
Let x be an integer
OUT: info
IN: x
IF x < 10
  OUT: small
OTHERWISE
  OUT: big
```

```c
#include <stdio.h>
int main()
{
  int x;
  printf("Please enter a number: ");
  scanf("%d", &x);
  if (x < 10)
/* condition */
    printf("small"); /*true branch*/
  else
    printf("big"); /*false branch*/
  return 0;
}
```

link

```
Please give an integer number: 5
small
```

Selection control – the `if` statement

Syntax of the `if` statement

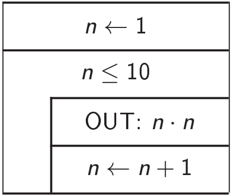
```
if (<condition expression>) <statement if true>
[ else <statement if false> ]opt
```

```c
if (x < 10)           /* condition */
  printf("small"); /* true branch */
else
  printf("big"); /* false branch */
```

```c
if (a < 0)        /* creating absolute value */
  a = -a;
/* no false branch */
```

Top-test loop in C – the while statement

Let's print the square of the integer numbers between 1 and 10!




```
Let n be an integer
n ← 1
WHILE n <= 10
  OUT: n*n
  n ← n+1
```

```c
#include <stdio.h>
int main()
{
  int n;
  n = 1; /* initialization */
  while (n <= 10) /* condition */
  {
    printf("%d ", n*n);/* printing */
    n = n+1; /* increment */
  }
  return 0;
}
```

link

```
1 4 9 16 25 36 49 64 81 100
```

Top-testing loop – the while statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the while statement

```
while (<condition expression>) <instruction>
```

- If <instruction> is a sequence, we enclose it in a {block}:

```
while (n <= 10)
{
  printf("%d ", n*n);
  n = n+1;
}
```

- In language C an instruction always can be replaced with a block.

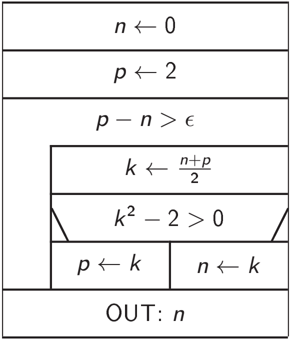
A complex application

- By using sequence, loop and selection, we can construct everything!
- We know enough to construct the algorithm of finding the zeros in C!
- A new element: a type for storing real numbers is called `double` type (to be learned later)

```
double a;           /* the real number */
a = 2.0;            /* assignement of value */
printf("%f", a);    /* printing */
```

Finding zero of a function

We are searching the zeros of function $f(x) = x^2 - 2$, between points $n = 0$ and $p = 2$, with $\epsilon = 0{,}001$ accuracy.




```c
#include <stdio.h>

int main()
{
  double n = 0.0, p = 2.0;
  while (p-n > 0.001)
  {
    double k = (n+p)/2.0;
    if (k*k-2.0 > 0.0)
      p = k;
    else
      n = k;
  }
  printf("The zero is: %f", n);

  return 0;
}
```

link

Chapter 2

Other structured elements

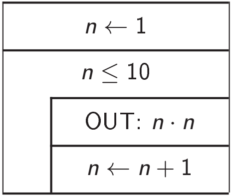
Elements of structured programs

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- We have seen that the structured elements we had learned so far are enough for everything.
- Only for a higher comfort, we introduce new elements, that of course origin from the earlier ones.

Top-test loop in C – the for statement

Let's print the square of the integer numbers between 1 and 10!




```
Let n be an integer
n ← 1
WHILE n <= 10
  OUT: n*n
  n ← n+1
```

Because the structure of

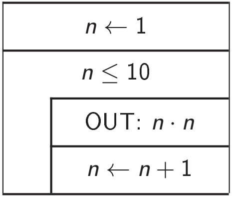
- Initializations
- As long as Condition is TRUE
  - Operation
  - Increment

is very common in programming, we simplify its application with a new statement.

Top-test loop in C – the for statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's print the square of the integer numbers between 1 and 10!




```
Let n be integer
from n=1, WHILE n<=10, one-by-one
  OUT: n*n
```

```c
#include <stdio.h>
int main()
{
  int n;
  for (n = 1; n <= 10; n = n+1)
    printf("%d ", n*n);
  return 0;
}
```

link

```
1 4 9 16 25 36 49 64 81 100
```

Top-test loop in C – the for statement

Syntax of the for statement

```
for (<init exp>; <cond exp>; <post-op exp>)
<instruction>
```

```c
for (n = 1; n <= 10; n = n+1)
  printf("%d ", n*n);
```

- Post-operation is performed after execution of the instruction.

n: 11

```
1 4 9 16 25 36 49 64 81 100
```

Multiplication table

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's print the $10 \cdot 10$ multiplication table!

- We have to print 10 rows (`row = 1, 2, 3, ...10`)
- In every row
  - we print into 10 columns (`col = 1, 2, 3, ...10`)
  - In every column
    - We print the value of `row*col`
  - After this we have to start a new line

```c
int row;
for (row = 1; row <= 10; row=row+1)
{
  int col;      /* declaration at beginning of block */
  for (col = 1; col <= 10; col=col+1)
    printf("%4d", row*col); /* printing with size 4 */
  printf("\n"); /* this is not inside the for */
}
```

link

Multiplication table

- It might be advantageous to enclose in a block even one single instruction, because it might make the code more understandable!

```
int row;
for (row = 1; row <= 10; row=row+1)
{
  int col;      /* declaration at beginning of block */
  for (col = 1; col <= 10; col=col+1)
  {
    printf("%4d", row*col); /* printing with size 4 */
  }
  printf("\n");
}
```

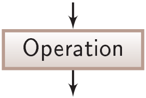
link

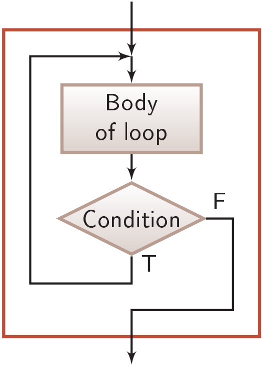
Elements of structured programs

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

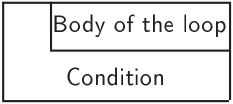
Bottom-test loop

Repetition of an operation as long as a condition is true.





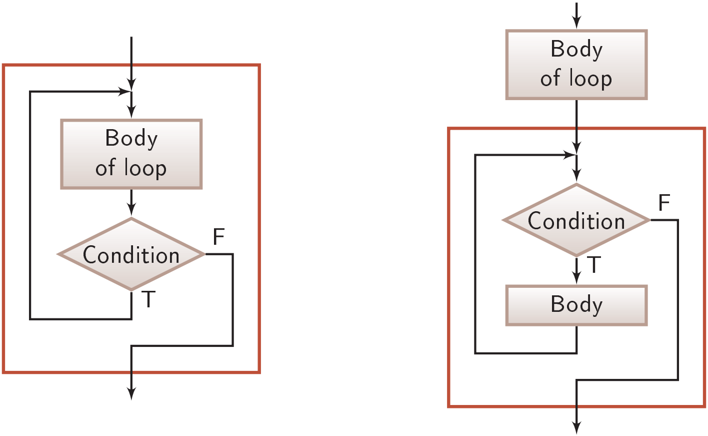

```
REPEAT
  Body of loop
WHILE Condition
```



Elements of structured programs

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

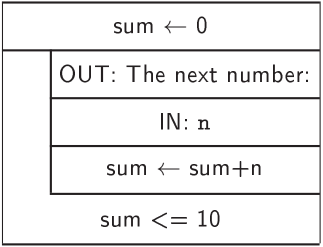
- It can be traced back to sequence and a top-test loop



Bottom-test loop – the do statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's read positive integer numbers! We stop if the sum of the numbers is larger than 10.




```
sum ← 0
REPEAT
  OUT: Info
  IN: n
  sum ← sum+n
WHILE sum ≤ 10
```

```c
#include <stdio.h>
int main()
{
  int sum = 0, n;
  do
  {
    printf("The next number: ");
    scanf("%d", &n);
    sum = sum+n;
  }
  while (sum <= 10);
  return 0;
}
```

link

Bottom-test loop – the do statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the do statement

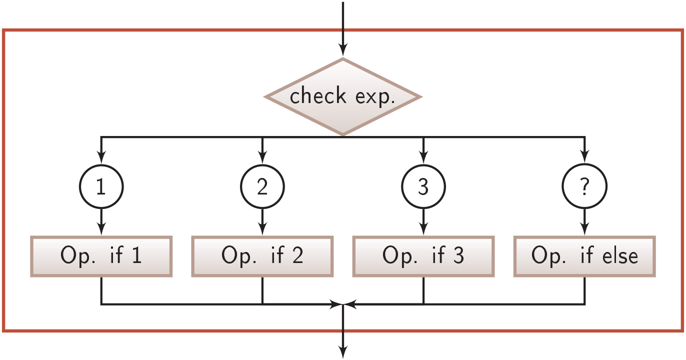
```
do <instruction> while (<condition expression>);
```

```
do
{
  printf("The next number: ");
  scanf("%d", &n);
  sum = sum+n;
}
while (sum <= 10);
```

Elements of structured programs

Integer-value based selection

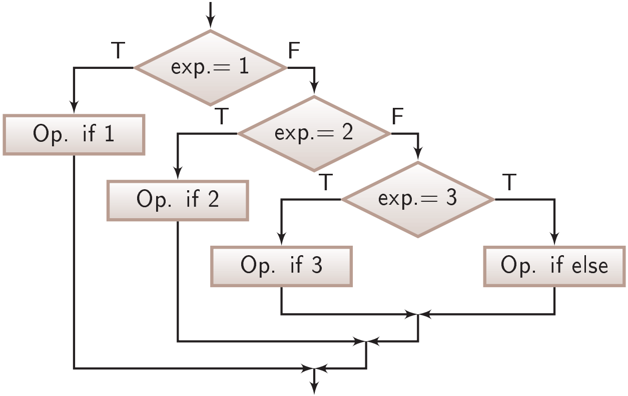
Execution of operations depending on the value of an integer expression



Elements of structured programs

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- It can be constructed as nested selections



Integer-value based selection – the `switch` statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Let's assign (connect) written evaluations to grades given in numbers!

OUT: info					
IN: n					
$n = ?$					
1	2	3	4	5	other
OUT: failed	OUT: poor	OUT: average	OUT: good	OUT: perfect	OUT: something wrong

Integer-value based selection – the switch statement

- Let's assign (connect) written evaluations to grades given in numbers!

```c
#include <stdio.h>
int main() {
  int n;
  printf("Please enter the grade: ");
  scanf("%d", &n);
  switch (n)
  {
    case 1: printf("failed"); break;
    case 2: printf("poor"); break;
    case 3: printf("average"); break;
    case 4: printf("good"); break;
    case 5: printf("perfect"); break;
    default: printf("something wrong");
  }
  return 0;
}
```

link

Integer-value based selection – the switch statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Syntax of the switch statement

```
switch(<integer expression>) {
  case <constant exp1>:  <instruction 1>
  [case <constant exp2>:  <instruction 2> ...]opt
  [default:  <default instruction> ]opt
}
```

```c
switch (n)
{
  case 1: printf("failed"); break;
  case 2: printf("poor"); break;
  case 3: printf("average"); break;
  case 4: printf("good"); break;
  case 5: printf("perfect"); break;
  default: printf("something wrong");
}
```

Integer-value based selection – the switch statement

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- The break instructions are not part of the syntax. If we omit them, the switch will remain syntactically correct, but it will not provide the same result as before:

```
switch (n)
{
  case 1: printf("failed");
  case 2: printf("poor");
  case 3: printf("average");
  case 4: printf("good");
  case 5: printf("perfect");
  default: printf("something wrong");
}
```

link

```
Please enter the grade: 2
pooraveragegoodperfectsomething wrong
```

Integer-value based selection – the switch statement

- The constant expressions are only entry points, and from this point on, all instructions are executed until the first break or until the enf of the block:

```
switch (n)
{
  case 1: printf("failed"); break;
  case 2:
  case 3:
  case 4:
  case 5: printf("passed"); break;
  default: printf("something wrong");
}
```

link

```
Please enter the grade: 2
passed
```

Chapter 3

Arithmetic types of C

Types – Introduction

Type is

- Set of values
- Operations
- Representation

In a real computer – the set of values is limited

- We can not represent arbitrary large numbers
- We can not represent numbers with arbitrary accuracy
  $\pi \neq 3.141592654$
- We must know the limits of what can be represented, in order to store our data
  - without any loss of information or
  - with an acceptable level of information loss, without wasting memory

Types of C language

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- void
- scalar
  - arithmetic
    - integer: integer, character, enumerated
    - floating-point
  - pointer
- function
- union
- compound
  - array
  - structure
- Today we will learn about them

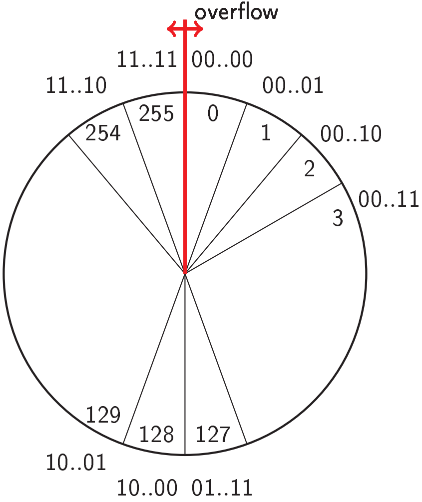
Binary representation of integers

- Binary representation of unsigned integers stored in 8 bits

dec	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	hex
0	0	0	0	0	0	0	0	0	0x00
1	0	0	0	0	0	0	0	1	0x01
2	0	0	0	0	0	0	1	0	0x02
3	0	0	0	0	0	0	1	1	0x03
⋮	⋮							⋮	⋮
127	0	1	1	1	1	1	1	1	0x7F
128	1	0	0	0	0	0	0	0	0x80
129	1	0	0	0	0	0	0	1	0x81
⋮	⋮							⋮	⋮
253	1	1	1	1	1	1	0	1	0xFD
254	1	1	1	1	1	1	1	0	0xFE
255	1	1	1	1	1	1	1	1	0xFF

The overflow

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES



- In case of unsigned integers stored in 8 bits
  - 255+1 = 0
  - 255+2 = 1
  - 2–3 = 255
- ”modulo 256 arithmetic”
  - We always see the remainder of the result divided by 256

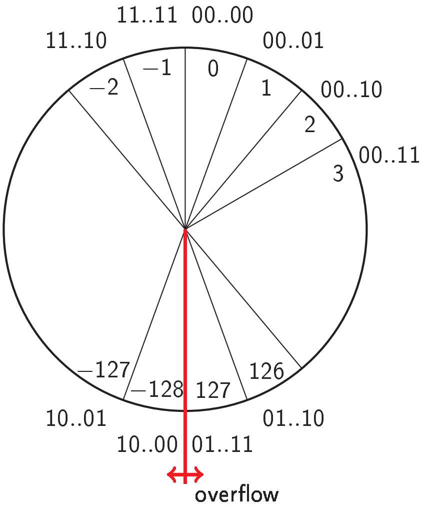
Two's complement representation of integers

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Two's complement representation of signed integers stored in 8 bits

dec	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	hex
0	0	0	0	0	0	0	0	0	0x00
1	0	0	0	0	0	0	0	1	0x01
2	0	0	0	0	0	0	1	0	0x02
3	0	0	0	0	0	0	1	1	0x03
⋮	⋮							⋮	⋮
127	0	1	1	1	1	1	1	1	0x7F
−128	1	0	0	0	0	0	0	0	0x80
−127	1	0	0	0	0	0	0	1	0x81
⋮	⋮							⋮	⋮
−3	1	1	1	1	1	1	0	1	0xFD
−2	1	1	1	1	1	1	1	0	0xFE
−1	1	1	1	1	1	1	1	1	0xFF

The overflow



- In case of signed integers stored in 8 bits
  - 127+1 = -128
  - 127+2 = -127
  - -127-3 = 126
- on the other hand
  - 2-3 = -1

Integer types in C

type	bit[1]	<limits.h>		printf
signed char	8	CHAR_MIN	CHAR_MAX	%hhd[2]
unsigned char	8	0	UCHAR_MAX	%hhu[2]
signed short int	16	SHRT_MIN	SHRT_MAX	%hd
unsigned short int	16	0	USHRT_MAX	%hu
signed int	32	INT_MIN	INT_MAX	%d
unsinged int	32	0	UINT_MAX	%u
signed long int	32	LONG_MIN	LONG_MAX	%ld
unsigned long int	32	0	ULONG_MAX	%lu
signed long long int[2]	64	LLONG_MIN	LLONG_MAX	%lld
unsigned long long int[2]	64	0	ULLONG_MAX	%llu

[1] Typical values, the standard only determines the minimum

[2] since the C99 standard

Declaration of integers

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Defaults
  - The signed sign-specifier can be omitted

```
int i;            /* signed int */
long int l;       /* signed long int */
```

  - If there is sign- or length-modifier, the int can be omitted.

```
unsigned u;          /* unsigned int */
short s;             /* signed short int */
```

Integer types

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- An example on how to use the previous table: a program that runs for a very long time[3]

```c
#include <limits.h> /* for integer limits */
#include <stdio.h>  /* for printf */

int main(void)
{ /* almost all long long int */
  long long i;

  for (i = LLONG_MIN; i < LLONG_MAX; i = i+1)
    printf("%lld\n", i);

  return 0;
}
```

link

[3] provided that `long long int` is 64 bit long, the program runs for 585 000 years if the computer prints 1 million numbers per second

Integer constants

- Specifying integer constants

```
int i1=0, i2=123, i4=-33;              /* decimal */
int o1=012, o2=01234567;               /* octal */
int h1=0x1a, h2=0x7fff, h3=0xAa1B      /* hexadecimal */

long l1=0x1al, l2=-33L;                /* l or L */

unsigned u1=33u, u2=45U;               /* u or U */
unsigned long ul1=33uL, ul2=1231U;     /* l and u */
```

- If neither u or l is specified, the first type that is big enough is taken:
  1. `int`
  2. `unsigned int` – in case of hexa and octal constants
  3. `long`
  4. `unsigned long`

Why do we need to know the limits of number representations?

Let's determine the following value!

$$\binom{15}{12} = \frac{15!}{12! \cdot (15-12)!}$$

(What is the number of possibilities of selecting 12 out of 15 different chocolates?)

- The value of the numerator is $15! = 1\ 307\ 674\ 368\ 000$
- The value of the denominator is $12! \cdot 3! = 2\ 874\ 009\ 600$
- None of them can be represented as a 32 bits `int`!
- But with simplifying the expression

  $$\frac{15 \cdot 14 \cdot 13}{3 \cdot 2 \cdot 1} = \frac{2730}{6} = 455$$

  all parts can be calculated without any problem, even on 12 bits.

Floating-point types

- Normal form

$$23.2457 = (-1)^{0} \cdot 2.3245700 \cdot 10^{+001}$$

$$-0.001822326 = (-1)^{1} \cdot 1.8223260 \cdot 10^{-003}$$

Representation of the normal form

- Floating-point fractional = sign bit + mantissa + exponent
  1. sign bit: 0–positive, 1–negative
  2. mantissa: unsigned integer (without the decimal comma), because of normalization, the first digit is ≥ 1
  3. exponent (or order, characteristic): signed integer

Floating-point types

- Binary normal form

$$5.0 = 1.25 \cdot 4 = (-1)^{0} \cdot 1.0100_{\mathrm{b}} \cdot 2^{010_{\mathrm{b}}}$$

0	**0100**	**010**

Representation of binary normal form

- Floating-point fractional = sign bit + mantissa + exponent
  1. sign bit: 0–positive, 1–negative
  2. mantissa: unsigned integer (without the **binary comma**), because of normalization, the first digit is = 1, so we don't store it[4].
  3. exponent: signed integer

[4] the leading bit is implicit

Floating-point types in C

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Floating-point types of C

type	typical values bits	mantissa	exponent	printf/scanf
`float`	32 bits	23 bits	8 bits	`%f`
`double`	64 bits	52 bits	11 bits	`%f`/`%lf`
`long double`	128 bits	112 bits	15 bits	`%Lf`

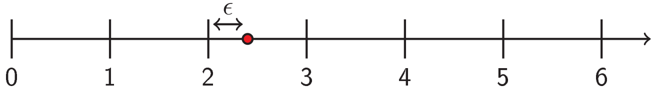
- Floating-point constants

```
float       f1=12.3f , f2=12.F , f3=.5f , f4=1.2e-3F ;
double      d1=12.3  , d2=12.  , d3=.5  , d4=1.2e-3  ;
long double l1=12.3l , l2=12.L , l3=.5l , l4=1.2e-3L ;
```

- In C we use decimal point and not a comma!

Representation accuracy of integer types

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

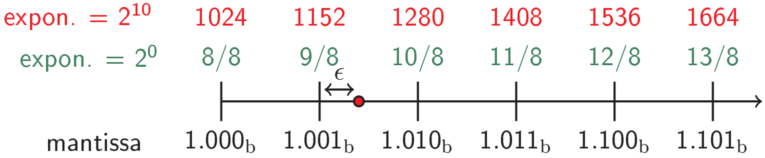


Absolute accuracy of number representation

It is the maximal ϵ error of representing an arbitrary real number with the closest integer

- The absolute accuracy of representing with integer types is 0.5

Representation accuracy of floating-point numbers



- in this example
  - The (absolute) representation accuracy of the mantissa is $1/16$
  - If the exponent is 2^0, the representation accuracy is $1/16$
  - If the exponent is 2^{10}, the representation accuracy is $2^{10}/16 = 64$
- There is no absolute, only relative accuracy, that is, in this present case, 3 bits.

Consequences of finite number representation

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- As the floating-point number representation is not accurate, we must not check the equality of results of operations!

$$\frac{22}{7} + \frac{3}{7} \neq \frac{25}{7}$$

instead

$$\left|\frac{22}{7} + \frac{3}{7} - \frac{25}{7}\right| < \varepsilon$$

- The exponent will magnify the rounding error of the finite long mantissa, thus the large numbers are much less accurate than small numbers. The errors of the large numbers can "eat up" the small ones:

$$A + a - A \neq a$$

Consequences of the binary representation of numbers

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- A decimal finite number might not be finite in binary form, eg.:

$$0{,}1_{\mathrm{d}} = 0{,}0\overline{0011}_{\mathrm{b}}$$

- How many times will be this cycle repeated?

```
double d;
for (d = 0.0; d < 1.0; d = d+0.1) /* 10? 11? */
{
  ...
}
```

- The good solution is:

```
double d;
double eps = 1e-3; /* what is the right eps for here? */
for (d = 0.0; d < 1.0-eps; d = d+0.1) /* 10 times */
{
  ...
}
```

Thank you for your attention.