Introduction to Programming and Computational Physics

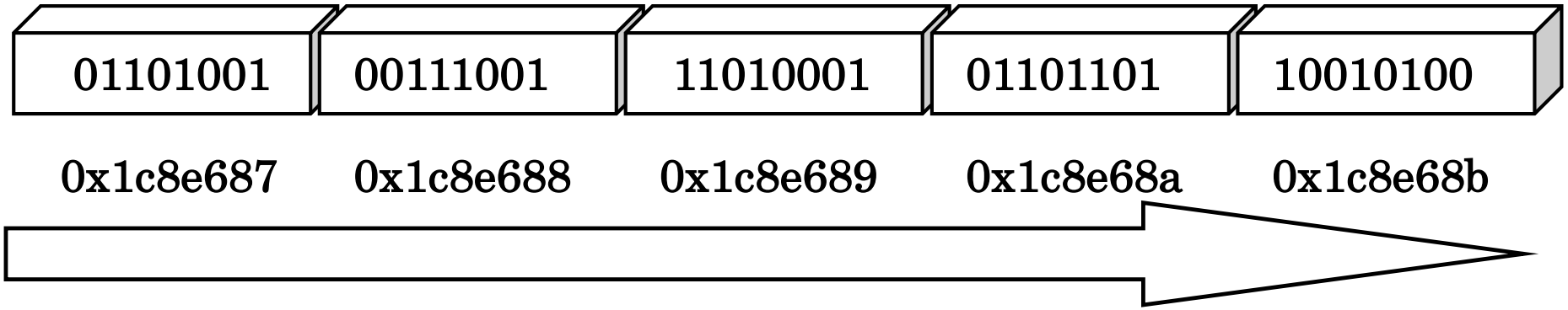
Lecture 2

Variables and data types
Arithmetic operators
Functions

Variables and data types

The memory is a sequence of contiguous cells, called **byte**, each byte being composed by 8 **bit** (possible values: 0 or 1).

Each cell has an own **address.**

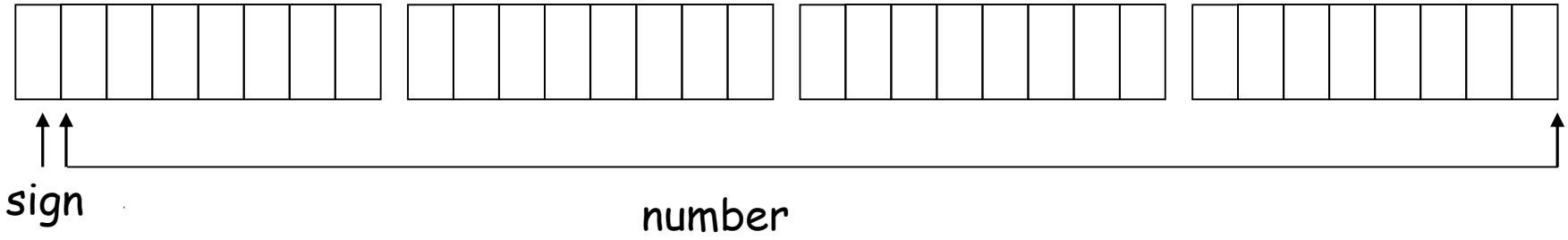


Declaring a *variable* means associating a purely symbolic name called variable to a memory location. In order to establish how much memory space should be allocated for the variable we need to specify its *type*.

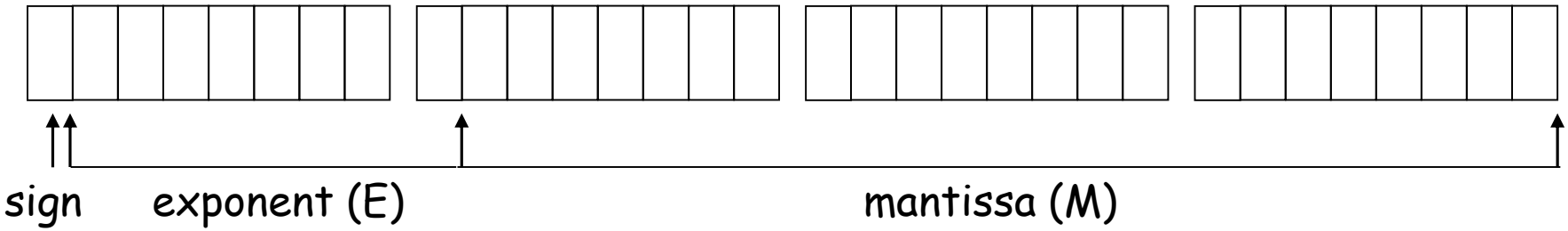
Numbers representation in computer science

There are many representations:

integer

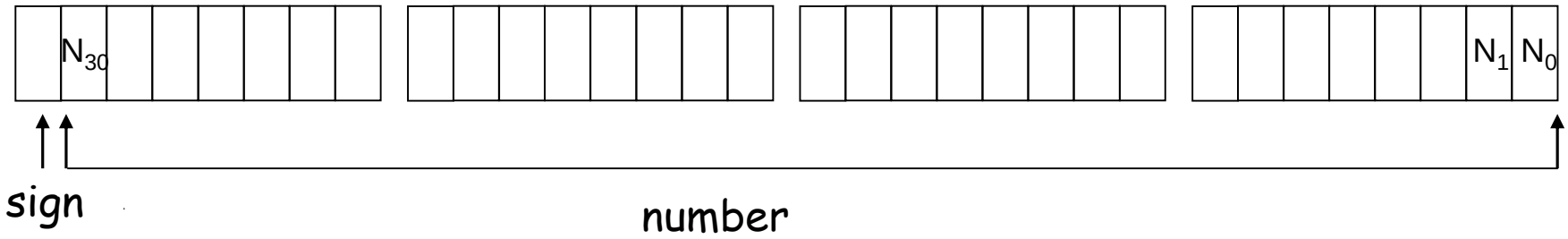


floating point $(-1)^S \times M \times 2^E$ M takes values between 1 and 2



If you read an integer as a float or the opposite you will get a meaningless result !!!

4 bytes integer type



<u>if sign is 0 the number will be non-negative</u> and its value will be

$$N = \begin{pmatrix} 2^{30} \\ 0 \end{pmatrix} \begin{matrix} N_{30}=1 \\ N_{30}=0 \end{matrix} + \ldots + \begin{pmatrix} 2^{1} \\ 0 \end{pmatrix} \begin{matrix} N_{1}=1 \\ N_{1}=0 \end{matrix} + \begin{pmatrix} 2^{0} \\ 0 \end{pmatrix} \begin{matrix} N_{0}=1 \\ N_{0}=0 \end{matrix}$$

when all the bit are 1 we get the maximum value 2147483647

<u>if sign is 1 the number will be negative</u>

If all the bits ($N_{30},\ldots,N_0$) are zero we have the min value -2147483648 otherwise just replace “N” with “min. value + N” in the given formula

If you add 1 to the value 2147483647 you will get the result -2147483648 …be careful…

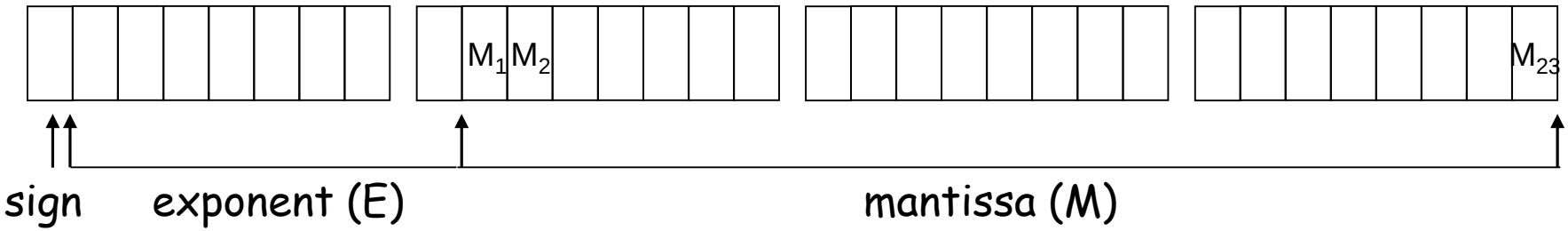
Data types in C: integer

Integer types		(size for gcc compiler)
int	integer with sign	(4 bytes)
short	integer with sign	(2 bytes)
long (long long)	integer with sign	(8 bytes)
unsigned int	like int without sign	(4 bytes)
unsigned short	like short without sign	(2 bytes)
unsigned long	like long without sign	(8 bytes)

Character types	
char	a 1 byte integer

The C language doesn't fix size for a given type. The size is instead machine/compiler dependent.

4 bytes floating point representation (IEEE-754)



$$(-1)^S \times M \times 2^E$$

The exponent is *biased* by 127 : if the bit content of the exponent is 10000011 = 131 the value of E will be 131-127 = 4

E is in the range (-126,127), the values -127 and 128 are reserved for special cases

Mantissa M is:

$$1+\begin{pmatrix}\frac{1}{2}\\0\end{matrix}\begin{matrix}M_1=1\\M_1=0\end{matrix} \quad +\begin{pmatrix}\frac{1}{4}\\0\end{matrix}\begin{matrix}M_2=1\\M_2=0\end{matrix} \quad +\ldots+\begin{pmatrix}\frac{1}{2^{23}}\\0\end{matrix}\begin{matrix}M_{23}=1\\M_{23}=0\end{matrix}$$

Precision of floats

As we use 23 bits for the mantissa we will have 2^{23} or ~10^{7}possible values for the mantissa

This means that mantissa spans numbers from 1 to 2 with step ~10^{-7}

The precision of the number is then ~ $2^{E} \times 10^{-7}$

Exponent	Minimum	Maximum	Precision
0	1	1.999999880791	1.19209289551e-7
1	2	3.99999976158	2.38418579102e-7
2	4	7.99999952316	4.76837158203e-7
10	1024	2047.99987793	1.220703125e-4
11	2048	4095.99975586	2.44140625e-4
23	8388608	16777215	1
24	16777216	33554430	2
127	1.7014e38	3.4028e38	2.02824096037e31

This means that operations like $10^{15}+10^{4}$ will give the result 10^{15}

http://www.math.grin.edu/~stone/courses/fundamentals/IEEE-reals.html

http://de.wikipedia.org/wiki/IEEE_754

Data types in C: floating point

Floating point types	(size for gcc compiler)	
float	rational number	(4 bytes)
double	rational number	(8 bytes)
long double	rational number	(16 bytes)

Real numbers in computer science do not exist !!!

double and long double have the same structure of float

double : **11** bits exponent and **52** bits mantissa step ~2×10^{-16}

long double: **15** bits exponent and **112** bits mantissa step ~2×10^{-34}

How to declare and use variables

```
int a; float pi;

double e,f,g;
```

declaration

```
int   input_data_acq_27_09_2007;

input_data_acq_27_09_2007 = 10;
```

Variable name: letters, numbers and _ (underscore)

```
float myvar = 23.601;

int mv1=2,mv2=3,mv3=0;
```

Declaration and value assignment in the same statement

```
int 3a;
```

WRONG: a variable name can't start with a number

```
int printf;
```

WRONG: a variable name can't take the name of a function

scanf

This is another “external” function defined in stdio.h header file

It is used to receive a value from the user

```
int a; float b;

scanf(“%d”,&a);

scanf(“%f”,&b);
```

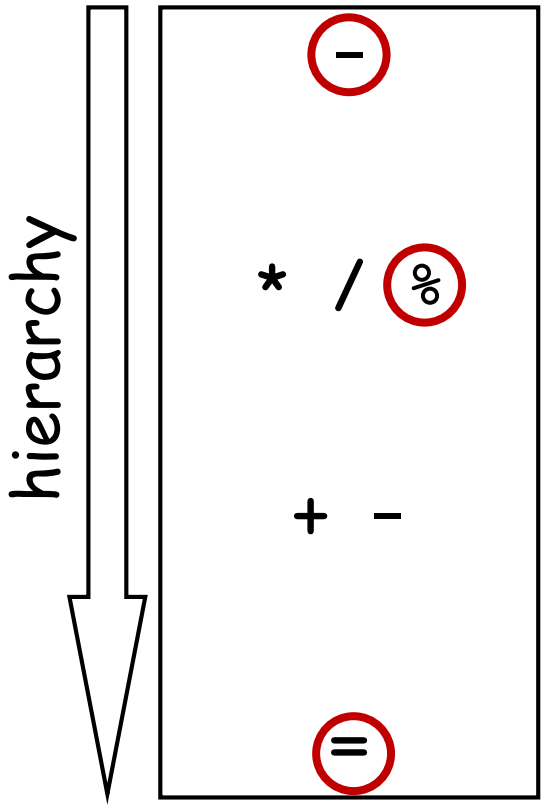
The execution of the program will stop at this point waiting for the input

Variables reading and writing

scanf and **printf** employ the following indicators for reading and writing correctly different data types

Indicators	converted types
%c	**char**
%d	**char, int, short int**
%ld %lld	**long int**
%f	**float**
%lf	**double**
%s	**string: array of char**
%u	**unsigned int, unsigned short int**
%lu	**unsigned long int**

arithmetic operators



negation: if x has value 5 -x is taken as -5

% is modulus (the remainder of the division). It cannot be applied to float or double

Assignment:

x = 5; (x takes value 5)

x = x + 2; (add 2 to the value of x)

The average of 3 numbers

```c
#include <stdio.h>

int main()
{
  float num1 = 0, num2 = 0, num3 = 0;
  float ave = 0;
  printf("\n");
  printf("The first value is? ");
  scanf("%f", &num1);
  printf("The second value is? ");
  scanf("%f", &num2);
  printf("The third value is? ");
  scanf("%f", &num3);
  ave = (num1+num2+num3)/3.0;
  printf("\n");
  printf("The average of %.3f,%.3f,%.3f is %.3f\n\n",num1,num2,num3,ave);

  return 0;
}
```

} variables declaration and initialization

to print only 3 decimal digits

Operations with different data types

If we mix different types in an operation (i.e. int + float) *usually* the lower level type is *promoted* to the higher type, following the scale:

$$\mathrm{char},\ \mathrm{short} \Rightarrow \mathrm{int} \Rightarrow \mathrm{long} \Rightarrow \mathrm{float} \Rightarrow \mathrm{double} \Rightarrow \mathrm{long\ double}$$

This means that if you try to add an int to a float the integer is converted to a float before the operation (and the result will be a float)

The C language allows also to convert a type "on site": if **var1** is an int and **var2** is a float we can write (float)**var1**+**var2** that means: "convert **var1** into a float before performing the operation".

This is called "casting": it is often useful, but sometimes dangerous: if **var3** is a float and we convert it to an int, by writing (int)**var3** then only the integer part is read (truncation) so that 1.99999 becomes 1

N.B: The casting is effective only in the statement where it is operated, it doesn't change permanently the type of the variable !!!

Constants

If we want to give a well defined value to a variable (i.e. we don't want to change it during the program) we may use const

```
const int max_number = 100;

const char yes = 'y';

const float pi = 3.14159265;
```

It can be used with any kind of data type

Inclusion of libraries

In our first C program we wrote:

```
#include <stdio.h>
```

The compiler is requested to insert the content of the stdio.h *header* file in order to have the definition of the `printf` function. Also the function `scanf` is defined in this file.

In order to use function like `sqrt` we will have to include

```
#include <math.h>
```

When math.h is included the file compilation needs an additional -lm:
gcc prog.c -lm -o prog.exe

We can also define our functions and include an header file as:

```
#include "myfunctions.h"
```

In this case, there must be a myfunctions.c file where the functions are implemented. We will come back to this point in future.

How to use functions

In order to use a function you need to look at its *prototype*, which is defined in the corresponding header file.

The most important functions are defined in math.h

```
double sqrt(double x)
```

output input

```
#include <math.h>

int main()

{

   double a;

   a = sqrt(10.5);

   double b = sqrt(a);

   ...

}
```

Some function defined in math.h

`double acos(double x)` -- Compute arc cosine of x.
`double asin(double x)` -- Compute arc sine of x.
`double atan(double x)` -- Compute arc tangent of x.
`double atan2(double y, double x)` -- Compute arc tangent of y/x.
`double ceil(double x)` -- Get smallest integral value that exceeds x.
`double cos(double x)` -- Compute cosine of angle in radians.
`double cosh(double x)` -- Compute the hyperbolic cosine of x.
`div_t div(int number, int denom)` -- Divide one integer by another.
`double exp(double x` -- Compute exponential of x
`double fabs (double x )` -- Compute absolute value of x.
`double floor(double x)` -- Get largest integral value less than x.
`double fmod(double x, double y)` -- Divide x by y with integral quotient and return remainder.
`double frexp(double x, int *expptr)` -- Breaks down x into mantissa and exponent of no.
`labs(long n)` -- Find absolute value of long integer n.
`double ldexp(double x, int exp)` -- Reconstructs x out of mantissa and exponent of two.
`ldiv_t ldiv(long number, long denom)` -- Divide one long integer by another.
`double log(double x)` -- Compute log(x).
`double log10 (double x )` -- Compute log to the base 10 of x.
`double modf(double x, double *intptr)` -- Breaks x into fractional and integer parts.
`double pow (double x, double y)` -- Compute x raised to the power y.
`double sin(double x)` -- Compute sine of angle in radians.
`double sinh(double x)` - Compute the hyperbolic sine of x.
`double sqrt(double x)` -- Compute the square root of x.
`void srand(unsigned seed)` -- Set a new seed for the random number generator (rand).
`double tan(double x)` -- Compute tangent of angle in radians.
`double tanh(double x)` -- Compute the hyperbolic tangent of x.

Don't forget to include it: #include <math.h>