Algorithmics and C basis

Antonio Mucherino

University of Rennes 1

www.antoniomucherino.it

last update: August 2013

What's an algorithm

First example

Make a travel by train

1. **begin**
2. **go** to the train station;
3. **buy** a ticket for your destination;
4. **look for** the platform;
5. **while** the train is not yet at the station: **wait**;
6. **get in** the train;
7. **for** each stop
   - **if** the train is at your destination
     - **get off** the train;
   - **else**
     - **stay in**;
8. **end**

What's an algorithm

Second example

Cook a pasta dish

1. **begin**
2. **put** water in the pot and **make** it boil;
3. **put** pasta in the boiling water;
4. **repeat**
   - **taste** the pasta;
5. **until** the pasta is *al dente*;
6. **remove** the water;
7. **add** some sauce;
8. **end**

Some remarks

If you can answer yes *to the following two questions:*

- Are you able to understand well this algorithm?
- Are you able to perform all these tasks?

then, this algorithm is well-defined.

Control structures:

- **while**,
- **repeat . . . until**,
- **if**,
- **for**.

Some remarks

If you can answer yes *to the following two questions:*

- Are you able to understand well this algorithm?
- Are you able to perform all these tasks?

then, this algorithm is well-defined.

Control structures:

- **while**,
- **repeat . . . until**,
- **if**,
- **for**.

while

The **while** loop is used to execute an instruction or a block of instructions while a given condition is satisfied.

Its general format is:

```
while (condition) do
   instruction(s);
end while
```

where:

- `condition` is a logical condition;
- `instruction(s)` represents the instruction or the block of instructions that are executed while `condition` is satisfied.

repeat ... until

The **repeat** ... **until** loop is used to execute an instruction or a block of instructions until a certain condition is satisfied.

Its general format is:

```
repeat
   instruction(s);
until (condition);
```

where:

- `condition` is a logical condition;
- `instruction(s)` represents the instruction or the block of instructions that are executed until `condition` is satisfied.

Main difference with while:
at least one execution of `instruction(s)` *is performed*.

if

The **if** keyword is used to execute an instruction or a block of instructions only when a certain condition is satisfied.

Its general format is:

```
if (condition) then
   instruction(s) A;
else
   instruction(s) B;
end if
```

where:

- `condition` is a logical condition;
- `A` marks the instructions that are executed if `condition` is true;
- `B` marks the instructions that are executed if `condition` is false.

for

The **for** loop repeats a set of instructions a predetermined number of times. It makes use of an internal counter.

Its general format is:

```
for (initialization; condition; change) do
   instruction(s);
end for
```

where:

- `initialization` defines the first value of the counter;
- `condition` defines the stopping condition for the counter;
- `change` indicates how to modify the counter at each iteration.

Compute the sum of n numbers $(x_1, x_2, \ldots, x_n)$

1. **begin**
2. **define** a new number and name it s;
3. **set** s equal to 0;
4. **for** each number x_i
   - **add** x_i to s;
   - **put** the result in s;
5. **end for**
6. **print** s;
7. **end**

Some remarks

Who can perform the algorithms we are studying?

- Can you easily perform this algorithm?
- Can a computer machine perform this algorithm?

The algorithm must be well-defined for the executer.

Algorithm

the formal definition

An **algorithm** is a finite sequence of well-defined and unambiguous instructions needed for performing a given task.

- finite sequence: infinite sequences are not useful!
- well-defined and unambiguous instructions: all instructions must be clearly stated so that the executer can understand them well;
- performing a given task: algorithms are supposed to solve a given problem.

The previous three are examples of algorithms that can be executed by humans. How to develop an algorithm that can be executed by a computer?

What's an algorithm

Rewriting the third example

Given a vector of n real numbers $(x_1, x_2, \ldots, x_n)$, compute the sum $s = x_1 + x_2 + \cdots + x_n$

1. **begin**
2. **set** $s = 0$; *// initialization of s*
3. **for** $i = 1, n$
   - $s = s + x_i$; *// adding each x_i to s*
4. **end for**
5. **print** s;
6. **end**

The product of n real numbers

version I

Given a vector of n real numbers $(x_1, x_2, \ldots, x_n)$, compute the product $p = x_1 \times x_2 \times \cdots \times x_n$

1. **begin**
2. **set** $p = 1$; *// initialization of p*
3. **for** $i = 1, n$
   - $p = p \times x_i$; *// multiplying the partial p and x_i*
4. **end for**
5. **print** p;
6. **end**

Is this algorithm the best one for computing products of real numbers?

The product of n real numbers

version II

Algorithmics and C basis

A. Mucherino

Introduction
For beginners ...
Definition of algorithm
Examples
The product of real numbers
Maximum element of a vector
C basis
Variables, arrays
Coding ...
Execution
A sorting algorithm
The algorithm
The C code
A little more advanced ...
Pointers
Libraries

Given a vector of n real numbers $(x_1, x_2, \ldots, x_n)$, compute the product $p = x_1 \times x_2 \times \cdots \times x_n$

1. **begin**
2. **set** $p = x_1$; *// initialization of p*
3. **for** $i = 2, n$
   - **if** $x_i \neq 0$ and $p \neq 0$ **then**
     - $p = p \times x_i$; *// multiplying the partial p and x_i*
   - **else**
     - $p = 0$; *// p is 0*
   - **end if**
4. **end for**
5. **print** p;
6. **end**

Again, is this algorithm the best one for computing products of real numbers?

The product of n real numbers

version III

Given a vector of n real numbers $(x_1, x_2, \ldots, x_n)$, compute the product $p = x_1 \times x_2 \times \cdots \times x_n$

1. **begin**
2. **set** $p = x_1$; *// initialization of p*
3. **set** $i = 2$; *// initialization of the counter i*
4. **while** ($i < n$ and $p \neq 0$) **do**
   - $p = p \times x_i$; *// multiplying the partial p and x_i*
   - $i = i + 1$; *// the counter i must be updated*
5. **end while**
6. **print** p;
7. **end**

This is the optimal *algorithm for computing the product of real numbers*

How to find the maximum element of a vector

Given a vector of integer numbers

7	2	1	8	3	4	2	9	0	8

How can find the maximum element contained in the vector?

$$max = ?$$

How to find the maximum element of a vector

Given a vector of integer numbers

7	**2**	1	8	3	4	2	9	0	8

Let us compare the first two elements: 7 is larger than 2.

$$max = ? \qquad partial\_max = ?$$

How to find the maximum element of a vector

Given a vector of integer numbers

7	2	1	8	3	4	2	9	0	8

We cannot say 7 is the maximum, but we are sure that 2 is not.

$$max = ? \qquad partial\_max = \mathbf{7}$$

How to find the maximum element of a vector

Given a vector of integer numbers

7	2	1	8	3	4	2	9	0	8

We compare now *partial_max* to the third element: 7 is greater than 1.

$$max = ? \qquad partial\_max = \mathbf{7}$$

How to find the maximum element of a vector

Given a vector of integer numbers

7	2	1	**8**	3	4	2	9	0	8

We scan the whole vector and we keep applying the same procedure.

$$max = ? \qquad partial\_max = \mathbf{7}$$

How to find the maximum element of a vector

Given a vector of integer numbers

7	2	1	**8**	**3**	4	2	9	0	8

We scan the whole vector and we keep applying the same procedure.

$$max = ? \qquad partial\_max = \mathbf{8}$$

How to find the maximum element of a vector

Given a vector of integer numbers

7	2	1	**8**	3	**4**	2	9	0	8

We scan the whole vector and we keep applying the same procedure.

$$max = ? \qquad partial\_max = \mathbf{8}$$

How to find the maximum element of a vector

Given a vector of integer numbers

7	2	1	**8**	3	4	**2**	9	0	8

We scan the whole vector and we keep applying the same procedure.

$$max = ? \qquad partial\_max = \mathbf{8}$$

How to find the maximum element of a vector

Given a vector of integer numbers

7	2	1	**8**	3	4	2	**9**	0	8

We scan the whole vector and we keep applying the same procedure.

$$max = ? \qquad partial\_max = \mathbf{8}$$

How to find the maximum element of a vector

Given a vector of integer numbers

7	2	1	8	3	4	2	**9**	**0**	8

We scan the whole vector and we keep applying the same procedure.

$$max = ? \qquad partial\_max = \mathbf{9}$$

How to find the maximum element of a vector

Given a vector of integer numbers

7	2	1	8	3	4	2	**9**	0	**8**

We scan the whole vector and we keep applying the same procedure.

$$max = ? \qquad partial\_max = \mathbf{9}$$

How to find the maximum element of a vector

Given a vector of integer numbers

7	2	1	8	3	4	2	**9**	0	8

At the end, *partial_max* contains the maximum element *max*, which is equal to 9.

$$max = partial\_max = \mathbf{9}$$

The algorithm

Given a vector of n integer numbers $(x_1, x_2, \ldots, x_n)$, find its maximum element *max*

1. **begin**
2. **set** $max = x_1$; *// initialization of max*
3. **for** $i = 2, n$
    - **if** $(x_i > max)$ **then**
        - $max = x_i$; *// updating max as the vector is scanned*
    - **end if**
4. **end for**
5. **print** max;
6. **end**

Note that we can avoid using a second variable for partial_max.

Implementation of the algorithm in C

We will see now how to implement this algorithm in C.

To this purpose, we need to learn how to write a program in C.

C programming language

We are going to study the **C** programming language.

It is a general-purpose computer programming language developed between 1969 and 1973 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system.

It is one of the most widely used programming languages of all time and there are very few computer architectures for which a C compiler does not exist.

C has greatly influenced many other popular programming languages, most notably C++, which began as an extension to C.

We will learn the basis of the language.

The main function

Each program in C is a function, which is called `main` function:

```
main()
{
   // the program in C can be written here
};
```

We will discuss more about functions in C later. *All you need to know right now is that all programs in C must have this general structure*.

Note that the symbol *//* indicates that what follows is a comment for the programmer, and therefore it is ignored by the compiler.

Variables

Variable: a symbol representing a quantity capable of assuming any of a set of values.

Data type: it defines the set of values that a variable can assume.

Standard data types in C:

- integer: int
- real: float (single precision) and double (double precision)
- character: char

The following code declares an integer variable called a

```
int a;
```

The following code assigns to the value 5 to the previously declared variable a:

```
a = 5;
```

Arrays

If we have to store *n* variables of the same type, we could use, in theory, *n* different variables, but it is usually preferable to consider just one **array of variables**.

In C, we can declare an array as follows:

```
int a[10];  // array of 10 integers
double v[3];// array of real numbers in double precision
char ch[5]; // array of 5 characters
```

and elements of an array can be assigned as follows:

```
a[3] = 1;
v[1] = 3.23;
ch[0] = 'x';
```

Note that the elements of an array are ordered from 0 to $n-1$, where *n* is the dimension of the array specified during the declaration.

Control structures in C

while ... end while

```
while (condition)
{
   // set of instructions
};
```

repeat ... until

```
do
{
   // set of instructions
}
while (!condition);
```

Control structures in C

if keyword

```
if (condition)
{
   // set of instructions A
}
else
{
   // set of instructions B
};
```

for loop

```
for (i = 0; i < n; i++)
{
   // set of instructions
};
```

I/O system in C

How can we communicate with the computer?

printf – function for writing a text on the standard output (generally, your monitor)

scanf – function for reading from the standard input (generally, your keybord)

Example

```
printf("please write an int, a double and a char:  ");
scanf("%d %lf %c",&a,&v,&ch);
printf("int %d, double %lf, char %c \n",a,v,ch);
```

Some special symbols:

`%d` (int) `%f` (float) `%lf` (double) `%c` (char) `%s` (array of char, string)

The function `scanf` *needs as input the address in the memory of the variables: the address can be obtained by adding* **&** *before the name of the variable.*

Translation in C of an algorithm


```
#include <stdio.h>
main ()
{
   int i,n;
   int x[100],max;

   printf("Max algorithm\n");
   printf("dimension of vector? ");
   scanf("%d",&n);
   printf("insert vector: ");
   for (i = 0; i < n; i++) scanf("%d",&x[i]);

   max = x[0];
   for (i = 1; i < n; i++)
   {
      if (x[i] > max) max = x[i];
   };
   printf("The max element is: %d\n",max);
};
```

Compilation with gcc

There are several C compilers, for Windows, Unix, Linux, etc.

We will consider the **gcc** compiler of **GNU**, which is installed on your Linux machine.

Compilation:

```
gcc -c filename1.c filename2.c
```

Generation of the executable:

```
gcc -o myprog filename1.o filename2.o
```

Running the program

Finally, that's how we can execute our program in C!

```
mylinuxmachine> myprog
Max algorithm
dimension of the vector?  4
insert vector: 4 6 -1 5
The max element is:  6
mylinuxmachine>
```

How to sort the elements of a vector

Given a vector *x* of *n* integer numbers

7	2	1	8	3	4	2	9	0	8

how can sort its elements from the smallest to the greatest?

Let's study one possible strategy for solving this problem.

How to sort the elements of a vector

Given a vector x of n integer numbers

7	2	1	8	3	4	2	**9**	0	8

how can sort its elements from the smallest to the greatest?

Find the element of the vector with maximum value x_{max}.

How to sort the elements of a vector

Given a vector x of n integer numbers

7	2	1	8	3	4	2	**9**	0	**8**

how can sort its elements from the smallest to the greatest?

Exchange positions for x_{max} and the last element of the vector.

How to sort the elements of a vector

Given a vector *x* of *n* integer numbers

7	2	1	8	3	4	2	**8**	0	**9**

how can sort its elements from the smallest to the greatest?

									9

The last element of the vector is now correctly sorted.

How to sort the elements of a vector

Given a vector x of n integer numbers

7	2	1	8	3	4	2	8	0	9

how can sort its elements from the smallest to the greatest?

									9

Let's apply the same procedure to $(x_1, x_2, \ldots, x_{n-1})$.

How to sort the elements of a vector

Given a vector x of n integer numbers

7	2	1	**8**	3	4	2	8	**0**	9

how can sort its elements from the smallest to the greatest?

									9

Let's apply the same procedure to $(x_1, x_2, \ldots, x_{n-1})$.

How to sort the elements of a vector

Given a vector x of n integer numbers

7	2	1	**0**	3	4	2	8	**8**	9

how can sort its elements from the smallest to the greatest?

									9

Let's apply the same procedure to $(x_1, x_2, \ldots, x_{n-1})$.

How to sort the elements of a vector

Given a vector x of n integer numbers

7	2	1	0	3	4	2	8	8	9

how can sort its elements from the smallest to the greatest?

								8	9

Let's apply the same procedure to $(x_1, x_2, \ldots, x_{n-1})$.

How to sort the elements of a vector

Given a vector x of n integer numbers

7	2	1	0	3	4	2	**8**	8	9

how can sort its elements from the smallest to the greatest?

							8	8	9

We can keep doing the same ...

How to sort the elements of a vector

Given a vector *x* of *n* integer numbers

7	2	1	0	3	4	2	8	8	9

how can sort its elements from the smallest to the greatest?

						7	8	8	9

We can keep doing the same . . .

How to sort the elements of a vector

Given a vector x of n integer numbers

2	2	1	0	3	4	7	8	8	9

how can sort its elements from the smallest to the greatest?

					4	7	8	8	9

We can keep doing the same . . .

How to sort the elements of a vector

Given a vector *x* of *n* integer numbers

2	2	1	0	3	4	7	8	8	9

how can sort its elements from the smallest to the greatest?

				3	4	7	8	8	9

We can keep doing the same . . .

How to sort the elements of a vector

Given a vector x of n integer numbers

2	2	1	**0**	3	4	7	8	8	9

how can sort its elements from the smallest to the greatest?

			2	3	4	7	8	8	9

We can keep doing the same . . .

How to sort the elements of a vector

Given a vector *x* of *n* integer numbers

0	2	1	2	3	4	7	8	8	9

how can sort its elements from the smallest to the greatest?

		2	2	3	4	7	8	8	9

We can keep doing the same . . .

How to sort the elements of a vector

Given a vector *x* of *n* integer numbers

0	1	2	2	3	4	7	8	8	9

how can sort its elements from the smallest to the greatest?

	1	2	2	3	4	7	8	8	9

We can keep doing the same . . .

How to sort the elements of a vector

Given a vector x of n integer numbers

0	1	2	2	3	4	7	8	8	9

how can sort its elements from the smallest to the greatest?

0	1	2	2	3	4	7	8	8	9

We can keep doing the same . . .

How to sort the elements of a vector

Given a vector x of n integer numbers

0	1	2	2	3	4	7	8	8	9

how can sort its elements from the smallest to the greatest?

0	1	2	2	3	4	7	8	8	9

We can keep doing the same . . .

The algorithm

Given a vector of n integer numbers $(x_1, x_2, \ldots, x_n)$, sort its elements from the smallest to the greatest

1. **begin**
2. **for** $k = n, 2, step = -1$
   - $m =$ **max**$(x_1, x_2, \ldots, x_k)$; *// index of maximum*
   - **exchange** x_m and x_k;
3. **end for**
4. **print** x;
5. **end**

Note that the algorithm makes use of a function named **max** *which provides the index of the maximum element in a vector*.

Not only the function “main”

Each program in C has a function named `main`:

```
main()
{
   // the program in C can be written here
};
```

Functions in C have this general structure:

```
int funct(int a,double b,char c)
{
   // the body of the function can be written here
};
```

This function is named `funct`, it has 3 input arguments `a` (`int`), `b` (`double`) and `c` (`char`), and its output value is an `int`.

Functions

Example of function in C:

```
int funct(int a,double b,char c);
```

Some remarks:

- the function has a returning value, whose data type is specified at the left of the function name;
- the list of input arguments of the function is after the function name, between parentheses.

Important:

- new copies of the variables are placed in memory when the function is called, so that variables modified *inside* the function remain unchanged *outside*;
- this is *generally not true* when arrays are considered: we'll come back to this later ...

Procedural programming

Each program is a set of functions:

- the program is divided in subprograms and subsubprograms, each of them represented by a single function, able to perform a predefined task;
- the data can be shared by all functions;
- each subprogram is a *mathematical function*, which, in general, provides the same output for the same input arguments;
- easier to projet, preferable for small, medium-small sized projects.

Procedural programming

Our sorting algorithm is based on another algorithm:
the algorithm for finding the maximum element in a vector.

Can we use the program we developed before?? NO.

We need to work a little more on this:

- we wrote the algorithm in the function `main`, and not in an independent C function;
- the output of our algorithm is the value of the maximum element, and not its index.

Procedural programming

Our sorting algorithm is based on another algorithm:
the algorithm for finding the maximum element in a vector.

Can we use the program we developed before?? **NO.**

We need to work a little more on this:

- we wrote the algorithm in the function `main`, and not in an independent C function;
- the output of our algorithm is the value of the maximum element, and not its index.

The program we already developed


```
#include <stdio.h>
main ()
{
   int i,n;
   int x[100],max;

   printf("Max algorithm\n");
   printf("dimension of vector? ");
   scanf("%d",&n);
   printf("insert vector: ");
   for (i = 0; i < n; i++) scanf("%d",&x[i]);

   max = x[0];
   for (i = 1; i < n; i++)
   {
      if (x[i] > max) max = x[i];
   };
   printf("The max element is: %d\n",max);
};
```

The C function


```
int max(int n,int *x)
{
   int i,m,maxv;

   m = 0;
   maxv = x[m];

   for (i = 1; i < n; i++)
   {
      if (x[i] > maxv)
      {
         m = i;
         maxv = x[m];
      };
   };

   return m;
};
```

- `int *x` indicates that `x` is a array;
- the keyword `return` is used to indicate the output variable at the end of the execution.

C function for the sorting algorithm


```
void sort(int n,int *x)
{
   int k,m,aux;

   for (k = n - 1; k >= 1; k--)
   {
      m = max(k+1,x);
      if (k != m)
      {
         aux = x[k];
         x[k] = x[m];
         x[m] = aux;
      };
   };
};
```

- `void` indicates that there is no output argument.

One main for the 2 functions


```c
#include <stdio.h>
void sort(int n,int *x); // function prototypes
int max(int n,int *x);

main ()
{
   int i,n;
   int x[100];

   printf("Sorting algorithm\n");
   printf("dimension of vector? ");
   scanf("%d",&n);

   printf("insert vector: ");
   for (i = 0; i < n; i++) scanf("%d",&x[i]);

   sort(n,x); // calling the function sort

   printf("the sorted vector is: ");
   for (i = 0; i < n; i++) printf(" %d ",x[i]);
   printf("\n");
};
```

Running the program

Compilation and execution:

```
mylinuxmachine> gcc -c main.c
mylinuxmachine> gcc -c max.c
mylinuxmachine> gcc -c sort.c
mylinuxmachine> gcc -o mysort main.o max.o sort.o
mylinuxmachine> ls
 main.c main.o max.c max.o mysort sort.c sort.o
mylinuxmachine> mysort
Sorting algorithm
dimension of vector?  10
insert vector:  -1 8 -3 11 5 13 -9 8 4 10
the sorted vector is:  -9  -3  -1  4  5  8  8  10  11  13
mylinuxmachine>
```

Other algorithms for sorting

The studied algorithm is not the most efficient algorithm for sorting vectors.

- Bubble sort;
- Insertion sort;
- Quicksort;
- Shell sort;
- Merge sort;
- Distribution sort;
- ...

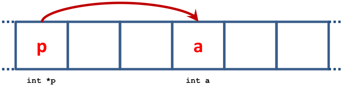
Comparisons and general information about the existing algorithms can be found at:

`http://en.wikipedia.org/wiki/Sorting_algorithm`

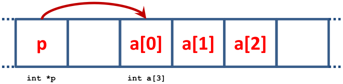
How to access to the memory

Pointer in C: a variable that holds the address of another variable or the first address of an array of variables.

```
int a;
int *p;
p = &a;
```




```
int a[10];
int *p;
p = a;
```



Note that:

- arithmetic operations can be performed on pointers (e.g. `p+1` is another pointer);
- different pointers can refer to the same memory address.

Pointers and functions

The input arguments of functions in C are copied in new places of the memory.

```
int funct(int a,double b,char c,int *d);
```

- If any of these arguments is modified during the execution of the function, its value remains unchanged in the calling function (e.g. the main function);
- If one of the arguments is an array, we specify its pointer:
  - if we modify the value of the pointer in the function, this change cannot affect the pointer in the calling function;
  - if we modify the array in the function, these changes are also visible in the calling function (*the two pointers refer to the same space in the memory*).

Static vs. Dynamic allocation of memory

We know how to allocate memory *statically*:

```
int a[10];
```

What if we don't know the dimension of the array when we declare it?

Solution: dynamic allocation

```
int *a; // pointer to int
...
a = calloc(n,sizeof(int)); // memory allocation
...
free(a); // memory deallocated
```

Note that:

- the variable `n` must be an integer that contains the desired dimension for `a`;
- `n` is defined during the execution of the program;
- `calloc` and `sizeof` are two functions included in the standard C library (`stdlib`).

Sorting algorithm: a new version for the main


```
#include <stdio.h>
#include <stdlib.h>
void sort(int n,int *x); int max(int n,int *x);

main ()
{
   int i,n,*x;

   printf("Sorting algorithm\n");
   printf("dimension of vector? ");
   scanf("%d",&n);
   x = calloc(n,sizeof(int));
   printf("insert vector: ");
   for (i = 0; i < n; i++) scanf("%d",&x[i]);

   sort(n,x); // calling the function sort

   printf("the sorted vector is: ");
   for (i = 0; i < n; i++) printf(" %d ",x[i]);
   printf("\n");
   free(x);
};
```

Matrices

A **matrix** A is a table of mathematical expressions, that, in the easiest case, consist of real or integer numbers.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2m} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3m} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nm} \end{pmatrix}$$

- matrices contain a predefined number n of rows, and a predefined number m of columns;
- each element of A is indexed by 2 integer numbers: a_{ij}.

Basic operations with matrices

The sum between two matrices A and B:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

is:

$$C = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} \end{pmatrix}$$

The dimensions n and m of the matrices must be the same.

Basic operations with matrices

The product between two matrices A and B:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

is:

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

such that:

$$c_{ij} = c_{i1} \times c_{1j} + c_{i2} \times c_{2j} + c_{i3} \times c_{3j}$$

The number of columns of A must correspond to the number of rows of B.

How to consider matrices in C

Different ways to see a matrix:

- in mathematics: a vector of vectors;
- in computer science: an array of arrays;
- in C: a pointer to an array of pointers.

Declaration in C of a matrix having dimension $n \times m$:

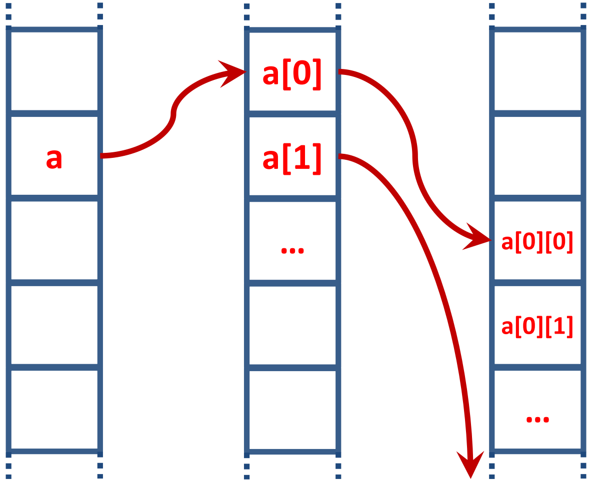
```
// declaration of a pointer to a pointer (**)
int **a;

// allocation of memory for an array of pointers (int*)
a = calloc(n,sizeof(int*));

// allocation of memory for each pointer in the array a
for (i = 0; i < n; i++)  a[i] = calloc(m,sizeof(int));

// memory deallocation
free(a);
```

The situation in memory



Sum between matrices: function in C

This function in C is able to perform the sum between two matrices:

```
void matrixSum(int n,int m,
               double **a,double **b,double **c)
{
   int i,j;
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < m; j++)
      {
         c[i][j] = a[i][j] + b[i][j];
      };
   };
};
```

Product between matrices: function in C

This function in C is able to perform the product between two matrices:

```
void matrixProduct(int nA,int mA,int nB,int mB,
                   double **a,double **b,double **c)
{
   int i,j,k;
   if (mA == nB)
   {
      for (i = 0; i < nA; i++)
      {
         for (j = 0; j < mB; j++)
         {
            c[i][j] = 0.0;
            for (k = 0; k < mA; k++)
            {
               c[i][j] = c[i][j] + a[i][k] * b[k][j];
            };
         };
      };
   };
};
```

Sorting the rows of a matrix

Let us consider now the following problem:

Given a matrix A,
sort its rows so that the first $i-1$ *elements of the* i^{th} *row are equal to 0.*

Questions and remarks

1. Can all matrices A be sorted as requested?
   - No, this matrix is an example: $\begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 7 \\ 2 & 8 & -1 \end{pmatrix}$;
2. How can we efficiently sort the rows of a matrix in C?
   - we sort the pointers to its rows, i.e. `a[i]`.
3. But the pointers represent a memory address, how to sort them?
   - we define an array which counts the number of elements that are 0 at the beginning of each row;
   - we sort this array by using algorithm we already studied;
   - while sorting, we also exchange the values of the corresponding pointers `a[i]`.

Implementation of the algorithm in C


```
void matrixSort(int n,int m,double **a)
{
   int i,j,k;
   int *ct,mx,aux;
   double maxv,*paux;
   ct = (int*)calloc(n,sizeof(int));
   for (i = 0; i < n; i++)
   {
      j = 0;
      ct[i] = 0;
      while (j < m && a[i][j] == 0)
      {
         ct[i]++;  j++;
      };
   };
   for (k = n - 1; k >= 1; k--)
   {
      mx = max(k+1,ct);
      if (k != mx)
      {
         aux = ct[k];  ct[k] = ct[mx];  ct[mx] = aux;
         paux = a[k];  a[k] = a[mx];  a[mx] = paux;
      };
   };
   free(ct);
};
```

Implementation of the algorithm in C


```
#include <stdio.h>
#include <stdlib.h>
void matrixSort(int n,int m,double **a);
int max(int n,int *x);

main ()
{
   int i,j,n,m;
   double **a;

   printf("Sorting algorithm for matrices\n");
   printf("dimensions of matrix (n,m)? ");
   scanf("%d%d",&n,&m);
   a = calloc(n,sizeof(double*));
   for (i = 0; i < n; i++) a[i] = calloc(m,sizeof(double));
   printf("insert matrix:\n");
   for (i = 0; i < n; i++)  for (j = 0; j < m; j++)  scanf("%lf",&a[i][j]);

   matrixSort(n,m,a);

   printf("sorted matrix:\n");
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < m; j++)  printf(" %lf ",a[i][j]);
      printf("\n");
   };

   free(a);
};
```

Running the program

Compilation and execution:

```
mylinuxmachine> gcc -c main.c
mylinuxmachine> gcc -c max.c
mylinuxmachine> gcc -c matsort.c
mylinuxmachine> gcc -o matsort main.o max.o matsort.o
mylinuxmachine> matsort
Sorting algorithm for matrices
dimensions of matrix (n,m)?  4 4
insert matrix:
0 0 0 1
0 1 2 3
0 0 1 2
1 1 1 1
sorted matrix:
 1.000000 1.000000 1.000000 1.000000
 0.000000 1.000000 2.000000 3.000000
 0.000000 0.000000 1.000000 2.000000
 0.000000 0.000000 0.000000 1.000000
mylinuxmachine>
```

Libraries of C functions

There are libraries of functions in C that can be used for the development of our programs:

- they are sets of object files (`.o`), obtained during the compilation of C functions
- the user can use the functions of the libraries for the development of his programs, but it might not have access to its sources (`.c`)
- libraries usually have the `.a` extension
- as object files, libraries can be added to the list of files that are needed for the generation of programs:

```
gcc -o myprog mymain.o myfun1.o myfun2.o lib.a
```