ESIPAP European School of Instrumentation in Particle and Astroparticle Physics

2016 edition

Supervisers: Eric CHABERT, Eric CONTE

Computing session 2

C++ model for the electromagnetic barrel calorimeter of the CMS (Compact Muon Solenoid) detector

Abstract:
This computing session is dedicated to the first notions of oriented-object programming. The physics topic chosen for the exercise is the electromagnetic calorimeter of the CMS experiment. In a first part, the students are invited to program a C++ model of the calorimeter from UML (Unified Modeling Language) diagrams. The developed code must describe the apparatus geometry, read data acquired by all cells and correct these data with calibration settings. The second part of the session consists in equipping the programming project with a *makefile*-based compilation and with an automatically generated documentation.

Pedagogical goals:

C++ language

- Writing new classes from UML diagrams.
- Instantiating objects from classes and initializing them.
- Reading and adapting an existing piece of code.
- Improving the robustness of the code in order to prevent abnormal termination or unexpected actions.

Collaboration work

- Respecting a given set of programming rules and conventions.
- Generating automatically the reference documentation related to the code with DOXYGEN.

Compiling/linking

- Creating an executable file from a simple source file.
- Compiling and linking a project made up of several source files: in a manual or automated (Makefile) way.

Requirements:

- Concept of class in C++, including constructors, destructor, mutators, accessors, ...
- Some particular C++ points: I/O access, arrays, pointers/references.

Contents

Part I

Introduction to the ESIPAP computing sessions

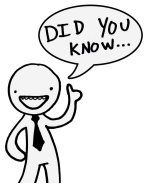
1 Foreword

Computing sessions belong to the educational program of the ESIPAP (European School in Instrumentation for Particle and Astroparticle Physics). Their goal is to teach the secrets of C++ programming through practical work in the context of high energy physics. The session is designed to be pedagogical. It is advised to read this document section-by-section. Indeed, except the *Physics context*, each section of the document is a milestone allowing to acquire computing skills and to validate them. The sections related to C++ programming are ranked in terms of complexity. In order to facilitate the reading of this document and to measure his progress, the student must **fill up the dedicated roadmap** which includes a check-list and empty fields for personal report.

In the document, some graphical tags are used for highlighting some particular points. The list of tags and their description are given below.

The student is invited to perform a pratical work by **writing a piece of code** following some instructions.

Analyzing or interpreting task is requested and the results must be reported in the roadmap.



Some **additional information** is provided for extending the main explanations. It is devoted to curious students.

A piece of **advice** is given to help the student in his task.

2 The ESIPAP framework

The practical works must be performed on devoted machines where all required software are properly installed. The user will find below all the instructions for setting the environment at each beginning of session.

2.1 Launching the `Windows` machine

You must choose a computer in the computing room, spot its name and check that no peripheral is missing (mouse, keyboard, ...). Then boot it and login to the `Windows` operator system (supervisors will provide the password access).

2.2 Accessing the `Linux` virtual machine

The practical sessions will be achieved on a `Linux` machine for pedagogical motivations. You must connect a virtual machine. First click on the "Start" button, i.e. the button with the `Windows` logo, located on the bottom left of the screen (see Figure 1).

Figure 1: The `Windows` Start button

According to Figure 2, click on the virtual machine called **"ESIPAP_slc6"**. A password could be necessary and should be supplied by the supervisors.

Figure 2: The screen showing the available virtual machines

2.3 Setting the environment

To load the work environment, you can issue the command below at the shell prompt.

```
bash$source␣/home/esipap/tools/setup.sh
```

If the system is properly installed, the version of each tool to study should be displayed at the screen like below. If you have an error, please call the supervisors.

```
-----------------------------------------------
              ESIPAP environment
-----------------------------------------------
 - GNU g++   version 4.9.1
 - ROOT      version 6.06/00
 - Geant4    version 10.2.0
-----------------------------------------------
```

You must work in your local folder. Of course, it is advised to create one folder for each practical session like: `session1`, `session2`, `session3` and `session4`. Do not overwrite or remove files that you wrote in a previous session.

2.4 Saving your work on a share disk

Your work will be evaluated from the the piece of code that you wrote. At the end of each session you must save your production on a share disk. The virtual machine is equipped with one share disk called "ESIPAP-SHARE" and saved everyday. For accessing this disk, click on the `Linux` tab named **"places"** according to Figure 3 and select the disk **"ESIPAP-SHARE"**.

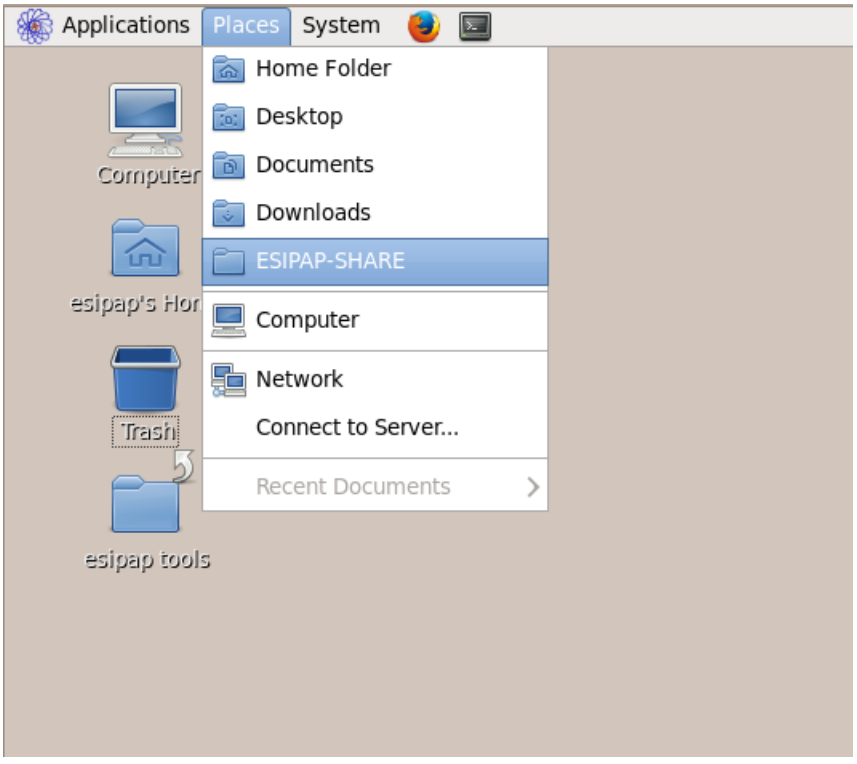


Figure 3: The ***Linux*** *tab named "places"*

After entering a password, the list of all connected machines in the room is displayed (see Figure 4). Select the folder corresponding to your machine and put there all you work. Please organize this folder by creating one folder for each practical session like: `session1`, `session2`, `session3` and `session4`.

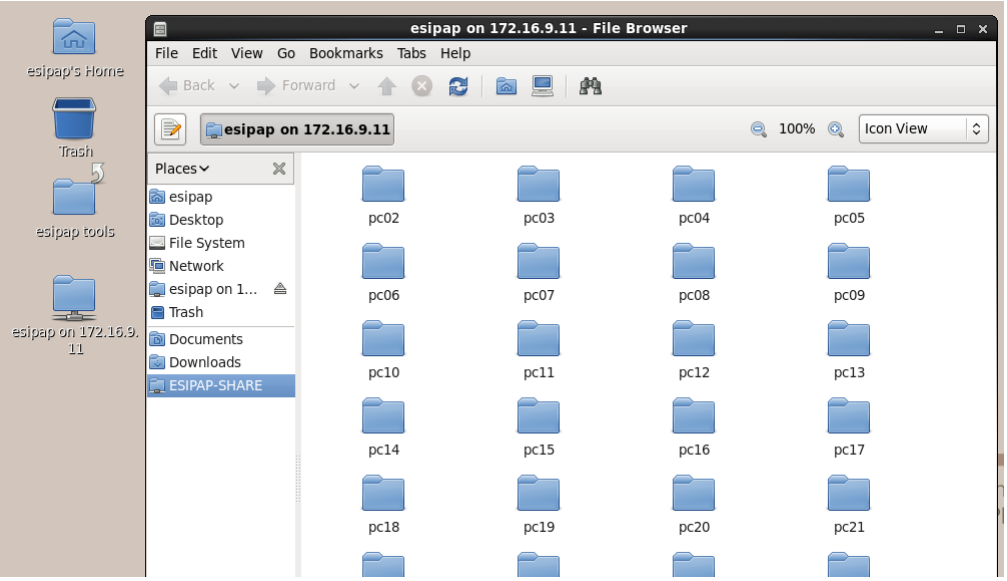


Figure 4: List of all available machines in the room

Part II

C++ model for CMS calorimeters

3 Physics context

3.1 The CMS detector

CMS(Compact Muon Solenoid) is one of the four main detectors build for analyzing particles produced by proton-proton collisions at the LHC (Large Hadron Collider). The detector is buried under about 100m at the point 5 of LHC ring. With a weight of 12500 tons, it has cylinder volume with a diameter of 14.6 m and a length of 21.6 m. The LHC beam cross the detector in its axis and the collisions occur in its middle. CMS is made up of several detector components: a **silicon tracker** equipped with a huge **solenoid magnet**, **electromagnetic and hadronic calorimeters** and finally **ionizing chambers** devoted to muon tracking. The figure below allows to distinguish the different components.

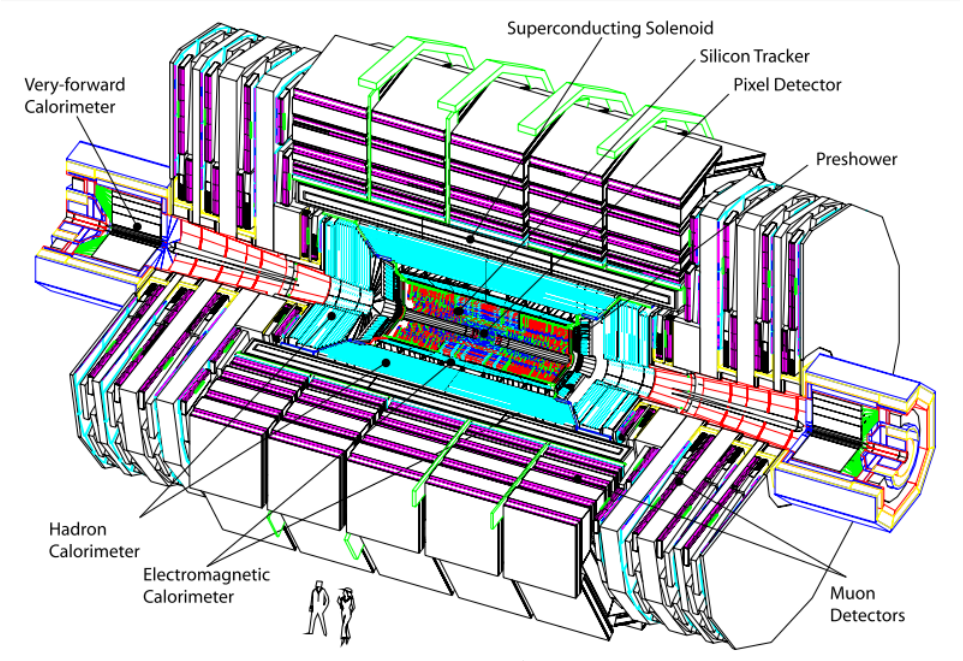


Figure 5: A perspective view of the CMS detector

The first run of data taking has begun since Fall 2008. It is designed to undergo 40 millions of proton-proton collisions per second. All collisions (we speak later in term of *events*) are not interested for the physicists and a **trigger** system selects in real-time the most relevant one. The dataflow is reduced to about 300 collisions per second.

3.2 CMS coordinates systems

It is important to remind the cartesian and cylindrical coordinates systems used in the CMS collaboration. The both coordinate systems has the origin centered at the nominal collision point inside the experiment.

- CARTESIAN. The y-axis pointing vertically upward, and the x-axis pointing radially inward toward the center of the LHC. Thus, the z-axis points along the beam direction toward the Jura mountains from LHC Point 5.
- CYLINDRICAL. The azimuthal angle ϕ is measured from the x-axis in the x-y plane and the radial coordinate in this plane is denoted by r. The polar angle θ is measured from

the z-axis. Pseudorapidity $\eta = -\ln \tan \frac{\theta}{2}$ is usually used instead of θ.

3.3 The electromagnetic calorimeter of the CMS detector

The aim of the electromagnetic calorimeter is to measure the energy of photons and electrons produced during the collisions. At high energies, electromagnetic particles induce electromagnetic shower when they interact with the calorimeter material. Loss energy is converted to light due to scintillating property of the material: lead tungstate (PbWO4) crystals with a short radiation length $X_0 = 0.89$ cm and a short Moliere radius equal to 2.2 cm. The CMS electromagnetic calorimeter is hermetic, homogeneous and compact. It covers the full range in azimuthal angle and the pseudorapidity range $|\eta| < 1.48$. The cells have a size of 22×22 mm^2 at the front face and a length of 230 mm corresponding with 25.8 X_0. The electromagnetic calorimeter is compound of two different geometries:

- the cylinder part, called *barrel*, has a radius of 1.29 cm and contains 61,200 cells.
- the two planes at each end of the cylinder (z=-1 m and z=+1 m), called *end-cap*, contain together 14,648 cells.

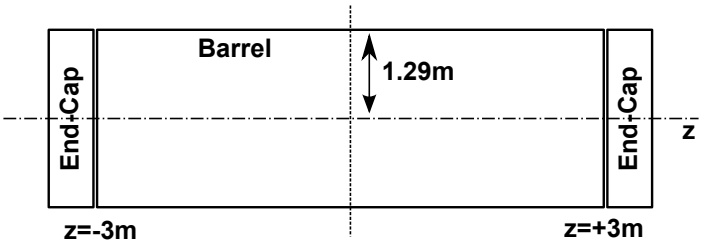


Figure 6: Barrel and End-cap part of the calorimeter in the transverse plane of the detector

Only the barrel part of the calorimeter is considered in the following.

3.4 Layout and mechanics of the barrel calorimeter

The cells are gathered in submodules; submodules are gathered in modules ; modules are gathered in supermodules. For simplifying the exercise, only the last structure is considered. There are 36 supermodules and one supermodule contains 25 × 68 cells. Their layout in the $\eta - \phi$ plane is shown by the figure below.

3.5 Data acquisition by a calorimeter cell

For the sake of completness, the acquisition chain of a calorimeter cell is briefly discussed. The scintillator crystals emit blue-green scintillation light which is collected by photodetectors (Avalanche PhotoDetectors). The signal is shaped by a MGPA (Multi-Gain Pre-Amplifier) and digitized by an ADC (Analogic Digital Converter). After an adaptation of the signal, the signal is sent to a Front-End electronics board which computes some information useful for the first level of trigger. If the trigger is fired, digital data are sent to the DAQ (Data AcQuisition).
The energy resolution can be parametrized as in the equation:

$$\left(\frac{\sigma}{E}\right)^2 = \left(\frac{S}{\sqrt{E}}\right)^2 + \left(\frac{N}{E}\right)^2 + C^2$$

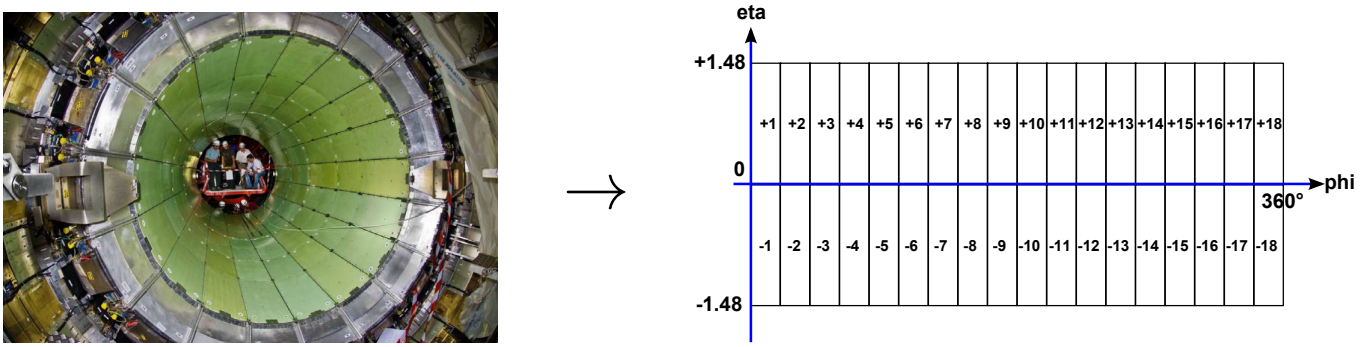


Figure 7: Subdivision of the calorimeter barrel in supermodules

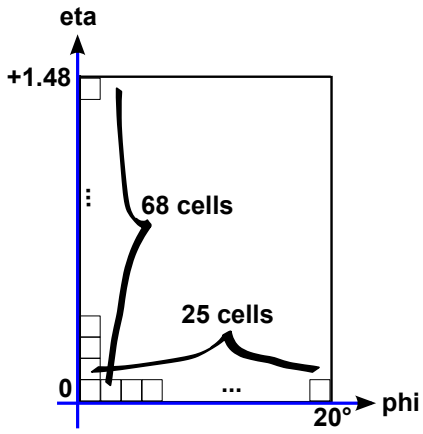


Figure 8: Subdivision of the supermodules in cells

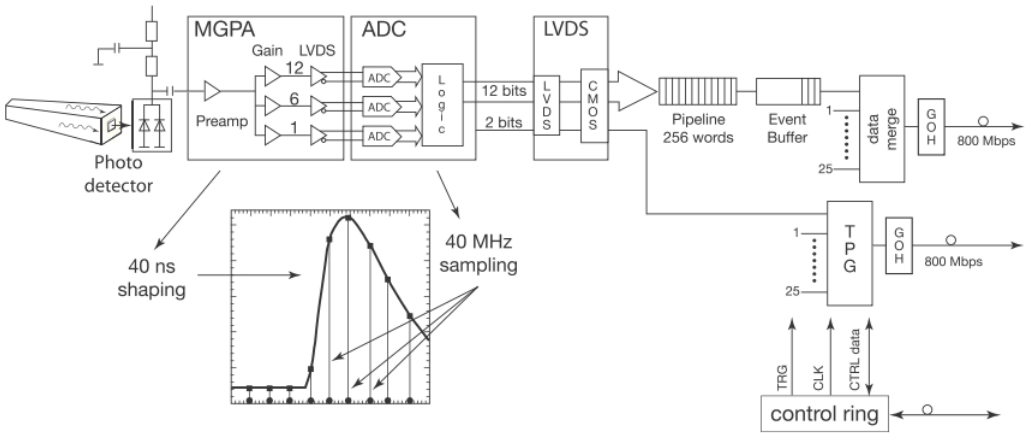


Figure 9: Simplified schematics of the calorimeter cell readout

where S is the stochastic term, N the noise term, and C the constant term. Typical values are S=2.8%, N=0.12 and C=0.30% for E in GeV.

4 Starting point

4.1 The main file `main.cpp`

The C++ project will be made up of one main source file called `main.cpp`. A skeleton of a such file could be found in Section 3 of Computing Session 1. According to the principles of modular programming, the classes that the student must develop should be stored in other source files. For compiling the project, a proposal is to use a generic `Makefile` like the one in Section 7.3 of Computing Session 2.

4.2 Programming conventions

This is a non-exhaustive list of recommendations for CMS software developpers. In the context of the exercise, the students must respect as much as possible these conventions in their source files.

- One source file and one header file per class. Naming rules: class name + suffix (.cpp or .h)
- Start method names with lower case. Use upper case initials for following words. Example: collisionPoint()
- Start data member names with lower case. User upper case initials for following words. Use "_" character at the end of the name. Example: collisionPoint_
- Do not use single character names, except for loop indices.
- Protect each header file from multiple inclusion with:

```
#ifndef className_h
#define className_h
...
#endif
```

- Header files must not contain any implementation except for class templates and code to be inlined.
- Classes must not have public data members.
- Do not use global data.
- Use "0" not "NULL".
- Use C++ casts, not C-style casting.
- Keep the ordering of methods in the header file and in the source file identical.
- Limit line length to 120 character positions.

5 Description of a calorimeter

In this section, a class called `caloCell`, corresponding to the files called `CaloClass.h` and `CaloClass.cpp`, must be written. This class must describe the status of each cell of the barrel calorimeter. Therefore 61,200 instances of this class are expected.

5.1 Specifications

Here are enumerated the functionalities of the class `caloCell`.

- The class must contain an identification code corresponding to its relative position in the supermodule. This can be done by two positive integer called `etaPosition_` and `phiPosition_`.
- The class must store the raw energy (`rawEnergy_`) coming directly from the DAQ.
- The class must also store calibration settings:
  - offset: real value to subtract to the raw energy.
  - gain: multiplicative value (defined as a strictly positive real).
  - boolean mask: if the mask is enable, a veto is applied to the cell (describing dead cell).
- A function called `getEnergy` must return the corrected energy value following the formula:

$$\begin{cases} \text{if mask=true} & \rightarrow & \text{energy*} & = & 0. \\ \text{if mask=false} & \rightarrow & \text{energy*} & = & (\text{energy - offset}) \times \text{gain} \end{cases}$$

- A function called `getResolution` must return the resolution value expected for the current corrected energy value. The formula is given in section "Physics context".
- In order to access all data members of the class, accessor (or getter) and mutator (or setter) functions must be defined. We choose the conventions that the name of these functions begin either by `get` either by `set`.
- A function called `print` allows to display at the screen the current values of the data members of the class.
- Two constructors will be implemented for this class: one constructor with no argument where data members will be initialized to the default values and a second constructor with some arguments (etaPosition, phiPosition, mask, offset, gain, energy).
- A function called clear allows to reinitialize all the data members.

The UML diagram corresponding to the class `caloCell` is supplied below.

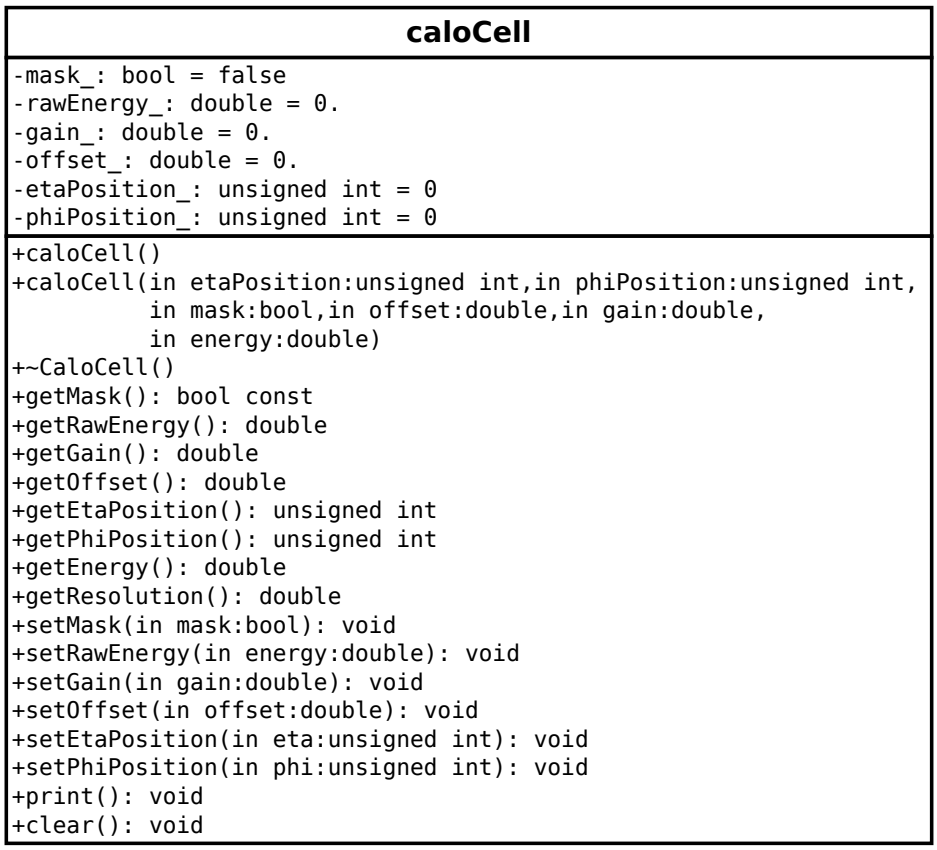


Figure 10: UML diagram of the class `caloCell`

5.2 First work to achieve

- **Implement the class `caloCell` according to the UML diagram.**
- **Test the class definition by instantiating an object and by performing some operations.**
- **Adapt the script `mymake` for building this project.**

- **Explaining how you test the implementation of `caloCell`.**

5.3 Enriching the class `CaloCell`

We suggest to improve the implementation of the class `caloCell` by advanced functionalities. These functionalities are not crucial for the next developments. Their goal is totally pedagogical.

- Add a copy constructor to the class.
- Associate the reserved word `const` to the appropriated functions.
- Overload the operator « to display the data member values when `std::cout` is applied directly to instance of this class.

- Have you other ideas (new function, optimization, ...) for improving the implementation of the class?

6 Description of a supermodule and a barrel

For modeling the electromagnetic barrel calorimeter, we would like to implement the two classes `caloSupermodule` and `caloBarrel`, corresponding to the files `caloSupermodule.h`, `caloSupermodule.cpp`, `caloBarrel.h` and `caloBarrel.cpp`. The implementation will be based on the supplied UML diagrams. We would like to have the most general and flexible classes as possible. For instance, the supermodule segmentation will be not fixed, but tunable by the user.

6.1 Implementation of `caloSupermodule` class

Here are enumerated the functionnalities of the class `caloSupermodule`.

- The class must contain a identification code corresponding to the supermodule position in the supermodule. This can be done by a signed integer called `Id_`.
- The class must store an array of `caloCell`. The data members `nPhi_` and `nEta_` mean the number of cells respectively in ϕ and η direction.
- In order to access all data members of the class, accessor (or getter) and mutator (or setter) functions must be defined. We choose the conventions that the name of these functions begin either by `get` either by `set`. Of course, changing `nPhi_` and `nEta_` implies changing the array dimension.
- A function called `print` allows to display at the screen the idenfication number and the array size.
- Two constructors will be implemented for this class: one constructor with no argument where data members will be initialized to the default values and a second constructor with some arguments (identification number, nEta, nPhi).
- A function called `clear` allows to reinitialize all the data members.
- A function called `getCell` allows to access, via a pointer, a `caloCell` located at *eta* and *phi* position.

The UML diagram corresponding to the class `caloSupermodule` is supplied below.

caloSupermodule
-id_: int = 0 -cells_: array = <empty array> -nPhi_: unsigned int = 0 -nEta_: unsigned int = 0
+caloSupermodule() +caloSupermodule(in id:int,in nEta:unsigned int, in nPhi:unsigned int) +~caloSupermodule() +getId(): int +getNEta(): unsigned int +getNPhi(): unsigned int +setId(in mask:bool): void +setId(in id:int): void +setNEta(in nEta:unsigned int): void +setNPhi(in nPhi:unsigned int): void +print(): void +clear(): void +getCell(in etaId:unsigned int,in phiId:unsigned int): caloCell*

6.2 Implementation of `caloBarrel` class

Here are enumerated the functionnalities of the class `CaloCell`.

- The class must store an array of `caloSupermodule`. The data member `nSupermodule_` mean the number of supermodules.
- In order to access all data members of the class, accessor (or getter) and mutator (or setter) functions must be defined. We choose the conventions that the name of these functions begin either by `get` either by `set`. Of course, changing `nSupermodule_` implies changing the array dimension.
- A function called `print` allows to display at the screen the idenfication number and the array size.
- Two constructors will be implemented for this class: one constructor with no argument where data members will be initialized to the default values and a second constructor with one argument (number of supermodules).
- A function called `clear` allows to reinitialize all the data members.
- A function called `getSupermodule` allows to access, via a pointer, a `caloSupermodule` with a given identification number. If no `caloSupermodule` is found, a null pointer is returned.
- A function called `getCell` allows to access, via a pointer, a `caloCell` located in a given supermodule, at relative $\eta - id$ and $\phi - id$. If no `caloCell` is found, a null pointer is returned.
- A function called `getCellDim` will give the absolute coordinate in the $\eta - \phi$ plane ($\eta_{\min}$, $\phi_{\min}$, $\eta_{\max}$ and $\phi_{\max}$) of a `caloCell` located in a given supermodule, at relative $\eta - id$ and $\phi - id$.

The UML diagram corresponding to the class `caloBarrel` is supplied below.

caloBarrel
-cells_: ARRAY<caloSupermodule> = <EMPTY> -nBarrel_: unsigned int = 0
+caloBarrel() +caloBarrel(in nSupermodule:unsigned int) +~caloBarrel() +getNSupermodules(): unsigned int +setNSupermodules(in nSupermodules:unsigned int): void +print(): void +clear(): void +getSupermodule(in supermoduleId:int): caloSupermodule* +getCell(in supermoduleId:int,in etaId:unsigned int, in phiId:unsigned int): caloCell* +getCell(in supermoduleId:int,in etaId:unsigned int, in phiId:unsigned int,out etaMin:double, out phiMin:double,out etaMax:double, out phiMax:double): void

6.3 First work to achieve

- **Implement the two classes according to the UML diagrams.**
- **Test the class definition by instantiating an object and by performing some operations.**
- **Adapt the script `mymake` for building this project.**

- **Explaining how you test theses implementations.**

6.4 Enriching the classes

Like the class `CaloCell`, we suggest to improve the implementation of the classes by advanced functionnalities. These functionnalities are not crucial for the next developments. Their goal is totally pedagogical.

- Add a copy constructor to the class.
- Associate the reserved word `const` to the appropriated functions. Advice: the methods `getCell` and `getSupermodule` will be duplicated in order to have a non-const version and a const version.
- Overload the operator $<<$ to display the data member values when `std::cout` is applied directly to instance of this class.

- Have you others ideas (new function, optimization, ...) for improving the implementation of the class?

7 Generating documentation from C++ sources

Annotation and comments inside the code is very useful for the understanding. In order to increase the documentation level, it is also possible to generate automatically reference documentation by reading the syntax and the annotations of the code. Whereas some documentation generators such as JAVADOC are specific to one programming language, the DOXYGEN program has the advantage to be used for plenty languages.

7.1 First words about the DOXYGEN package

DOXYGEN can read not only C++ language but also JAVA PYTHON, FORTRAN, PHP and others. The formats of the generated documentation are mainly HTML and LATEX (PDF or PS after LATEX compilation). It can cross reference documentation and code, so that the reader of a document can easily refer to the documentation.

The package can be downloaded from the official website (`www.doxygen.org`). From the LXPLUS session, DOXYGEN program can be launched from any folder. A small test to check the presence of this package consists in issuing the command below at the shell prompt. If the program is found, the version release must appear at the screen.

```
bash$doxygen␣--version
```

7.2 Standard doxygen configuration file

The starting point consists in writing a DOXYGEN configuration file. A template of a such file can be generated by typing the following command:

```
bash$doxygen␣-g␣doxygen.cfg
```

A text file called `doxygen.cfg` is then created and can be modified with a text editor. It contains all the available `Doxygen` options set with the default values. The syntax is very similar to a shell script. To enter into details, comment line begins with a # character and options are specified by the scheme `tag = value`. The options values are usually the reserved words `YES` or `NO` for binary options, or string for other option kinds. Appearance order of the options is not relevant.

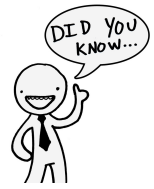
For generating HTML, the user must set the following settings:

```
1 GENERATE_HMTL␣=␣YES
2 HMTL_OUTPUT␣=␣html␣#␣name␣of␣the␣folder␣where␣HTML␣document
3 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣#␣will␣be␣generated
```

and for LATEX, the following lines

```
1 GENERATE_LATEX␣=␣YES
2 LATEX_OUTPUT␣=␣latex␣#␣name␣of␣the␣folder␣where␣LATEX␣document
3 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣#␣will␣be␣generated
```

By default, all source files (C++ and other programming languages) placed in the local folder are taken into account. These properties can be tuned by changing options such as `FILE_PATTERNS`, `RECURSIVE` and `EXCLUDE`.



A GUI (Graphical User Interface) wizard configuration tool, called `doxywizard`, exists also. It facilitates the DOXYGEN configuration and running. Nonethess this program is not installed on LXPLUS session.

7.3 Adding graphics in the reference documentation

DOXYGEN tool can use GRAPHVIZ package for generating graphs and diagrams. It can be downloaded from the official website (`http://www.graphviz.org/`). From the LXPLUS session, GRAPHVIZ is already installed and ready to used. A small test to check the presence of this package consists in issuing at the shell prompt the command below. The version of GRAPHVIZ must appear at the screen.

```
bash$dot␣-v
```

For enabling all the graphical options in the report, the user must apply the following settings:

```
HAVE_DOT␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣=␣YES
CLASS_GRAPH␣␣␣␣␣␣␣␣␣␣␣␣=␣YES
COLLABORATION_GRAPH␣␣␣␣=␣YES
GROUP_GRAPHS␣␣␣␣␣␣␣␣␣␣␣=␣YES
UML_LOOK␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣=␣NO
TEMPLATE_RELATIONS␣␣␣␣␣=␣YES
INCLUDE_GRAPH␣␣␣␣␣␣␣␣␣=␣YES
INCLUDED_BY_GRAPH␣␣␣␣␣=␣YES
CALL_GRAPH␣␣␣␣␣␣␣␣␣␣␣␣=␣YES
CALLER_GRAPH␣␣␣␣␣␣␣␣␣␣=␣YES
GRAPHICAL_HIERARCHY␣␣␣␣=␣YES
DIRECTORY_GRAPH␣␣␣␣␣␣␣␣=␣YES
DOT_MULTI_TARGETS␣␣␣␣␣␣=␣YES
```

7.4 Launching DOXYGEN

To generate automatically documentation, the user has just to type the `Doxygen` command following the name of the configuration file:

```
bash$doxygen␣doxygen.cfg
```

During the documentation generation, error or warning could be displayed. The user is invited to read these messages and to investigate the relevant ones. If the running is successful, folders `html` and `latex` are generated according to the configuration file.

- `html` folder contains all HTML files and can be browsed with a navigator internet from the file `index.html`.

- `latex` folder contains `latex` files and can be compiled with latex with a makefile. By issiung the command `make`, a PDF file is created and can be viewed with a PDF reader.

7.5 Work to do

- **Generate the documentation related to your code in LaTeX and HTML format**
- **Add/adjust annotations in your code in order to improve the generated documentation.**

Some suggestions about the documentation layout:

```
FULL_PATH_NAMES ========= NO
JAVADOC_AUTOBRIEF ======= YES
HIDE_UNDOC_CLASSES ====== NO
GENERATE_LATEX ========== NO
TAB_SIZE ================ 4
OPTIMIZE_OUTPUT_FOR_C === YES
BUILTIN_STL_SUPPORT ===== YES
EXTRACT_ALL ============= YES
RECURSIVE =============== YES
SOURCE_BROWSER ========== YES
ALPHABETICAL_INDEX ====== YES
GENERATE_TREEVIEW ======= YES
TEMPLATE_RELATIONS ====== YES
SEARCHENGINE ============ YES
REFERENCED_BY_RELATION = YES
```