Computer Science Class

Master ENG/ENV/TECH

Lab and Homework 4

2018

General instruction for all lab

See resource Assignment in https://hippocampus.ec-nantes.fr/ "Labs" section of M1_M_ENG_PROGR course for due date and delivery.

Your work consists of sources answering exercises given below. They must be provided as individual text files following the naming convention of your subject. For each exercise a table, called 'requested information table' hereafter, sets precisely what is expected in terms of naming convention, precises order of input data for the programs you create and output format of these programs results (see chapter 4.4 for complete explanations). You are expected to upload an archive containing all your work in resource Assignments of course "Labs" section of Pedagogic server (no individual file will be allowed).

An auto evaluation tool called auto_eval.bash (see chapter 4.1, 4.2, and 4.3 for details) given with this document, has to be used to generate this archive. It can also be used to auto evaluate your archive before submission and check partially the correctness of your work.

The same auto evaluation tool will be used by the teachers to evaluate your work.

Failure to return the sources archive on time on the server will result in a grade of zero for this homework.

Uploading an archive with a wrong structure or name (i.e. archive not readable or not named correctly) will result in a grade of zero for this homework.

Note : Some useful tricks are given in chapter 4.5.

1 Template function

In lab 1, the computation of π with a trapezoidal rule may be implemented using a rather general function like it is proposed in the correction. It has been isolated in the given *trapezoidal.cc* and *trapezoidal.h* files. In this implementation we see that *trapez_rule* depends explicitly on *unitCircle* (see the given *trapezoidal.cc* file). We want to expand *trapez_rule* to make it compute the trapezoidal rule of any user given function. For that you will use template programming.

1.1 Function object

A first simple solution is to add a 4^{th} argument to the initial function to pass a function object (see course C6 slide "Function objects"). Follow the track :

1. Copy the given *trapez_rule* in a *trapTemplate.h* (declaration), and *trapTemplate_imp.h* (implementation).
2. Change the name of the function to *trapTemplate*
3. Add a 4^{th} argument, that we can call *func*, to *trapTemplate* that will be a constant reference to an object. This object *func* will be of generic type. You must transform *trapTemplate* into a function template to have *func* generic.
4. This 4^{th} argument object will be consider to have at least a
"double operator ()(const double &x)const"
method that take a "double" as argument and return the result of the computation of $f(x)$. f is the function that this object represent. This is the only requirement on generic type *func*. This kind of object are called "function object".
5. Use this *func* instance in *trapTemplate* implementation in place of *unitCircle* calls. You will keep all the algorithmic part from the original function unchanged.
6. To test the new *trapTemplate* template function you will first have to create in *objFunc.cc* and declare in *objFunc.h* a set of object functions representing f corresponding to :

Name of the class/struct	$f(x)$ mathematical definition
unitCircleOF	$4\sqrt{1-x^2}$
lineOF	x
trigoOF	$sin(15 \times x) + cos(3 \times x)$
bilineOF	$\begin{cases} x & if x < 0.5 \\ 1-x & if x \geqslant 0.5 \end{cases}$

7. You must then test your work with the main program given in the file *test_trapTemplate.cc*. Whatever implementation you choose for these function and object functions, this program should give the appropriate results without modification of *test_trapTemplate.cc* source file.

	Requested information
Files	trapTemplate.h, trapTemplate_imp.h, objFunc.h, objFunc.cc
Input	
Output	

Scaling : 1/5

Here is the expected output :

```
Give number of interval
5
PI : 3.03705
ERRORPI : 0.0332773
LINE : 0.5
ERRORLINE : 0
TRIGO : 0.0580991
ERRORTRIGO : 0.646497
BILINE : 0.24
ERRORBILINE : 0.04
```

Optional :

For those who want to go further you can modify the *main* in *test_trapTemplate.cc* to use *lambda function* instead of function object (see C6). All *objFunc.cc*, *objFunc.h* and all their class/struct (*unitCircleOF*, *lineOF*, *trigoOF* and *bilineOF*) have to be replaced by *lambda functions* in *test_trapTemplate.cc* . You will have to compile with "−std=c++11" option.

1.2 Generalisation to every arithmetic types

Imagine that we want to use another arithmetic or precision to compute the trapezoidal rule. Says instead of "double" real we want to use for example the ***Real*** type (from Lab 3). Generalization of the initial *trapez_rule* function is now not only related to the use of any f function but also to the use of any arithmetic. Follow the track :

1. Copy the given *trapez_rule* in a *trapTemplateGeneral.h* (declaration), and *trapTemplateGeneral_imp.h* (implementation).
2. Change the name of the function to *trapTemplateGeneral*
3. Add a 4^{th} argument, that we can call *func*, to *trapTemplateGeneral* that will be a reference to a function that takes one "const double" reference as unique argument and returns a "double" (see course C2 slide "Pointers to functions" where you can replace pointers by references). The exact syntax is the following :
   "double (&func)(const double &)"
   Use this *func* variable name (representing a function) in the implementation in place of *unitCircle* calls. You will keep all the algorithmic part from the original function unchanged. Note that this gives another solution than function objects and template functions to the problem of treating any f user given function. You can do some partial tests. Normally if you call this new function with *unitCircle* (and 0.,1.,n for first 3 arguments) you should get a result approaching π.
4. Now continue the generalization by making *trapTemplateGeneral* a template function on the arithmetic type used for the computation. Says your *trapTemplateGeneral* might be able to integrate a function f dealing with int,float,double or ... Modify all the implementation according to this new template parameter representing generic arithmetic type.
5. To test the new *trapTemplateGenaral* template function, you will first have to create in *func.cc* and declare in *func.h* a set of function representing f corresponding to :

Name of the function	$f(x)$ mathematical definition	Arithmetic
unitCircle (you can copy the version from *trapezoidal.cc*)	$4\sqrt{1-x^2}$	double
line	x	integer
trigo	$sin(15 \times x) + cos(3 \times x)$	double
biline	$\begin{cases} x & if x < 0.5 \\ 1-x & if x \geqslant 0.5 \end{cases}$	float

6. You must then test your work with the main program given in file *test_GtrapTemplate.cc*. Proceed step by step. First comment out all the tests and includes related to other arithmetics than int, float and double. Test. If you are successful pass to the next step.

7. Starting from the given *Starting_Real1.h* and *Starting_Real1.cc* (same as lab 2 correction) try now to compute the integral of a *rline* function in ***Real*** arithmetic. For that use now the full original version of *test_GtrapTemplate.cc*. Add a function *rline* to *func.cc* and *func.h* that does the same as *line* but in ***Real*** arithmetic. Copy *Starting_Real1.h* and *Starting_Real1.cc* files in *Real1.h* and *Real1.cc* respectively and try to compile. If it does not compile modify class ***Real*** so that it fulfills *trapTemplateGeneral*,*rline* and '*test_GtrapTemplate*' requirements. Whatever implementation you choose for *trapTemplateGeneral* function and testing functions, this program should give the appropriate results without modification of *test_GtrapTemplate.cc* source file.

	Requested information
Files	trapTemplateGeneral.h, trapTemplateGeneral_imp.h, func.h, func.cc, Real1.cc, Real1.h
Input	
Output	

Scaling : 1/5

Here is the expected output :

```
Give number of interval
5
LINE : 0
BILINE : 0.24
ERRORBILINE : 0.04
PI : 3.03705
ERRORPI : 0.0332773
TRIGO : 0.0580991
ERRORTRIGO : 0.646497
RLINE : 0.5E40
ERRORRLINE : 0.1E1
```

Optional :

For those who want to go further you may try to use two template parameters to do this question: one like question 1.1 to represent the function type (may be used with function object or *lambda function*) and one like in this exercise that represent the arithmetic used for computation. Ask yourself why this solution may be unsafe ?

2 Template Class

For this exercise, you will come back to the polynomial class created during the previous lab. A new implementation design, based on the use of two member variables is now mandatory for this exercise (this is somehow a kind of mixing of ***Vector*** class into ***Poly*** class). Those two members define your polynomial coefficients and the order of the polynomial. To avoid losing time on this specific reshaping task header file (*Poly.h*) and implementation file (*Poly.cc*) are given as a starting point.

You must expand this class to make it a template class on the space your polynomial works on. Up to now you could only compute polynomial of variables represented as double, with double coefficients. Change ***Poly*** class so that it becomes a template class, in order to compute polynomials of any number type with any coefficients type (but number and coefficients are of the same type). The objective is to test the new version of ***Poly*** class with the class ***Real*** that you have defined in previous lab.

Follow the track :

1. Implement your template version of ***Poly*** class. You must start from ***Poly*** class defined in *Poly.h*, keeping all the interface functionality unchanged and number of members unchanged. Store your declaration in *PolyTemplate.h* and your implementation in *PolyTemplate_imp.h*.

2. Before any attempt to use special type as template parameter, test double type template parameter and verify that results (computed values, computed coefficients, output format) are the same as those given by non template class. For that use *test_PolyTemplate_d.cc* ; using the following data, you should get the following output :

```
 === Testing template Poly class for double ===
Input degree of p1d: 3
Input coefficients of p1d:
1
0
3
2
Input degree of p2d: 6
Input coefficients of p2d:
2
3
0
0
6
1
5
This is not possible : You cannot define a polynome of order 4 with a null coeficient for mon
Error: trying to create a polynomial of order n with null coefficient of order n
 Test print():
p1d: poly(x)=1+3*x^2+2*x^3
p2d: poly(x)=2+3*x^1+6*x^4+1*x^5+5*x^6
 Test addition:
p1d+p2d: poly(x)=3+3*x^1+3*x^2+2*x^3+6*x^4+1*x^5+5*x^6
p2d+p1d: poly(x)=3+3*x^1+3*x^2+2*x^3+6*x^4+1*x^5+5*x^6
 Test substraction:
p1d-p2d: poly(x)=-1-3*x^1+3*x^2+2*x^3-6*x^4-1*x^5-5*x^6
p2d-p1d: poly(x)=1+3*x^1-3*x^2-2*x^3+6*x^4+1*x^5+5*x^6
 Test multiplication:
p1d*p2d: poly(x)=2+3*x^1+6*x^2+13*x^3+12*x^4+1*x^5+23*x^6+15*x^7+17*x^8+10*x^9
p2d*p1d: poly(x)=2+3*x^1+6*x^2+13*x^3+12*x^4+1*x^5+23*x^6+15*x^7+17*x^8+10*x^9
 Test euclidian division:
Q=(p2d*p1d)/p1d=p2d: poly(x)=2+3*x^1+6*x^4+1*x^5+5*x^6
Q=p2d/p1d: poly(x)=-13.0625+7.875*x^1-3.25*x^2+2.5*x^3
R=p2d-(p2d*p1d)/p1d=0: poly(x)=0
R=p2d-p2d/p1d: poly(x)=15.0625-4.875*x^1+42.4375*x^2
Testing () operator
First try, input a value for x1:
1.5
p1d(x1): 14.5
```

```
p2d(x1): 101.422
Second try, input a value for x2:
2
p1d(x2): 29
p2d(x2): 456
```

3. Then try ***Real*** class type with *test_PolyTemplate_r.cc*. Use for that the class files given in previous exercise (*Starting_Real1.cc* and *Starting_Real1.h*) or the one you provide for it (*Real1.cc* and *Real1.h*). Copy them in *Real2.cc* and *Real2.h*.

   Normally, independently on how you transform Poly class, you should not be able to compile the program. Make up your reflection on that problem, and avoid them by modify ***Real*** class so that function member of template class ***Poly*** "works" without algorithmic modification. If you don't modify ***Real*** class and you pass compilation stage successfully, it means that you changed too much the initial ***Poly*** class. It certainly won't do things the way the initial template free class ***Poly*** did. Please revert to initial class and start again from this point.

   Using *test_PolyTemplate_r.cc* with the following data, you should get the following output :

```
 === Testing template Poly class for Real numbers ===
Input degree of p1r: 1
Input coefficients of p1r (mantissa first, then exponent):
1
0

2
0

Input degree of p2r: 2
Input coefficients of p2r (mantissa first, then exponent):
0
0

1
0

2
5

This is not possible : You cannot define a polynome of order 2 with a null coeficient for mon
Error: trying to create a polynomial of order n with null coefficient of order n
 Test print():
p1r: poly(x)=0.1E1+0.2E1*x^1
p2r: poly(x)=+0.1E1*x^1+0.2E6*x^2
 Test addition:
p1r+p2r: poly(x)=0.1E1+0.3E1*x^1+0.2E6*x^2
p2r+p1r: poly(x)=0.1E1+0.3E1*x^1+0.2E6*x^2
 Test substraction:
p1r-p2r: poly(x)=0.1E1+0.1E1*x^1-0.2E6*x^2
p2r-p1r: poly(x)=-0.1E1-0.1E1*x^1+0.2E6*x^2
 Test multiplication:
p1r*p2r: poly(x)=+0.1E1*x^1+0.200002E6*x^2+0.4E6*x^3
p2r*p1r: poly(x)=+0.1E1*x^1+0.200002E6*x^2+0.4E6*x^3
 Test euclidian division:
Q=(p2r*p1r)/p1r=p2r: poly(x)=+0.1E1*x^1+0.2E6*x^2
Q=p2r/p1r: poly(x)=-0.499995E5+0.1E6*x^1
```

```
R=p2r-(p2r*p1r)/p1r=0: poly(x)=0E0
R=p2r-p2r/p1r: poly(x)=0.499995E5+0.100999E-9*x^1
Testing () operator
First try, input a value for x1 (mantissa first, then exponent):
4
658
p1r(x1): 0.8E659
p2r(x1): 0.32E1323
Second try, input a value for x2 (mantissa first, then exponent):
5
-2
p1r(x2): 0.11E1
p2r(x2): 0.50005E3
```

At the end, with *PolyTemplate.h*, and *PolyTemplate_imp.h* files (where respectively you declare and implement template version of the class ***Poly***), whatever implementation you chose for this class ***Poly***, '*test_PolyTemplate_d*' and '*test_PolyTemplate_r*' programs with the data test files (auto evaluation tools or output above) should give the appropriate results without modification of *test_PolyTemplate_d.cc* and *test_PolyTemplate_r.cc* source files.

	Requested information
Files	PolyTemplate.h, PolyTemplate_imp.h, Real2.h, Real2.cc
Input	
Output	

Scaling : 2/5

3 STL

Based on previous exercise, and the implementation of the class polynomial of previous lab (the correction), create a new template version of class polynomial, with appropriate STL container as your **unique** member variable to store polynomial coefficients (of any kind) and polynomial order.

Only generate the interface (i.e. the declaration in a header) that looks the more logical to you. Put this declaration in *PolytemplateSTL.h*

	Requested information
Files	PolyTemplateSTL.h
Input	
Output	

Scaling : 1/5

Optional :

For those who wish to verify their choice, do the implementation part (*PolyTemplateSTL_imp.h*) and compile with *test_PolyTemplateSTL.cc* which differ from *test_PolyTemplate_d.cc* and *test_PolyTemplate_r.cc* by included file and the way it test all types via a template function.

You may play same kind of test as exercise 2 with this new executable (you can use the dat file has input). If you switch back to implementation of exercise 2 by including *PolyTemplate.h* in

replacement of inclusion of *PolyTemplateSTL.h*, you will see that results are the same with same set of input. This illustrates the fact that only public interface count for a user. Implementation is hidden and your main "doesn't care" that polynomial class use STL or anything else.

4 Appendix

4.1 Auto_eval.bash limits

This tool is just a help to verify mainly that:

- Your program is located in correct source files having correct names.
- No file is missing. If so you will be able to pass through all steps but your points for the missing file will be lost.
- All programs, at least, compile and run correctly (to obtain the 2 points see 4.2).
- With the student parameter setting, it gives correct results (to obtain the 6 points see 4.2).
- The archive given to teacher has the correct structure and name.

What this tool does not verify is that:

- Your programs are correct in absolute. Be careful auto_eval.bash is tuned to have exercises running with a set of parameters but a different setting will be used by the teacher to correct your work. This means that your code may work with auto_eval.bash and the student parameter setting but not with the teacher parameter setting. It is your responsibility to ensure that your program is correct and works with any setting.
- Your programs are correct if you just generate an archive and didn't pass through evaluation step.
- You give your correct group identification number. If you generated an archive following a wrong group ID and upload it as it is then you will have a grade of zero for this work.

4.2 Evaluation rule

The auto evaluation tool auto_eval.bash will be used by teachers (in a slightly modified version) to evaluate your work with the following rule:

- For an exercise the mandatory program doesn't compile => 0 points for the exercise.
- For an exercise the mandatory program compiles but doesn't execute correctly (i.e. crash,stop in the middle ...) => 1 point.
- For an exercise the mandatory program compiles and executes without bug but doesn't give correct results => 2 points.
- For an exercise the mandatory program compiles, executes without bug and gives correct results/behavior => 6 points.

Naturally when you are in position of having only 1 or 2 points, the program will be inspected to see if it is relevant (answering the question). If not (something that has nothing to do with the exercise but compiles and runs correctly) => 0 points.

For every exercise an extra scaling is done. After each requested information table (see 4.4) you will have the coefficient used for this scaling.

By using auto_eval.bash you will be able to estimate how many source evaluation points you will have for this lab without scaling.

4.3 Getting and using auto_eval.bash

auto_eval.bash like this pdf files comes from self extracting file labX.bash that you download from resource "Lx subject" in https://hippocampus.ec-nantes.fr "Labs" section of M1_M-ENG_PROGR course, where X corresponds to the current lab number.

Place auto_eval.bash in the folder where you have all exercises sources that have to be given to teachers. Then in this folder just type:

```
··> ./auto_eval.bash
```

The first step is to input your group ID (letter A, B or C and a number):

```
../auto_eval.sh

================================================================================
Setting your lab group
================================================================================

Please what is your group letter ?
Type your choice ( A B C ) :A


Please what is your group number ?
Type your choice ( 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 ) :01
```

Input your group letter and number

If you launch the script for the first time, you have to create an archive of your work. Select "Prepare an archive of your work".

```
Please what is your group number ?
Type your choice ( 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 ) :01



//////////////////////////////////////////////////////
///   This is auto_eval version 0.1 for LAB 1 2014
//////////////////////////////////////////////////////
/// Current status
///   No archive file ./archive_lab1_A_01.tar.gz present
//////////////////////////////////////////////////////

What do you want to do ?

    a) prepare an archive of your work.
    q) stop using this program

Type your choice ( a q ) :a



================================================================================
Archiving file for lab 1
================================================================================

File 'bary_triangle.cc' correctly archived
File 'mathfunction.h' correctly archived
File 'mathfunction.cc' correctly archived
File 'test_mathfunction.cc' correctly archived
File 'Poly.h' correctly archived
File 'Vector.h' correctly archived
File 'ex1.cc' correctly archived
File 'Poly.cc' correctly archived
File 'Vector.cc' correctly archived
Compressing archive

================================================================================
Archive ./archive_lab1_A_01.tar.gz ready to be auto evaluate or uploaded on the server.
================================================================================
```

Create archive

A *.gz* archive file is created, and ready to be uploaded on the pedagogic server. If you want, at this point you can stop the program (Type "q"). Note that theses two steps at least are mandatory to create the archive of your work that you will upload on the server. You can create this archive by yourself (for instance using the *tar* command), but if there is any kind of problem when we try to extract it, your grade may be affected with no protestation possible (the most stupid problem

would be that your archive doesn't have the correct name which leads to ... zero for the whole lab evaluation) .

Otherwise, you can use this tool to get an auto-evaluation of your work. Now that your archive is created, you can notice that a new command appeared, "evaluate archive already generated". Type "b".

```
////////////////////////////////////////////////////////
///   This is auto_eval version 0.1 for LAB 1 2014
////////////////////////////////////////////////////////
/// Current status
/// Archive file ./archive_lab1_A_01.tar.gz present
////////////////////////////////////////////////////////

What do you want to do ?

    a) prepare an archive of your work.
    b) evaluate archive already generated.
    q) stop using this program

Type your choice ( a b q ) :b
```

Auto-evaluate your work

Some information are displayed on the screen, saying that the auto-evaluation worked through well (or not). The most interesting information for you are the ones on the next screen-shot.

```
================================================================================
Compile archive ./archive_lab1_A_01.tar.gz
================================================================================

Program bary_triangle compile correctly
test_mathfunction.cc compile correctly
mathfunction.cc compile correctly
Program test_mathfunction compile correctly
ex1.cc compile correctly
Poly.cc compile correctly
Vector.cc compile correctly
Program ex1 compile correctly

================================================================================
Run archive ./archive_lab1_A_01.tar.gz
================================================================================

Program bary_triangle run correctly with bary_triangle_00.dat
Program test_mathfunction run correctly with test_mathfunction_00.dat
Program ex1 run correctly with ex1_00.dat

================================================================================
Evaluate result of archive ./archive_lab1_A_01.tar.gz
================================================================================

=> For input parameter setting, program 'bary_triangle' give 3 out of 3 correct result(s).
=> For input parameter setting, program 'test_mathfunction' give 2 out of 2 correct result(s).
For ex1_00.dat input parameter setting, program 'ex1' don't give correct result for p2 result.
=> For input parameter setting, program 'ex1' doesn't give any correct result from 1 expected.

////////////////////////////////////////////////////////
///   This is auto_eval version 0.1 for LAB 1 2014
////////////////////////////////////////////////////////
/// Current status
/// Archive file ./archive_lab1_A_01.tar.gz present
/// Archive unpacked successfully
/// 3 out of 3 program compile successfully
/// 3 out of 3 program run successfully
/// 5.0 out of 6 correct results where obtain
////////////////////////////////////////////////////////
```

Results of your auto-evaluation

Here you can see:

- Which of your source codes compile correctly.
- Which of your programs run correctly.
- Which of your programs give the expected results.

This way you can have an estimation of your grade according to the rules detailed in section 4.2. However keep in mind that the grade estimated using this tool may not be your final grade (see section 4.1). Note that you can use this command only if you have generated an archive first.

You could see that in the previous example, one of the results given by program "ex1" is incorrect. To display the difference between your results and the expected one, select the new command "display difference from solution of last generated archive" by typing "c".

```
=> For input parameter setting, program 'bary_triangle' give 3 out of 3 correct result(s).
=> For input parameter setting, program 'test_mathfunction' give 2 out of 2 correct result(s).
For ex1_00.dat input parameter setting, program 'ex1' don't give correct result for p2 result.
=> For input parameter setting, program 'ex1' doesn't give any correct result from 1 expected.

////////////////////////////////////////////////////
///   This is auto_eval version 0.1 for LAB 1 2014
////////////////////////////////////////////////////
/// Current status
/// Archive file ./archive_lab1_A_01.tar.gz present
/// Archive unpacked successfully
/// 3 out of 3 program compile successfully
/// 3 out of 3 program run successfully
/// 5.0 out of 6 correct results where obtain
////////////////////////////////////////////////////

What do you want to do ?

    a) prepare an archive of your work.
    b) evaluate archive already generated.
    c) display difference from solution of last evaluated archive
    q) stop using this program

Type your choice ( a b c q ) :c



================================================================================
Display difference from last evaluation of archive ./archive_lab1_A_01.tar.gz
================================================================================

ex1 00 p2 :
yours    >  poly(x)= +0.5*x^2
solution >  poly(x)=  0.5*x^2
```

Compare obtained results and expected results

Now you can:

- Generate a new archive of your work, if you modified some of your source files.
- Auto-evaluate the current archive.
- Compare the results obtained with your program and the solution.
- Stop using the program.

In the last case, if you decide to quit the program, you will be asked whether you want to clean or save a certain temporary folder. This temporary folder was generated during the auto-evaluation step, and can possibly be used for debugging even if the script is not launched. Those of you who are the most at ease with programming may want to use it, otherwise you can just delete it.

```
//////////////////////////////////////////////////////
///   This is auto_eval version 0.1 for LAB 1 2014
//////////////////////////////////////////////////////
/// Current status
/// Archive file ./archive_lab1_A_01.tar.gz present
/// Archive unpacked successfully
/// 3 out of 3 program compile successfully
/// 3 out of 3 program run successfully
/// 5.0 out of 6 correct results where obtain
//////////////////////////////////////////////////////

What do you want to do ?

    a) prepare an archive of your work.
    b) evaluate archive already generated.
    c) display difference from solution of last evaluated archive
    q) stop using this program

Type your choice ( a b c q ) :q



================================================================================
Warning : Cleaning
================================================================================

Do you want to clean (c) or save (s) temporary folder /home/users/struct/ble/Enseignement/CPP/AutoEval/lab1/lab1/auto
_eval_dir22261 ?
This folder may contain compilation, execution and evaluation of last evaluated archive.
If you save it, it's your responsability to do the cleaning

Type your choice ( c s ) :s
```

Stop the auto-evaluation program

4.4 Requested information

Tables given at the end of each exercise entitled 'Requested information' contains the following items:

- Files: this is the list of files which must be present in your archive uploaded in the server. auto_eval.bash harvests these files to generate the mandatory archive.
- Input: describes for each program the precise order in which information must be entered.
- Output: describes for each program the output format of the results. **Results (that teachers want to check easily) must be presented by your program in a rather rigid format described by the following**:
  - A result is present on one single line. Nothing else may be written on that line.
  - A result is first identified by a token which starts the line.
  - After the token a ':' must separate the token from result value.
  - After ':' result value must be written (it could be anything).
  - token, ':' and result value must be separated at least by one blank character.

  In this section you have the list of mandatory results given by token names and in parentheses by result name coming from subject.

Here is an example of a '*Lame*' program made of one file *lame.cc*. It is generating 2 output results called λ and μ in the subject. From the subject, this program should ask for 2 input values called E and ν to compute results. The requested information table would look like:

	Requested information
Files	lam.cc
Input	ν then E
Output	MU(μ), LAMBDA(λ)

Scaling : 1/8

'*Lame*' program execution would look like with $E = 2500.$, $\nu = 0.28$ and computed $\mu = 976.5625$ and $\lambda = 1242.8977$:

```
blablabla
please enter nu :
0.28
blabla
please enter E :
2500.
 LAMBDA : 1242.8977
blabla
 MU : 976.5625
blabla
```

Notice in this example that ν is asked before E as mandatory by "Input" directive. And λ and μ appear in undefined order but with mandatory format (i.e token LAMBDA and MU).

Very important. You should avoid at any cost the use of token for anything else than the result it corresponds to. Consequences may result in a zero grade just because you messed up the tools with false information.

In example above if in any 'blabla' you write MU or LAMBDA you will get a zero for associated results.

Last but not least when you enter information and output first a question message you must always add a new line to your question if it precedes a result. Let's take the example of '*Lame*' above which outputs token 'LAMBDA' right after asking to input E. If you don't follow this last recommendation you may ask for E without inserting a new line after the question as you can see in the example below:

Your executable is Lame and on terminal you type:

```
··> ./Lame
```

And obtain the following bunch of outputs:

```
blabla
please enter E : 2500.
LAMBDA : 1242.8977
blabla
```

This works well on the terminal by typing on the keyboard '2500.' and 'enter'. But auto_eval.bash works differently. It uses what is called redirection mechanisms. **And in fact this leads auto_eval.bash to miss recognizing your results.**

In this example you must add a "end of line" after the question to obtain:

```
··> ./Lame
```

```
blabla
please enter E :
2500.
LAMBDA : 1242.8977
blabla
```

and auto_eval.bash will work correctly in this case.

4.5 Useful

During Labs you will have questions where testing programs ask for many inputs. During debugging stage you may then be obliged to re enter those inputs manually many times. That may be error prone and time consuming. To simplify your work use redirection.

For example lets take a '*prg*' program which runs the following:

```
··> prg
```

```
Enter number :
23
```

```
Enter number :
13
Enter number :
-1
Mean value :
18
```

Here for illustration we just limit inputs to three entries. Put all those inputs in a file, with one input per line. Let us call this file "ip.txt". Its contents for this example will be:

```
23
13
-1
```

Now if you type the following command you get the same execution of '*prg*' without typing anything:

```
···> prg < ip.txt
```

```
Enter number :
Enter number :
Enter number :
Mean value :
18
```

Note that inputs just vanish from terminal display.