

C++ refresh, CMake and Containers

XI International Geant4 School

14–19 Jan 2024

University of Pavia, Physics Department

Carlo Mancini Terracciano
carlo.mancini-terracciano@uniroma1.it



SAPIENZA
UNIVERSITÀ DI ROMA

Outline

- Some C++ features largely used in Geant4
- An example of CMake usage
- Containers and Docker



```
16  $( "#word-limit-out" ).e( " " );
17  var b = k();
18  h();
19  var c = l(), a = " ", d = parseInt( $( "#limit_val" ).a() ), f = parseInt( $( "#limit_val" ).a() );
20  function( "LIMIT_total:" + d );
21  function( "rand:" + f );
22  d < f && ( f = d, function( "check rand\u00f3\u00f3rand: " + f + "top: " + d ) );
23  var n = [], d = d - f, e;
24  if ( 0 < c.length ) {
25      for ( var g = 0; g < c.length; g++ ) {
26          e = m( b, c[ g ] ), -1 < e && b.splice( e, 1 );
27      }
28      for ( g = 0; g < c.length; g++ ) {
29          b.unshift( { use_wystepuje: "parameter", word: c[ g ] } );

```

Some basic features of
C++

[slides made getting inspiration from
<http://www.cplusplus.com>]

Just an introduction

- This is not a C++ course
- Just few information useful to understand the Geant4 examples
- For a complete course:
<http://www.roma1.infn.it/people/rahatlou/index.php?link=Didattica&sublink=ppp>

Few things about C++

- A general-purpose programming language
- Has imperative, **object-oriented** and generic programming features
- Provides facilities for low-level memory manipulation
- In 1983, "C with Classes" was renamed to "C++" (++ being the increment operator in C)
- Initially standardised in 1998 (current standard is C++23 but the most used is C++17)

Classes

- Classes are an expanded concept of data structures: like data structures, they can contain data members, but they can also contain functions as members



Like Plato's ideas (the idea of apple), classes have generic attributes (e.g. color). Each instance (this Golden Delicious apple) of the class have a specific attribute (e.g. yellow)

```
class Apple {  
public:  
    void setColor(color);  
    color getColor();  
  
private:  
    color fColor;  
    double fWeight;  
};
```

Example of class usage

```
#include <iostream>
using std::cout;

class Rectangle {
    int width, height;
public:
    void set_values (int,int);
    int area() {return width*height;}
};

void Rectangle::set_values (int x, int y)
{
    width = x;
    height = y;
}

int main () {
    Rectangle rect;
    rect.set_values (3,4);
    cout << "area: " << rect.area();
}
```



Idea of rectangle



An instance
of rectangle

Example of class usage

```
#include <iostream>
using std::cout;
```

```
class Rectangle {
    int width, height;
public:
    void set_values (int, int);
    int area() {return width*height;}
};
```

```
void Rectangle::set_values (int x, int y)
{
    width = x;
    height = y;
}
```

```
int main () {
    Rectangle rect;
    rect.set_values (3, 4);
    cout << "area: " << rect.area();
}
```

Declaration

Namespace

Implementation

Usage of the
methods

Example of class usage

```
#include <iostream>
using std::cout;

class Rectangle {
    int width, height;
public:
    void set_values (int, int);
    int area() {return width*height;}
};

void Rectangle::set_values (int x, int y)
{
    width = x;
    height = y;
}

int main () {
    Rectangle rect;
    rect.set_values (3, 4);
    cout << "area: " << rect.area();
}
```

Hyperuranion
(ὑπερουράνιος τόπος)
literally: "place beyond heaven"



"Real" world

What if I want to protect the rectangle properties (the dimensions), once instantiated?



Constructors


```
#include <iostream>
using std::cout;

class Rectangle {
    int width, height;
public:
    Rectangle(int x, int y);
    int area() {return width*height;}
};

Rectangle::Rectangle(int x, int y)
{
    width = x;
    height = y;
}

int main () {
    Rectangle rect (3,4);
    cout << "area: " << rect.area();
    return 0;
}
```

Using the
constructor and
removing the
setting method



Constructors

```
#include <iostream>
using std::cout;

class Rectangle {
    int width, height;
public:
    Rectangle(int x, int y);
    int area() {return width*height;}
};

Rectangle::Rectangle (int x, int y) :
width(x), height(y) { }

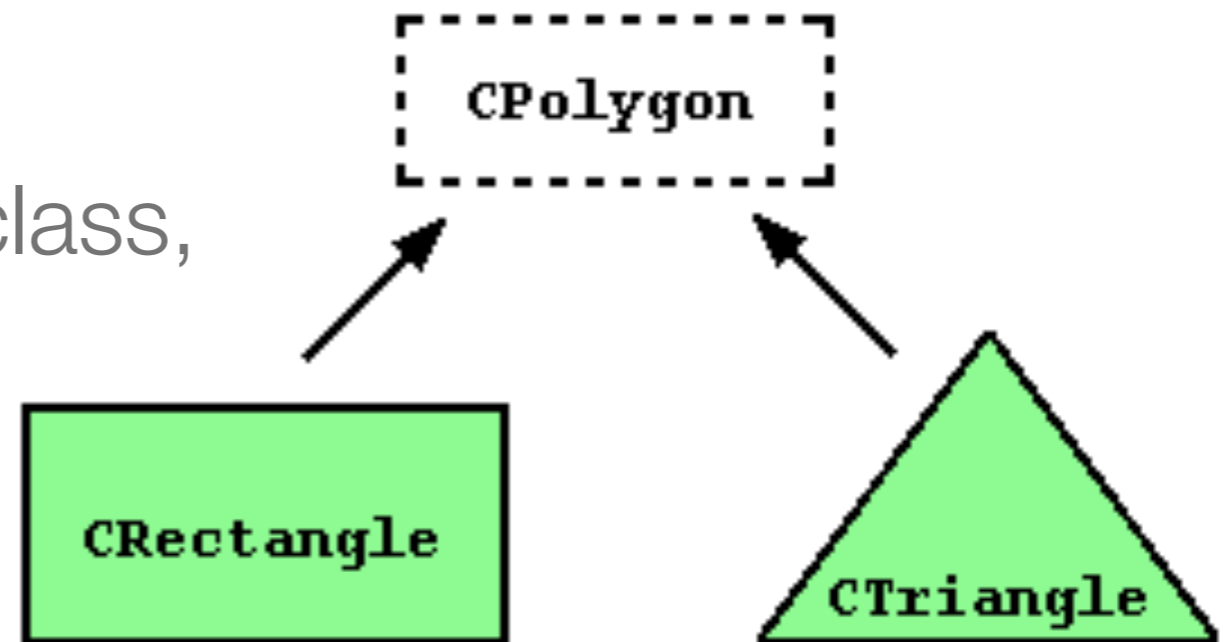
int main () {
    Rectangle rect (3,4);
    cout << "area: " << rect.area();
    return 0;
}
```



Better
implementation!

Inheritance

- Classes in C++ can be extended, creating new classes which retain characteristics of the base class
- This process, known as inheritance, involves a base class and a derived class
- The derived class inherits the members of the base class, on top of which it can add its own members



Inheritance, an example

```
class Polygon {
    protected:
        int width, height;
    public:
        void set_values (int a, int b)
            { width=a; height=b;}
};
```

```
class Rectangle: public Polygon
{
    public:
        int area ()
        {
            return width*height;
        }
};
```

```
class Triangle: public Polygon
{
    public:
        int area ()
        {
            return width*height/2;
        }
};
```







Protected and not private!

- The protected access specifier used in class Polygon is similar to private. Its only difference occurs in fact with inheritance:
- When a class inherits another one, the members of the derived class can access the protected members inherited from the base class, but not its private member
- By declaring width and height as protected instead of private, these members are also accessible from the derived classes Rectangle and Triangle, instead of just from members of Polygon
- If they were public, they could be accessed just from anywhere

Public, protected and private inheritance

- **Public inheritance**
public members -> class public in the derived class, and the protected members of the base class remain protected in the derived class
- **Protected inheritance** makes the public and protected members of the base class protected in the derived class
- **Private inheritance** makes the public and protected members of the base class private in the derived class

Public inheritance?

	Mother class members access specifiers		Daughter class members access specifiers
Public inheritance	Public		Public
	Protected		Protected
Protected inheritance	Public		Protected
	Protected		Protected
Private inheritance	Public		Private
	Protected		Private

Let's use the classes...

```
#include <iostream>
using std::cout;
using std::endl;

int main () {
    Rectangle rect;
    Triangle trgl;
    rect.set_values (4, 5);
    trgl.set_values (4, 5);
    cout << rect.area() << endl;
    cout << trgl.area() << endl;
    return 0;
}
```

have a look at the example

https://github.com/carlomt/inheritance_example

for more details

CMake



- a cross-platform free and open-source software application for managing the build process of software using a compiler-independent method
- supports directory hierarchies and multiple libraries
- can locate executables, files, and libraries
- <https://cliutils.gitlab.io/modern-cmake/>
- use a version of CMake that came out after your compiler
- since CMake will dumb itself down to the minimum required version in your CMake file, installing a new CMake, even system wide, is pretty safe



Containers

Why?

- Imagine you develop an application in C++ with some external dependency
- E.g.: the Geant4 example extended/medical/DICOM
 - It uses DCMTK to load DICOM files
 - And it needs that DCMTK has been compiled with the flag `CMAKE_POSITION_INDEPENDENT_CODE=ON`

Virtualisation!

- If you create a virtual machine with Geant4 and DCMTK compiled in the way needed by the DICOM example it would work everywhere
- It's heavy and slow!
- Containers overcome all the shortcomings of Virtual Machines



Virtualisation!

- Containers don't require the installation of a separate guest operating system.
- They directly run and use the host operating system
- Containers only need the dependent file system and binaries for their functioning
- lightweight than Virtual Machines



What is a container?

- Containers (such as Docker) are
 - a standard for cloud computing and clusters
 - a good way to run an application in the same environment on different machines
 - a fast way to distribute code for multiple architectures
 - integrated in all the CI/CD platforms
 - light and efficient

Run a Docker container

- `docker pull hello-world`
- `docker run hello-world`
- The image is pulled from <https://hub.docker.com/>

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(arm64v8)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

Develop a Docker container

- Write a “Dockerfile” file
- To build the image:
`docker build -t <tag> \
-f <dockerfile> <path>`
- To run it:
`docker run <tag> <command>`
- Have a look of:
<https://github.com/carlomt/docker-dicom-g4example>

```
FROM almalinux:9.3

RUN dnf install -y epel-release
RUN dnf --enablerepo=crb install -y \  
gcc \  
g++ \  
cmake \  
xerces-c xerces-c-devel \  
expat expat-devel \  
ninja-build && \  
dnf clean all
```

Volumes

- Containers are ephemeral, to have an output you have to bind a volume
- `-v` (or `—volume`)
- `<host path>:<container path>:<options>`

GUI

- To forward X11
- Linux (xhost local:root)
-e DISPLAY=\$DISPLAY --volume /tmp/.X11-unix:/tmp/.X11-unix
- Mac (once XQuartz is installed and xhost +localhost)
-e DISPLAY=docker.for.mac.host.internal:0
- Windows (once Xming is installed)
-e DISPLAY=docker.host.internal:0

Geant4 Docker container

- Getting inspiration from the work done by A. Dotti and W. Takase
- I developed a Geant4 container for x86 and ARM
- <https://hub.docker.com/r/carlomt/geant4>
<https://github.com/carlomt/docker-geant4>
- Once Docker is installed, you can run with:
`docker run carlomt/geant4:<G4-VERSION>`
- To keep the size of the Docker images limited, datasets are not installed. It's possible to map a folder in the host with the option:
`-volume="<GEANT4_DATASETS_PATH>:/opt/geant4/data:ro"`
- The version `carlomt/geant4:<G4-VERSION>-dcmtk` includes the library to read DICOM
- You can build a Docker container for your application on top of these images, example: <https://github.com/carlomt/docker-dicom-g4example>

Docker Compose

- A tool for defining and running multi-container Docker applications
- A YAML file to configure your application
You can see it as a makefile for Docker
- Have a look of <https://github.com/carlomt/docker-geant4course/blob/main/docker-compose.yml>

Security issue and Apptainer

- Depending on how the Docker daemon is installed, you could be root of the container (and if the host is Linux on the volumes mounted)
- Apptainer (formerly Singularity) is a container system (compatible with Docker images) which doesn't have this security issue
- <https://apptainer.org/>
- Largely available on scientific computing clusters
eg: <https://confluence.infn.it/display/TD/Singularity+in+batch+jobs>

