Introduction to MATLAB basics

This is Chapter 1 of the textbook plus Chapter 3 (page 99)

Disclaimer: *These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Fall 2011 ; distribution outside this group of students is NOT allowed.*

C0.1 What is a function (mathematics) and what is a function (computer science)? In mathematics a function is a relation between a set of elements (the domain) and another set of elements (the codomain) defined so that each element of the domain is associated with exactly one element of the codomain. In computer science (or programming), a function is a piece of code that performs a specific task. A function in computer science is sometimes called a subroutine, subprogram, method, procedure, routine, or just program (if it performs only a single task).

C0.2 Function names in mathematics and computer science and MATLAB. In mathematics when we write `incr`(x), we so denote the INCREMENT functions that returns $x+1$ for input x. If we decide to implement this function, we use the same name `incr` or something different that is more informative (e.g. `incrementbyone`). Of course using such a long name raises readability issues: we might write instead `increment_by_one` by using underscores to separate words. For other cases, a more drastic action is need; for example most of us will be hard-pressed to type in a keyboard $\sqrt{x}$. Instead, we usually write something like `sqrt`(x) to denote the $\sqrt{x}$ function. Thus for some very familiar mathematical functions (e.g. those appearing in a scientific calculator), MATLAB might be using different names than the ones with which we are familiar.

C0.3 How is a function defined ? Parameter vs Argument. In mathematics we could define `incr`(x) very simply by writing $f(x) = x + 1$. Math function names are very short and most of the times f, g, h are being used as names. When we write computer programs we need to be more imaginative: we can't name all of our functions f. In addition when we write x the implicit assumption is that x is a number. In MATLAB for example inputs to a functions can be a number, a vector (a linear collection of numbers), or a generic matrix. Given that computers are dummy, they do not know that f denotes a mathematical function and not say the sixth letter of the alphabet in lower-case. Thus when we write programs we need to provide lots of details about what we attempt to do (program) and what we expect as input and as an outcome (output). Is the output a number similarly to the input, or is it just a number despite the input being a vector? For example is $f(1/3)$ ok , i.e. is $x = 1/3$ a legitimate argument? Are rational numbers allowed as an input? And if you say yes, how about complex numbers? If you don't know what a complex number is that is fine: our intend was to show you to expect the unexpected! Now that we are still talking about f, let us remind ourselves that x is the **parameter** of function f. When we evaluate f at a point and we ask 'What is the value of $f(1/3)$' we substitute $1/3$ for x. This value becomes the **argument** to the function. Thus **parameter** is something generic that is used to help us define a function, and argument is something specific that help us with the evaluation of a function at a given point.

Conclusion. MATLAB functions might have different names than their MATH counterparts and might also behave differently. Be prepared!

(c) Copyright A. Gerbessiotis. CS 101-005 : Fall 2011 . All rights reserved.

C0.4 MATLAB is an interpreted weakly-typed language. This means that the same variable x can be assigned an integer value, a real value, a string value, yet MATLAB will never complain about this. However it is the responsibility of the user and/or programmer to remember how a variable is being used and treated.

C0.5 MATLAB file types: An M-file. MATLAB recognizes and treats special filenames in a specific way. One such case is a filename ending with the suffix `m`. This is known in MATLAB as an M-file. For example `file.m` is such a file. For this case `file` is the prefix, `m` is the suffix and the period separates the prefix from the suffix. Sometimes `m` is called the file extension. In MATLAB an M-file can contain a sequence or collection of MATLAB functions/commands. One can call such an M-file from within MATLAB by typing in the name of the prefix only, that is by typing `file` without the `.m`. Then the sequence of MATLAB functions in the M-file will be executed and when this completes control will pass back to the MATLAB command window and the prompt will appear.

Reminder. When calling an M-file, do not type `.m`.

C0.6 MATLAB special M-files: startup.m and finish.m. There are two special M-files in MATLAB whose function are executed. The M-file `startup.m` that is executed as soon as MATLAB loads up (and the command window appears), and `finish.m` that is being called when the user is about to exit the command window by say, typing the function `quit`. Note that those two file will be executed as long as they reside in the directory from which MATLAB will be invoked or from which MATLAB will be terminated. One way to see whether the execution of `startup.m` or `finish.m` has occurred or is to occur is to type `dir` to get the directory listing of the current directory and confirm the existence of one or the other file there.

C0.7 dir,ls, type, pwd, cd The function `dir` (MSDOS/Windows command prompt relic), or `ls` (UNIX relic) both list the contents of the current working directory of MATLAB. What this is can be revealed by typing `pwd` (which stands for `print working directory`). You can view the contents of a file say `finish.m` by typing the MSDOS/Windows-style `type finish.m` command with a space separating the two. And if you want to change directory, the `cd` command can be used appropriately (UNIX or Windows syntax as required).

(c) Copyright A. Gerbessiotis. All rights reserved.

C0.8 ans, clear, who, whos, semicolon(;) and clc . The simplest use of MATLAB is as a rudimentary calculator. Every time a calculation such as `1+2` is typed and the ENTER/Return key is pressed, the result is computed and stored into a default variable named `ans` for answer. In addition the assignment of the result to `ans` is shown onscreen. This display can be suppressed by typing a semicolon at the end of the expression as in `1+2;`. An expression can be assigned to another variable such as x by typing `x=1+2` or `x=1+2;` respectively. MATLAB keeps track of all defined variables in its Workspace. The user can detect them by typing `who` where minimal information is provided, or `whos` where size information (per variable and total is provided). Note that the default abstraction in MATLAB is that of a MATRIX and every variable so defined is a MATRIX or more specifically a scalar value (i.e. a 1×1 matrix). The default data-type for a MATRIX element is a `double` 64-bit real number. Given that the internal representation of a MATRIX is by using an array, this information is also available when one types a `whos`.

If one issues a `clear` this function will erase all variable already defined and its associated with values. A `clc` on the other hand will only clear the display but not the MATLAB workspace.

C0.9 Special variables: pi, i,j. MATLAB uses some special variable. `pi` attains the value of π (the ratio of the circumeference to the diameter of a circle) up to a number of decimal digits allowed by the use of a `double` data type. When one types `pi` an approximation of π is printed stored into variable `pi` to a number of decimal digits. The default is 4 decimal digits that can be enforced also by typing `format short`. If more digits are needed `format long` can achieve this and prints 15 decimal digits as needed. However the number of decimal digits maintained by a double data type may be larger than 4 or 15. There are two other special variable i, j. These are the square root of -1 and are used to describe complex numbers. Thus $i * i = j * j = -1$.

C0.10 Beware of the special variables. Even if special variables have special meaning their value can be overidden. For example `pi=3` will create a local variable named `pi` whose value is 3. The special variable still exists, but the local variable has PRECEDENCE of appearance. Thus a clear will only destroy the local copy of `pi` and will restore access to the more desirable value/variable. The same of course applies to i, j as well.

C0.11 MATLAB comment is the percent sign. A comment in MATLAB can be introduced by typing in the percent (%) sign. Everything that follows it on a line is considered a comment and thus ignored by the MATLAB interpreter (and thus our Ouch! did not cause a problem) in the line that reset `pi` to 3.

The following pages give MATLAB output for the examples of this page and more.

(c) Copyright A. Gerbessiotis. All rights reserved.

```
>> % This is a comment; Comment lines begin with a percentage sign
>> a=1
a =
     1

>> b=a/3
b =
    0.3333

>> b-0.3333
ans =
   3.3333e-05

>> f=4;
>> g=4
g =
     4
>> f
f =
     4
>>
```

In the example above a comment line was introduced; it starts with a percentage % character. The remainder of that line is considered a comment and is not interpreted by MATLAB.

In the following line a variable `a` was introduced. A variable like `a` becomes an 1×1 matrix, i.e. it is an elementary vector that we used to call `scalar`. Another variable `b` was introduced. The initial value of `a` was a numeric constant i.e. `1`. The initial value of `b` was the value of an expression. That expression `a/2` was evaluated according to the precedence rules of Subject 3 and its value was assigned to `b`. Note that `1/3` for MATLAB is a number with some internally adjusted precision. When we print `b` or MATLAB gives us the value of `b` only few (in this case 4) decimal digits are printed. This does not mean that `b` is indeed `0.3333` ; it means that `b` is a number that if printed to four decimal digits we get `0.3333`. Thus if we try to evaluate `b -0.3333` the results won't be equal to zero. As it is shown it is larger than zero.

Note that after every line we can conclude that line with a semicolon `;`. In `f=4;` nothing is printed afterwards afer we hit ENTER. As opposed to the `g=4` that has a semicolon missing. We can retrieve the value of a variable by typing in its name and (of course hitting ENTER).

(c) Copyright A. Gerbessiotis. All rights reserved.

```
>> x=1+2;
>> y= 4
   y =
      4
>> who
Your variables are:
 x  y
>> whos
  Name      Size                    Bytes  Class
  x         1x1                         8  double array
  y         1x1                         8  double array
Grand total is 2 elements using 16 bytes
>> clear
>> who
>> whos
>> pi
 ans = 3.1416
>> format long
>> pi
 ans = 3.14159265358979
>> i*i
ans = -1
>> j*j
ans = -1
>>i
   0+ 1.000000000000000i
>> whos
  Name      Size                    Bytes  Class
  ans       1x1                         8  double array
Grand total is 1 element using 8 bytes
>> pi= 3   % Ouch !!
>> whos
Name      Size                    Bytes  Class
  ans       1x1                         8  double array
  pi        1x1                         8  double array
Grand total is 2 elements using 16 bytes
>> pi
 pi = 3
>> clear
>> pi
 ans = 3.14159265358979
>> f = @(x) (x*x)
f = @(x) (x*x)
>> f(2)
ans = 4
>> f(5)
ans = 25
```

C0.12 MATLAB anonymous functions. In MATLAB a very easy way to define a very simple function is by creating on-the-fly an "anonymous" function whose presence does not require an M-file (most MATLAB function are required to reside in an M-file). Thus a function x^2 is defined by typing `f=@(x) (x*x)`. We not that x, the indeterminate, is the parameter of f in this context whereas $2, 5$ are the arguments for which the function f evaluates to 4 and 25 respectively.

(c) Copyright A. Gerbessiotis. All rights reserved.

C0.13 MATLAB MAT file. A MAT file is a file with the `mat` suffix. A file such as `mydata.mat` is such a file. In MAT files MATLAB stores data such as variables (their names) and their values). Thus incomplete calculations can be saved into a MAT file and later retrieved.

C0.14 MATLAB load, save functions. All variables defined in a MATLAB command window can be saved into a MATLAB file say `mydata.mat` by typing `save mydata` (the suffix will be created automatically). One can then perform a clear to clear the workspace. The data are still available in `mydata.mat`. Then one can load them by using `load mydata` (the suffix is added as needed). A user can selectively load a specific variable by typing instead `load mydata y`, or `load mydata x y`, where the list of variables to be loaded follows the name of the MAT file. This applies also to the `save` command. A `save mydata x y` could have also been used for the same effect as `save mydata`. The `append` option of save can be used to add one more variable into `mydata` without modifying `x,y`. This can be effected by typing `save -append mydata z`.

```
>> x=1;
>> y=2;
>> whos
  Name      Size                    Bytes  Class
  x         1x1                         8  double array
  y         1x1                         8  double array
Grand total is 2 elements using 16 bytes
>> save mydata
>> dir
 mydata.mat
>> clear
>> whos
>>
>> load mydata
>> whos
  Name      Size                    Bytes  Class
  x         1x1                         8  double array
  y         1x1                         8  double array
Grand total is 2 elements using 16 bytes
>> z=4;
>> save -append mydata z
>> clear
>> whos
>>load mydata
>>whos
  Name      Size                    Bytes  Class
  x         1x1                         8  double array
  y         1x1                         8  double array
  z         1x1                         8  double array
Grand total is 3 elements using 24 bytes
```

(c) Copyright A. Gerbessiotis. All rights reserved.



Figure 1: Scalar operator $+, -, *, /$ usage



Figure 2: Scalar operator $\backslash, \hat{}$ usage

C1.1. Operators $+, -, *, /, \backslash, \hat{}$. An operator such as $+, -, *, /, \backslash$ is a binary operator that accepts two arguments and combines them according to the rules implied by the indicated operator by performing the corresponding operation. Thus operator $+$ (i.e. plus) adds its two arguments by realizing the operation known as addition. The operator $-$ indicates subtraction and the operator $\backslash$ backdivision, that is $x\backslash y$ is y/x. Operator $\hat{}$ used as $a\hat{}b$ raises a to the b-th power.

Left-to-right order of evaluation. If multiple but same type operators are defined as in `4 * 3 *2 *1` the product is realized left-to-right i.e. first `4*3` is computed, then the result is multiplied by 2 and so on. Thus for `1-2-3`, we first perform `1-2` that gives `-1` as the answer and then `-1-3` is performed to give `-4` as the final answer.

Precedence of $+, -$ and $/, *$. Operators $+$ and $-$ have the same precedence and so do $*$ and $/$. Same precedence operations are evaluated left to right. Thus $2 * 10/5$ will first evaluate $2 * 10$ and then the result will be divided by 5. This is different from the rule that says that division has higher priority!

C1.2. Associativity. Operations such as addition and multiplication are called associative in the sense that the corresponding operators satisfy $a * (b * c) = (a * b) * c$ and $a + (b + c) = (a + b) + c$ for all a, b, c. Subtraction and division are not associative. For example, $(1 - 2) - 3 = -4 \neq 1 - (2 - 3) = 2$.

C1.3. Commutativity. Operators $*$ and $+$ are commutative, that is, $a + b = b + a$ and $a * b = b * a$, for all a, b.

C1.4. Unary operators $+, -$, and operator $\hat{}$. The unary operator $+$ indicates a positive number and the $-$ a negative number. Both of them have higher precedence than the corresponding binary operators $+, -$ but $\hat{}$ has even higher precedence than all of them (binary or unary).

(c) Copyright A. Gerbessiotis. All rights reserved.

C1. More on scalar operators $+, -, *, /, \backslash, \hat{}$

In Figure 1 we use MATLAB to perform some very simple scalar operations involving + and − (i.e. addition and subtraction) and $*, /$ (i.e. multiplication and division). When we type in `1+2` in the MATLAB prompt and press ENTER, the result is stored in a MATLAB scalar called `ans` (for answer). The value calculated is then printed. Because the answer in all cases involved an integer the answer is printed as an integer. In Figure 2 we use MATLAB to further study its division capabilities. The symbol / when used in the context of `8/3` calculates a non-integer result. The answer is then printed using the default number of decimal digits, in this case 4 and the result is rounded appropriately. When the alternative division operator is used $2\backslash 8$, the result 4 is printed as an integer. We are reminded that $a\backslash b$ is equal to b/a. Then the expression `1-2-3` is evaluated. This is an ambiguous expression because between the two subtractions one needs to figure out which one to perform first. If there is an ambiguity MATLAB decides based on it own rules (left-to-right) to resolve this. If you want another order of evaluation, you must tell MATLAB so by using parentheses! **WE STRONGLY RECOMMEND** that you use parentheses in your MATLAB expressions.

Order of evaluation. The order of evaluation of an expression that uses multiple scalar operators and is absent of parentheses that force execution in a certain way depends on the precedence of a particular operator. Precedence of operators including scalar operators is discussed at the end of this Subject.

C1.4. MATLAB abstraction: matrix MATLAB's data model revolves around the matrix object. Every object is MATLAB is a matrix. A variable storing a single value (integer or real or something else) in MATLAB, is called a **scalar**, and is a 1×1 matrix.

C1.5 MATLAB implementation of a matrix: array A matrix object is represented internally in MATLAB by a data structure known as an array which is a linear segment of space that holds multiple number of elements the same data type.

C1.6 Operator overloading. The operators $+, -$ signify addition not just of number but of arbitrary matrices as well. Thus $1 + 2$ for MATLAB denotes the addition of two 1×1 matrices. $*, /, \backslash$ can be used for general matrices (more than one rows/columns) to denote matrix multiplication and division as explained in Subject 2. However if an operation similar to $+, -$ is required the corresponding array operator must be used. Thus $.*, ./, .\backslash$ are the array operators for array multiplication, division and backdivision (See Subject 2, page 17).

C1.7 Variable names. Although we will discuss them in more detail in the next Subject variables names in MATLAB can have no more than 63 characters (or whatever `namelengthmax` returns) must start with a lower-case or capital case (English alphabet) letter (A to Z or a to z) but the remaining characters can be those alphabetic characters or a digits 0 to 9 or the underscore sumbol _. (The underscore symbol is not the dash symbol − used for the subtraction operator.)

(c) Copyright A. Gerbessiotis. All rights reserved.



Figure 3: Relational operator usage



Figure 4: Relational operator arithmetic behavior

C2.1 Relational operators $<, >, <=, >=, \sim=, ==$.
MATLAB defines operators $<, >, <=, >=, \sim=, ==$ which are the familiar
`less-than`, `greater-than`, `less-than-or-equal-to` and `greater-than-or-equal-to` relational operators plus the unique to MATLAB form of the `not-equal-to` operator as well as the more prevalent form of the `equal-to` operator; ;in both cases the operator is denoted by two consecutive symbols the tilde followed by the equal sign $\sim=$ for `not-equal-to`, or two consecutive equal signs for `equal-to`. All operators are **binary** i.e. they accept two arguments (left and right of the operator). The result of the comparison between the arguments is true or false. For MATLAB, true is the numeric value `1` and false is `0`. In MATLAB there are variables `true` and `false` whose values are respectively `1` for true and `0` for false. Note that even if the value of such a variable is a single bit, MATLAB uses at least a full byte to represent it.
Therefore, the result $1 < 2$ is true, i.e. `1` whereas $1 > 2$ is false, i.e. `0`. If we want to test for equality we need to write two $=$ one immediately following the other, or just test for inequality and flip the answer. $1 \sim= 1$ is naturally false, i.e. `0`. Figure 3 depicts some examples related to this discussion. Figure 4 depicts some irregular examples that take advantage of the fact that the relational operators return a `0,1` as an answer. Give that they are binary operators, $1 < 2 < 3$ is correctly computed as `1` using the left-to-right evaluation absence of parenthesization. If there is one good reason to avoid practices such as those depicted in Figure 4, then the last example of $(2 < 3 > 0)$ returning `1` is such.

(c) Copyright A. Gerbessiotis. All rights reserved.

Note that we have used invariably three different ways to denote the same thing.

- We wrote true or false to say that the result of a comparison is true of false. This is written in plain-text without any font insertion and it's a statement.
- We wrote `true` and `false` in a different font to indicate the MATLAB variables of the same name that hold a true and false value respectively. These variables, by default take the values of `1` and `0` respectively, corresponding to a true and false value.
- We wrote `1` and `0` to indicate the values of MATLAB variable `true` and `false` respectively. These variables represent in MATLAB true and false.

One way to disambiguate this usage , is to consider the well known π symbol (by the way MATLAB prefers to using `pi` instead). This variable takes, for most human beings the value of $\pi = 3.14$. In fact we call it a **constant** because its value is fixed through its definition that states that π is the ratio of any circle's circumference to its diameter. The value of 3.14 set to π is not the true value of the ratio but an approximation of it to two decimal digits. So we humans picked the symbol π to denote that ratio, and also picked the value of 3.14 (to two decimal digits) as the value of it. Similarly, MATLAB picked the variable names `true` and `false` (rather than TRUE or True etc) to denote the true and false values and to ease evaluation with them assigned to them the numeric values of 1 and 0 respectively. However the MATLAB `true, false, pi` are not **constant** but `variables`; you can changed their values intentionally or accidentally. **Therefore, be very very careful in dealing with them**.

```
>> 1 == 2 ;    % A very dangerous sequence of evaluations  1==2 return false and result stored in ans
>> ans == 0    % thus ans == 0 is equivalent to asking 0 == 0 which is true(1) and result stored in ans
  ans =1
>> ans         % Thus when ans is printed value is 1
  ans = 1
>> whos
  Name      Size                    Bytes  Class
  ans       1x1                         1  logical array
Grand total is 1 element using 1 bytes
```

C2.2 Relational operator results stored into a `logical` data type not `double`. As a result of the abovementioned evaluation MATLAB uses not a `double` data type but a `logical` data type to store the result in `ans`.

```
>> 1 == 1
  ans = 1
>> whos
  Name      Size                    Bytes  Class
  ans       1x1                         1  logical array
Grand total is 1 element using 1 bytes
>> 1*1
 ans =1
>> whos
  Name      Size                    Bytes  Class
  ans       1x1                         8  double array
Grand total is 1 element using 8 bytes

>> true  = 0;
>> false = 1;
```

C2.3 MATLAB is a weakly-type language. Notwithstanding the disucssion of the item `C2.2`, if a relational operator is succeeded by an arithmetic evaluation (such as $1 * 1$) this will be detected by MATLAB and will result to storing the new result into a arithmetic data type; the default such data type is a `double`.

C2.4 `true, false` overload! A note of danger. One can easily mistype a `true == 0` (i.e. an application of a relational operator) to type instead a `true = 0` (i.e. an assignment of a zero value to special variable `true`). MATLAB will create a local Workspace variable `true` with that value and from that point on this `true` will be used for `true`. **BE VERY CAREFUL** with your equals!

C3.1. Logical operators $\&, |,$ `xor`$, \sim$**.** The definitions for the
(a) LOGICAL-AND (&),
(b) LOGICAL-OR or LOGICAL-INCLUSIVE-OR (|),
(c) LOGICAL-EXCLUSIVE-OR or LOGICAL-XOR (`xor`), and
(d) the LOGICAL-NOT (∼)
operators are given in Table 1, Table 2, Table 3, Table 4 respectively.

C3.2 `true` **vs** `FALSE` The arguments of the logical operators can be the `true` or `false` variables of MATLAB or arbitrary arithmetic expressions. If the expression evaluates to something equal to `0` it is `false` i.e. it takes a value equal to the value of variable `false` which is 0. Otherwise, if it evaluates to any non-zero value it is `true` i.e 1. Any non-zero value is `true` and a zero value is always `false`. Note that `false` is lower-case false; `FALSE` is something totally different.

C3.3. LOGICAL-AND. A LOGICAL-AND on two arguments evaluates true if both its arguments are `true`; otherwise it is false. We write $a\&b$ to denote the LOGICAL-AND of a and b.

C3.4. LOGICAL-OR. A LOGICAL-OR on two arguments evaluates false if both its arguments are `false`; otherwise it is true. We write $a \mid b$ to denote the LOGICAL-OR of a and b.

C3.5. LOGICAL-XOR. A LOGICAL-XOR on two arguments evaluates false if both its arguments are `false` or both are `true`; otherwise it is true (i.e. when one argument is `true` and one is `false`). It is for this reason that we call it an exclusive OR; it does not allow both arguments to be true/false at the same time. We write `xor`(a, b) to denote the LOGICAL-XOR of a and b.

C3.6. LOGICAL-NOT. A LOGICAL-NOT on an argument is `true` if the argument is `false`, and `false` if the argument is `true`. We write $\sim a$ to denote the LOGICAL-NOT of a.

Caution. In the heading of C3.2 we used `true` and `FALSE`. If variable `FALSE` is 1 (and note `false` and `FALSE` are different) then it is conceivable that `true` & `FALSE` gives as an answer `1`, i.e. `true`.

Caution. First all of the argument are evaluated and then the corresponding operator is applied for LOGICAL-AND, LOGICAL-OR, LOGICAL-XOR, LOGICAL-NOT.

(c) Copyright A. Gerbessiotis. All rights reserved.

C3.6. Logical shortcut operators &&, ||**.** The definitions for the
(e) LOGICAL-AND-WITH-SHORTCUT-EVALUATION (&&), and
(f) LOGICAL-OR-WITH-SHORTCUT-EVALUATIOn (||)
are given by tables identical to & and | respectively. The only difference is that the SHORTCUT operators &&, || may not evaluate one of their two arguments. This means that if we were to evaluate a && b, as soon as a is evaluated to `false` there is no reason to evaluate b. The expression is `false`. Similarly for a || b, if a is `true`, there is no reason to evaluate b. If you expect that both arguments will be evaluated use the standard &, | operators; if you want to lazy-evaluate the expression, use (WITH CAUTION) the SHORTCUT versions of the operators.

C3.7. LOGICAL-AND-WITH-SHORTCUT. A LOGICAL-AND-WITH-SHORTCUT on two arguments evaluates immediately false if the left argument is false; otherwise the second argument is evaluated and its value determines the result (true if it is `true` and `false` otherwise).

C3.8. LOGICAL-OR-WITH-SHORTCUT. A LOGICAL-OR-WITH-SHORTCUT on two arguments evaluates immediately true if the left argument is true; otherwise the second argument is evaluated and its value determines the result.

Caution. For LOGICAL-AND-WITH-SHORTCUT, LOGICAL-OR-WITH-SHORTCUT the left argument is evaluated first; whether the right argument will be evaluated or not will depend on the first evaluation and whether the first evaluation suffices to obtain the answer! Thus 0&& `toast` will not complain about the undefined variable `toast` whereas 0& `toast` will loudly complain about it.

(c) Copyright A. Gerbessiotis. All rights reserved.

C3. Tables and examples of using Logical operators in MATLAB

In the tables that follow, an `1` is taken to mean any non-zero value; and `0` is anything that evaluates to zero. The answer `true` in MATLAB is given by an output that is `1` and `false` an output that is `0`.

Thus if one gives in MALTAB `pi&pi`, then the output will be `1` for true. This is because both `pi` evaluate to a non-zero i.e. `true` value. And of course if one gives `true | false`, the output will also be `1` since `true` evaluates to `1` and `false` to `0`.

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	0

Table 1: Operator &

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

Table 2: Operator |

a	b	`xor`(a,b)
0	0	0
0	1	1
1	0	1
1	1	0

Table 3: Operator `xor`

a	$\sim$ a
0	1
1	0

Table 4: Operator $\sim$

(c) Copyright A. Gerbessiotis. All rights reserved.

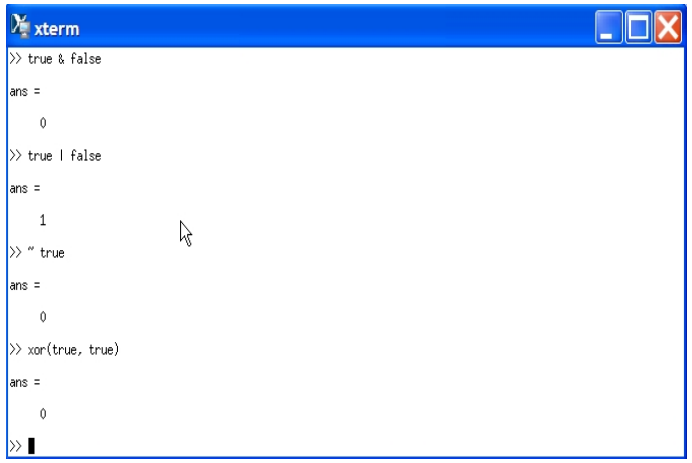


Figure 5: Logical operator usage

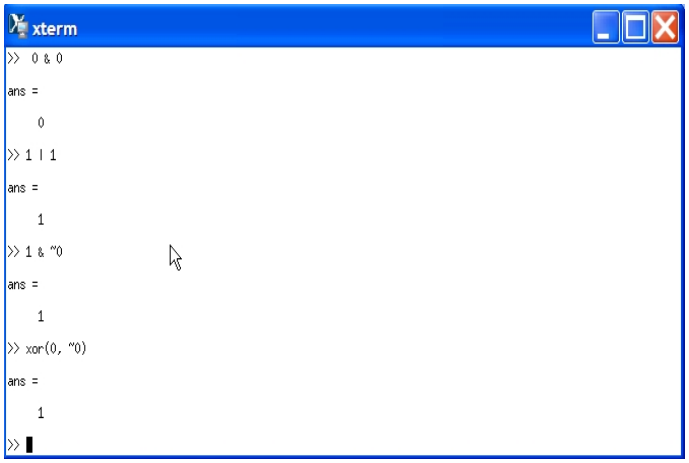


Figure 6: Logical operator arithmetic behavior

In Figure 5 we use the `true` and `false` variables with the logical operators. In Figure 6 we use instead the numerical values of `0,1`. Note that expressions can be complicated. For example 1& ∼) evaluates to 1. This is because 0 evaluates first to 1.

(c) Copyright A. Gerbessiotis. All rights reserved.

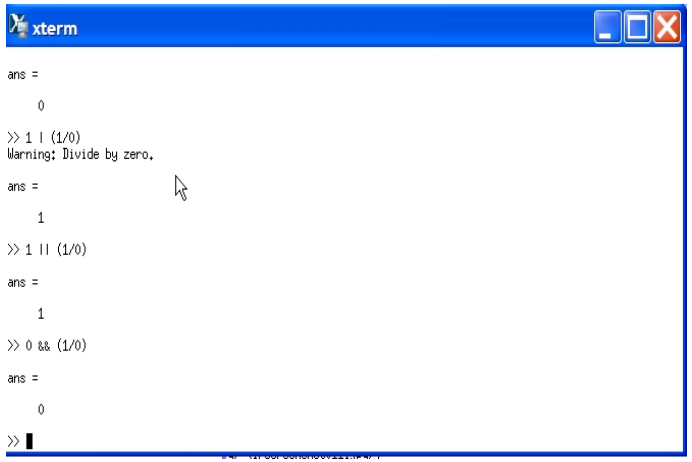


Figure 7: Logical operator usage

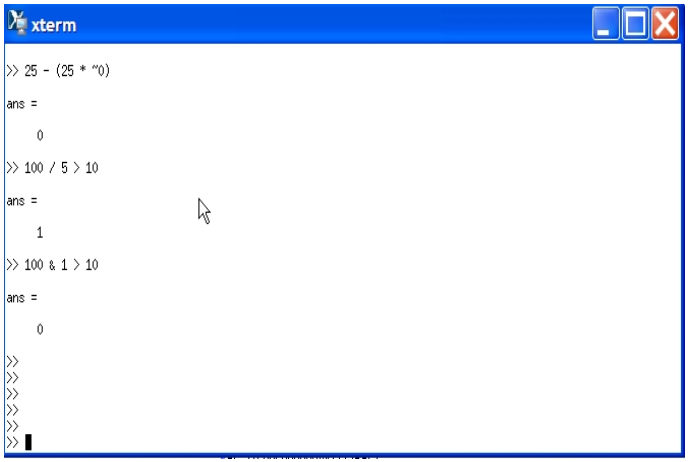


Figure 8: Logical operator arithmetic behavior

In Figure 7 we give some example of how shortcut operators might help, or might cause headaches if you are not careful. In Figure 8, we give some arithmetic examples in which scalar, relational, and logical operators are mixed. **Never ever do something as reckless** as Figure 8; always parenthesize.

Figure 8 shows that it is time to discuss the precedence of the various operators. If for example the same expression has $\&, \&\&, >, <, *$ and no parentheses, which operator is performed first?

(c) Copyright A. Gerbessiotis. All rights reserved.

In MATLAB we are going to use quite frequently strings.

C4.1 String. A **string** is a concatenation of zero or more characters. The string with no characters will be called the NULL string or the empty string and for this discussion we shall use the symbol ϵ to represent it.

C4.2 Length of a string. The length of a string is the number of characters in it or zero if it is ϵ.

C4.3 MATLAB strings. If we want to define a string in MATLAB we use the single-quote character to surround it. Thus `'abcdefgh'` is a string of eight characters, the first eight letters of the alphabet in lower-case. And `''` is the empty string; note that the second quote is immediately after the first with no other space in between. We usually assign a string to a variable to hold it in the memory of a computer and to retrieve it easily by the name of the variable rather than the (quite possibly lengthy) value of it. Thus `a= 'abcdefgh'` assigns string `'abcdefgh'` to variable `a` in MATLAB. It is easier to say or write string `a` rather than `'abcdefgh'`. MATLAB has a `function` that gives us the length of the string; this is helpful if we forgot its length, or we have difficulty counting characters in very long strings. The function `length` returns the length of a string. Thus `length(a)` will return 8 and so will `length('abcdefgh')`. Naturally `length('')`, i.e. the length of the empty string is `0`.

In Figure 9 we define two strings `a, b`. String `a` is as defined above, i.e. `a= 'abcdefgh'`, whereas `b` is the empty string. The two variables used to represent the two strings are stored internally by MATLAB. If we forget their names, we can retrieve all variables defined by us, the user, by typing in MATLAB `whos`. Command (or function) `whos` provides us with information about the variables defined, such as their name, what array dimension they are, their length in characters if they are strings (this is our case), the amount of space used internally by MATLAB for them and some other classification of the variables.

For our case `a` is an 1×8 array (i.e. a row vector). We can hypothesize that a character in MATLAB is stored in two bytes since the total number of bytes used is 16; it is very likely UNICODE is being used to represent characters.

In Figure 10 we show how one case use the `length` function in MATLAB.

If the variables defined are no longer needed, we can clean them up by using the function `clear`. After we use the `clear`, the variables previously defined such as `a,b` are no longer available to us.

C4.4 String storage. Note that MATLAB is using UNICODE characters that require two bytes per character. Thus a string of length 10 characters will require a total of 20 bytes when stored in MATLAB. A `char` data type will be used to store a string in a `char array`.

(c) Copyright A. Gerbessiotis. All rights reserved.

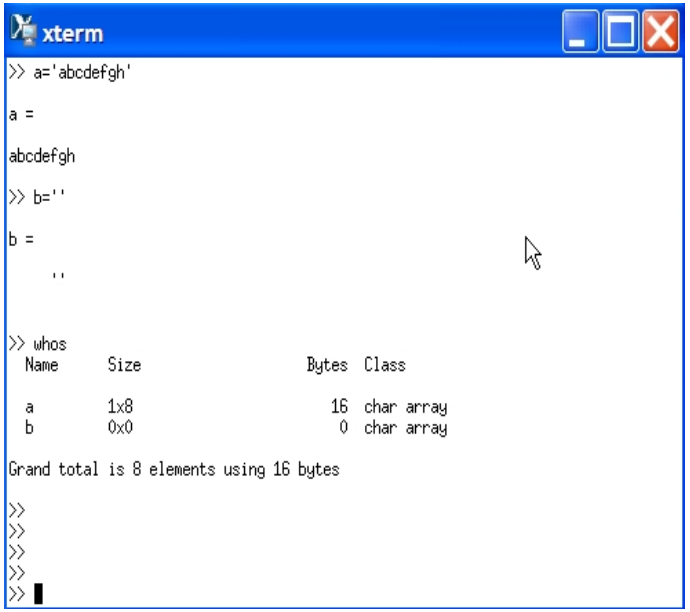


Figure 9: String definitions

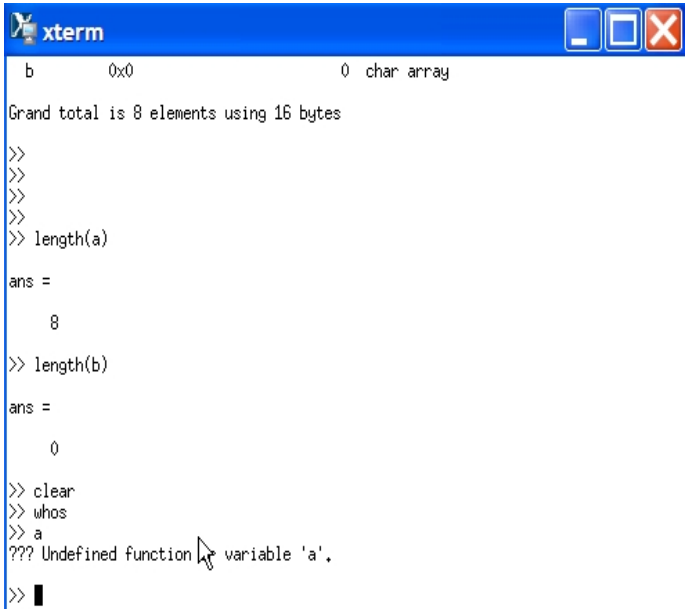


Figure 10: String length and string manipulation

C.4.5 `intr2str` **and** `num2str` The former converts a integer into a string of the appropriate length and the latter an arbitrary number (integer or not) into a string. The length of the string will depend on the digits of the number plus a possible sign (for negative numbers) plus a decimal point for real numbers.

Thus `num2str(-1.234)` will be converted into the MATLAB string '-1.234' of 6 characters stored into a `char array` of 12 bytes. Note that `-1.234` can be stored into a `double` in arithmetic form that requires only 8 bytes!

C4.6 `double(x)` **and** `single(x)` `char(x)`. We can explicitly convert a matrix object from one data type to another. One needs to be careful in interpreting the result. Yet for the example below the weakly-typed nature of MATLAB is exemplified. In the first case a is a logical variable, in the second case it becomes a double, and then a character occupying 1, 8 and 2 bytes respectively! Function `single(x)` converts x into a 4-byte floating-point number as opposed to `double(x)` that converts x into an 8-byte floating-point number.

```
>> a= true;
>> whos
  Name      Size                    Bytes  Class
  a         1x1                         1  logical array
Grand total is 1 element using 1 bytes
>> a=double(a);
>> whos
  Name      Size                    Bytes  Class
  a         1x1                         8  double array
Grand total is 1 element using 8 bytes
>> a=char(a);
>> whos
  Name      Size                    Bytes  Class
  a         1x1                         2  char array
Grand total is 1 element using 2 bytes
```

C5. Cryptic MATLAB data types and care with arithmetic

In MATLAB we can view the internal representation of numbers or strings by using the function `whos` that shows the variables defined their internal data-type plus size information.

C5.1 `char`**.** MATLAB is implemented in the programming language called `C`. Thus it uses data types available in C. The `char` array of a previous example is a `character` array. Thus `char` stands for a C language character. But MATLAB uses UNICODE to present characters. Thus one MATLAB character is 2 byte long and a string of n character uses $2n$ bytes.

C5.2 `double` **and** `single`**.** A `double` is a real number data type in C (and MATLAB). This is the default data type for matrix objects in MATLAB. **NOTE THAT a** `double` **represents only a very small fractions of all real numbers.** Thus very often an approximation of what we have in mind is being used. This for example is the case for π, for which MATLAB uses the variable `pi` that holds an approximation of it to some decimal digits. We can normally view only four of those digits through the default disply (i.e. `format short`) but we can view 15 decimal digits if we choose `format long`. How many decimal digits of π MATLAB stores internally depends on the implementation details of the `double` data type. A `double` is a 64-bit (8-byte) representation of a real number. A `single` is an alternative that uses only 32-bits (4 bytes) for a real number.

C5.3 `logical`**.** The results of relational or logical operator that evalue to a true (in the form of 1) or false (in the form of 0) are stored not in a `double` but in a `logical` data type that uses only 1 bytes.

C5.4 `int8,int16,int32,int64,uint8,uint16,uint32,uint64`**.** MATLAB allows for 4 signed and 4 unsigned integer data type that represent unsigned and two's complement integers in 8, 16,32, and 64 bits.

C5.5 type conversion in MATLAB. If a programmer wants to convert from a given datatype to another-one one can used the name of the data-type to as a function to force this conversion. For example `uint8(10)` will convert MATLAB number 10 (a double) into an 8-bit unsigned integer that occupies only 1 byte in memory, and and `int32(10)` will convert it into a 4-byte 32-bit two'complement signed integer. Note that an `int8` data type can only represent in two's complement the integers from -128 to 127 (inclusive). Thus `int8(128)` and `int8(-129)` will return 127 and -128 respectively, the largest and smallest number that can be represented in 8-bit two's complement.

C5.6 Bonus For the `int, uint` data types of C5.5 use `format hex` to view them in hexadecimal format.

(c) Copyright A. Gerbessiotis. All rights reserved.

C5.3 When is a zero a zero? Note also that when we apply a variety of functions the result is not the real number we have in mind but an approximation of it as well. For example $\sin(pi)$ is the mathematical function of applying the `sine` function to MATLAB's representation (or approximation of) π. Though normally $\sin(\pi)$ is 0, the result given by MATLAB is not zero although it might be very close to it. In my case $\sin(pi)$ returns $1.2246e-16$ which is how MATLAB write 1.2246×10^{-16}. It is a very small, close-to-zero number, but not zero. So beware of real numbers in MATLAB, and how they compare (or not) to zero.

C5.4 Exponential notation. Note that MATLAB has problems raising 10 to powers that need to appear as a superscript to 10. Thus writing 10^{-16} in MATLAB poses a variety of output problems. An easy way is to write it as $1.e-16$ instead, or $1.0e-16$, or $1.E-16$, or $1.0E-16$, where a lower-case `e` or an upper-case `E` might be used.

C5.5 `true` and `false`. The result of a logical (sometimes called boolean operator) is a `true` or `false`. MATLAB stores the result in a `logical` data type that only uses one byte. Thus `1` the number may be stored in a `double` or 8-bytes but `1` for `true` only uses one-byte. Thus `1==1` will give `ans=1` and this would use one byte (do a `whos`). However if it is followed by `ans+0` the arithmetic operator will convert the result into an 8-byte double. For MATLAB `true, false` are aliases for 1 and 0 respectively; most of the time MATLAB operates with these numeric values rather than `true,false`.

(c) Copyright A. Gerbessiotis. All rights reserved.

C6. Other functions available in MATLAB

A set of functions that are available in MATLAB for the benefit of the user includes the following. A partial list is given that includes some common mathematical functions, rounding functions and string related functions beyond those discussed previously

```
Mathematical functions
  abs(x)                                : absolute value of x ,  [abs(-5)=5, abs(5)=5]
  sin(x),  cos(x),  tan(x)              : Trigonometric functions
 asin(x), acos(x), atan(x), atan2(x)    : Inverse trigonometric functions
  exp(x), log(x)                        : Exponential and Logarithmic functions
  sqrt(x)                               : Square root of x
  mod(x,y)                              : Integer remainder of division of x by y  [mod(5,3)=2]
  floor(x)                              : largest integer <= x  [floor(3.1)=3, floor(-3.1)=-4]
  ceil(x)                               : smallest integer >=x  [ceil(3.1)=4,ceil(-3.1)=-3]
  fix(x)                                : Closest (towards zero) integer to x.
                                          [fix(-3.5)=-3, fix(-3.1)=-3, fix(4.9)=4]
  round(x)                              : rounds x to nearest integer [round(-3.4)=-3,round(-3.5)=-4,
                                          round(3.5)=4,round(3.4)=3]

String functions
  int2str(x)                            : Converts x into an integer character string
  num2str(x)                            : Converts x into a character string
  str2num(x)                            : Converts character string s into a numeric array
  char(x)                               : Creates char array (string) of array of (say ASCII) codes.
  double(x)                             : Converts x into double (if already double, no effect)

Vector functions
[value,index ] = min(x)         % <- The percent indicates a MATLAB comment; the rest of the line is ignored
[value,index ] = max(x)         % min(x),max(x) return two value, the maximum/minimum and the index of x
                                % in which it occurs
```

(c) Copyright A. Gerbessiotis. All rights reserved.

Name	Type	Description
+	Scalar?	Addition
−	Scalar?	Subtraction
*	Scalar?	Multiplication
/	Scalar?	Division (Forward)
\	Scalar?	Division (Backward)
^	Scalar?	Exponentiation
<	Relational	Less-than
>	Relational	Greater-than
<=	Relational	Less-than-or-equal-to
>=	Relational	Greater-than-or-equal-to
~=	Relational	Not-equal-to
==	Relational	Equal-to
&	Logical	Logical-AND
|	Logical	Logical-OR(inclusive)
`xor`	Logical	Logical-OR(exclusive)
~	Logical	Logical-NOT
&&	Logical	Logical-AND-shortcut
||	Logical	Logical-OR-shortcut
'	String	String-begin
'	String	String-end
`length`	String	String-length
`whos`	Command	List of user variables
`clear`	Command	Clear user variables
`char`	Internal	Internal data type (character)
`int`	Internal	Internal data type (integer)
`double`	Internal	Internal data type (64-bit real)

Figure 11: MATLAB Operators or Functions

Note. Scalar? operators means that ordinarily these operators are intended for scalars. If MATLAB overloads them and (thus can apply them to matrices or vectors) is beyond the scope of this discussion. More on this later in the course.

Name	Type	Description
abs(x)	Math.Funct.	absolute value
acos(x)	Math.Funct.	Inverse of cosine
asin(x)	Math.Funct.	Inverse of sine
atan(x)	Math.Funct.	Inverse of tangent
atan2(x)	Math.Funct.	Inverse of tangent(angle restriction)
ceil(x)	Math.Funct.	ceiling
cos(x)	Math.Funct.	Cosine
exp(x)	Math.Funct.	Exponential e**x
fix(x)	Math.Funct.	closest to x (towards zero)
floor(x)	Math.Funct.	floor
log(x)	Math.Funct.	Logarithmic base e
mod(x,y)	Math.Funct.	modulo(x,y)=r, x=qy+r 0¡=r¡y
round(x)	Math.Funct.	rounds x
sin(x)	Math.Funct.	Sine
sqrtx)	Math.Funct.	Square root
tan(x)	Math.Funct.	Tangent
int2str(x)	String	Converts x into an integer character string
num2str(x)	String	Converts x into a character string
str2num(x)	String	Converts string x into a numeric array
[value,index] = max(x)	Vector	Find value and location of maximum in vector x
[value,index] = min(x)	Vector	Find value and location of minimum in vector x

Figure 12: MATLAB Operators or Functions (continued)

(c) Copyright A. Gerbessiotis. All rights reserved.

C8.1 `Inf` and `NaN` and function isnan(). The sumbol `Inf` denotes infinity in MATLAB and `NaN` denotes a `Not a Number` in MATLAB. Logical function `isnan()` can verify whether a number is indeed a number or NOT!.

The definition of `isnan` can be shown through an example. The result of `isnan(Inf*0)` returns true, because infinity times 0 is undefined Think of n and $1/n$ as n goes to ∞ and also n^2 and $1/n$ or n and $1/n^2$ and multiply the pairs together. The result can be a constant, ∞ or zero.

C8.2 computations involving `NaN, Inf.` The $\sqrt{NaN}$ is `NaN`. `Inf+(-Inf)` is `NaN`. Multiplication `0*Nan=NaN` and $0/0$ and `Inf/Inf` is also `NaN`.

C8.3 Comparing `NaN` and `NaN`. For this reason, the operation `NaN == NaN` will return a false (0).

C8.4 Square roots of -1. As a reminder, we also point out that $i * i = j * j = i * j = -1$.

C8.5 Is sin(π) equal to zero and how about `sin(pi)`? Although the answer is YES for the former, the internal representation of `pi` is such that that `sin(pi)` gives (at least on my machine) a `1.22e-16` i.e. 1.22×10^{-16}.

C8.6 Is $1.00000000000000001 == 1$**?** Yes it is!

C8.7 Examples. `NaN == NaN` is 0 (false) but $5/0 == 1/0$ is true (1) and be careful with a $5/0 + 3/0 == 4/0$ which is also true (despite the Divide by zero warnings the result evaluates to true).

(c) Copyright A. Gerbessiotis. All rights reserved.

C8. Precedence of operations

An expression in MATLAB is evaluated by following the rules below in the order specified. The rules are listed in order of precedence, higher-order rules are listed first and thus resolved first; if multiple instances of the same or equivalent operator appear, the resolution occurs in a left-to-right fashion with a left-most operator resolved (i.e. evaluated) first. **CAUTION. In (high) school you learned that divisions were to be performed before multiplications! This does not apply to MATLAB. They are at the same level, i.e. they have the same precedence.**

1. In an expression all parenthesized subexpressions are resolved (i.e. evaluated) first starting with the innermost and proceeding to the outermost of nested parenthesized expressions.
2. After parenthesized expressions are resolved, then expressions involving exponentiation (^) or transposition ($'$) are resolved left-to-right.
3. Then expressions with unary plus(+), minus(-) or LOGICAL-NOT $\sim$ are resolved left-to-right.
4. Then expressions with $*, /, \backslash, .\backslash, ./, .*$ are resolved left-to-right.
5. Then expressions with $+, -$ are resolved left-to-right.
6. Then the colon operator (:) is resolved.
7. Then expressions involving relational operators such as $<, >, <=, >=, \sim=, ==$ are resolved left-to-right.
8. Then expressions involving the LOGICAL-AND & are resolved left-to-right.
9. Then expressions involving the LOGICAL-OR, LOGICAL-XOR $|$, `xor`
10. Then expressions involving the short-circuit LOGICAL-AND && are resolved left-to-right.
11. Then expressions involving the short-circuit LOGICAL-OR || are resolved left-to-right.

CAUTION. We called several of the operators of this page scalar operators. The truth that we have hidden (intentionally) is that these operators can also apply to vectors and other matrices. This is what we call in computer-science `operator overloading`. We will discuss more about using these operators on non-scalar arguments later in this course.

Examples.

The expression 2^3^4 is 4096 since (2^3)^4 = 8^4 = 4096. The expression 1&1 > 10 is 0 since 1&1 > 10 = 1&(1 > 10) = 1&0 = 0, with 0 signifying the value for a `false`. The result 2^(3^4) is a very long number. The only way for MATLAB to represent that result is by default as a real number and a result `2.4179e+24` might be output. Without the explicit parenthesization, a 4096 would have resulted as an answer.

(c) Copyright A. Gerbessiotis. All rights reserved.