Geant4 & NEXUS Hands-on Tutorial

Justo Martín-Albo (IFIC)

IFIC • 24–25 February 2020

GOAL AND MEANS

Our main goal for this tutorial:

- Become familiar enough with NEXUS to be able to start contributing to its development.

How?

- Review the C++ concepts most frequently used in Geant4 and NEXUS.
- Introduce the basic components that form a Geant4 simulation.
- Learn how to build and run NEXUS, and review its structure and organization.
- Show what documentation resources can be useful when writing NEXUS code.
- Write a full simulation within NEXUS and analyze the generated data.

NEXUS (*NEXt Utility for Simulation*) is the Geant4-based detector simulation of the NEXT experiment. It handles the different *detector geometries* (DEMO, NEW, NEXT-100…) and *event generators* (DBDs, radioactive backgrounds, muons…) we need, and produces output files (using now HDF5) in a common format understood by the software downstream.

Geant4 is a C++ library for the simulation of the passage of particles through matter using Monte-Carlo methods.

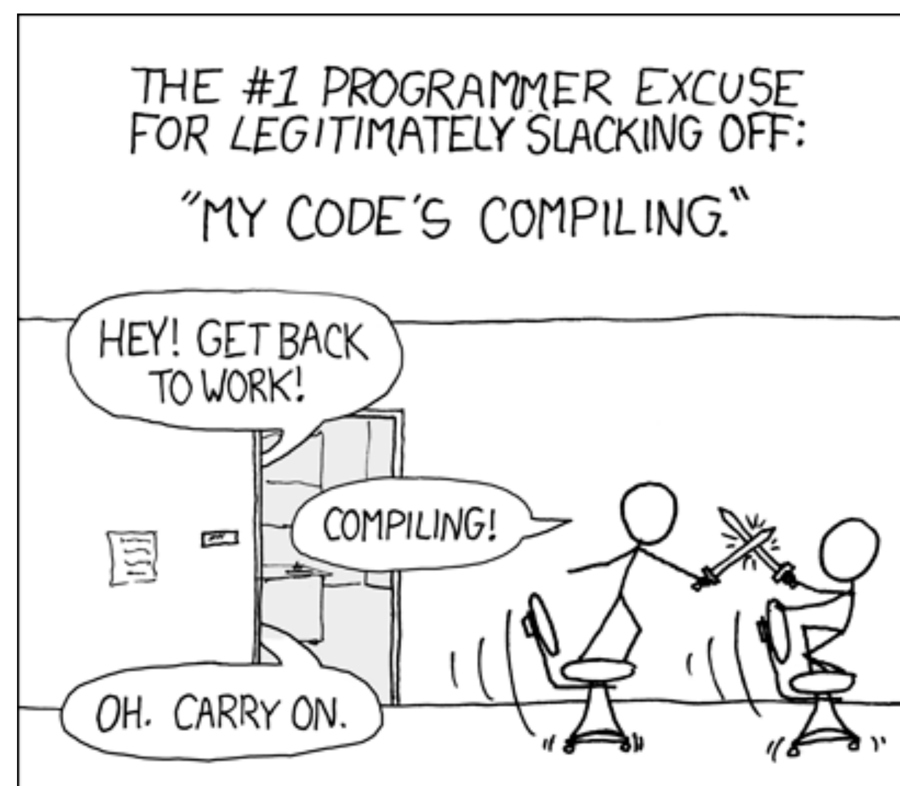
A very brief introduction to C++

Fork and clone the following repository:

https://github.com/jmalbos/nexus-tutorial

Build the examples of the C++ introduction:

```
cd nexus-tutorial/C++Intro
mkdir build && cd $_;
cmake ..
cmake —build .
```



C++ INTRO

0. Function declaration and definition. Main function and control flow. Scope of variables and namespaces.

1. Passing arguments by value, reference and pointer.

2. Declaration and definition of classes. Constructors and destructors.

3. Class inheritance. Memory allocation.

4. Polymorphism. Vectors.

C++ INTRO: SCOPE


```
#include <iostream>

int var = 123; // Global variable with file scope

int main()
{
 int var = 456; // Local variable with scope limited to main()
 std::cout << "var = " << var << std::endl;
 return 0;
}
```

Output:

```
var = 456
```

C++ INTRO: PASSING ARGUMENTS


```
#include <iostream>

int add_one_by_val(int  n); // Passing argument by value
int add_one_by_ref(int& n); // Passing argument by reference

int main()
{
 int var = 0;
 std::cout << add_one_by_val(var) << "!=" << var << std::endl;
 std::cout << add_one_by_ref(var) << "==" << var << std::endl;
 return 0;
}
```

Output

```
0 != 1
1 == 1
```

C++ INTRO: CLASSES


```
#include <iostream>

class MyFirstClass
{
public:
 MyFirstClass() {}
 ~MyFirstClass() {}
 void Greeting() { std::cout << greeting_ << std::endl; }
private:
 std::string greeting_ = “Hello World!”;
};

int main()
{
 MyFirstClass mfc;
 mfc.Greeting();
 return 0;
}
```

Output:

```
Hello World!
```

C++ INTRO: INHERITANCE


```
#include <iostream>

class BaseClass
{
public:
 virtual BaseClassFunction()
 { std::cout << "BaseClassFunction()" << std::endl; }
};

class DerivedClass: public BaseClass
{
public:
 virtual DerivedClassFunction()
 { std::cout << "DerivedClassFunction()" << std::endl; }
};

int main()
{
 DerivedClass dc_instance;
 dc_instance.BaseClassFunction();
 dc_instance.DerivedClassFunction();
 return 0;
}
```

Output:

```
BaseClassFunction()
DerivedClassFunction()
```

C++ INTRO: INHERITANCE

SPECIFIER	WITHIN SAME CLASS	WITHIN DERIVED CLASS	FROM OUTSIDE
`public`	✔	✔	✔
`protected`	✔	✔	✗
`private`	✔	✗	✗

C++ INTRO: POLYMORPHISM


```
#include <iostream>

class Square: public Shape
{ (…) };

class Circle: public Shape
{ (…) };

int main()
{
 Shape* square = new Square(1.0);
 Shape* circle = new Circle(1.0);
 std::vector<Shape*> vs = {circle, square};
 for (auto i: vs) std::cout << “Area: ” << i->Area() << std::endl;
 return 0;
}
```

Output:

```
Area: 3.14159
Area: 1.0
```

Geant4 Basics

A GEANT4 RUN

BASIC COMPONENTS

To build a Geant4 application, one must provide at least the following components:

- **Detector geometry** (`G4VUserDetectorConstruction`): description of the detector setup in terms of shapes, their relative positions and the materials they are made of.
- **Primary generation** (`G4VUserPrimaryGeneratorAction`): initial conditions of the events; what primary particles should be produced, where and with what kinematics.
- **Physics list** (`G4VUserPhysicsList`): particle types and physical processes considered for the simulation.

Pointers to these objects are passed to the `G4RunManager` — the class in charge of run control — in the main function of the simulation application.

BASIC COMPONENTS

In addition to the above mandatory classes, several action classes can be provided to extract information from the simulation at several points in the processing:

- **Run action** (`G4UserRunAction`): actions before and after each run.
- **Event action** (`G4UserEventAction`): actions before and after each event.
- **Tracking action** (`G4UserTrackingAction`): actions before and after the processing of each track.
- **Stepping action** (`G4UserSteppingAction`): actions after each tracking step.