Héritage en C++: les animaux

classe mère: Pangolin.hpp

```cpp
#include <string>

class Pangolin {
  int nb_ecailles_;

public:
  const std::string nom_;

  Pangolin(std::string nom, int nb_ecailles) : nom_(nom), nb_ecailles_(nb_ecailles) {}
  ~Pangolin() = default;

  int get_nb_ecailles() const { return nb_ecailles_; }
  void set_nb_ecailles(int nb) { this->nb_ecailles_ = nb; }

  const std::string cri() const { return "Gwwark Rhââgn Bwwikk"; /* Cri du pangolin */  }
};
```

classe fille: PangolinALongueQueue.hpp

```cpp
#include "Pangolin.hpp"

class PangolinALongueQueue : public Pangolin {
  double lgr_queue_;

public:
  PangolinALongueQueue(std::string nom, int nb_ecailles, double lgr)
     : Pangolin(nom, nb_ecailles), lgr_queue_(lgr) {}
  double get_longueur() const { return lgr_queue_; }
};
```

usage

```cpp
#include "PangolinALongueQueue.hpp"
#include <iostream>

int main() {
    Pangolin p("toto", 2241);
    PangolinALongueQueue q("tutu", 3321, 42);
    std::cout << p.nom_ << "={ecailles = "  << p.get_nb_ecailles() << "; cri= " << p.cri() << "}\n";
    std::cout << q.nom_ << "={ecailles = "  << q.get_nb_ecailles() << "; cri= " << q.cri() << ";"
              << "lgr:" << q.get_longueur() << "}\n";
    return 0;
}
```

affichage lors de l'usage

```
$ clang++ MainPangolin.cpp -o pangolin && ./pangolin
toto {ecailles = 2241; cri= Gwwark Rhââgn Bwwikk}
tutu {ecailles = 3321; cri= Gwwark Rhââgn Bwwikk; lgr:42}
```

Héritage multiple en diamant

```cpp
class Base {};
class Deriv1 : public virtual Base {};
class Deriv2 : public virtual Base {};
class Fille : public Deriv1, public Deriv2 {};
```

Opérateurs de conversion

```cpp
struct A {
  A(int) { }      // converting ctor
  A(int, int) { } // converting ctor
  operator bool() const { return true; }
};
```

```cpp
A a1 = 1;      // copy-init
A a2(2);       // direct-init
A a3 {4, 5};   // direct-list-init
A a4 = {4, 5}; // copy-list-init
A a5 = (A)1;   // explicit cast
if (a1) ;      //
bool na1 = a1; // copy-init
bool na2 = static_cast<bool>(a1);
```

Redéfinition de fonctions

Marsouin.hpp

```hpp
#include <string>

class Marsouin {
    public:
    virtual const std::string cri();
};
class MarsouinFou : public Marsouin {
    public:
    const std::string cri() override;
};
```

Marsouin.cpp

```cpp
#include "Marsouin.hpp"

const std::string Marsouin::cri() {
    return "Gahbahiii";
}

const std::string MarsouinFou::cri()  {
    return std::string("Kweghou ") + Marsouin::cri();
}
```

Implémentation du C++

Héritage trivial en mémoire

```cpp
class Mere {
   int u;
   int v;
};
class Fille: public Mere {
   int x;
   int y;
};

int main() {
   Mere a;
   Fille b;
   Mere a2 = Fille();
   //Fille b2 = static_cast<Fille>(a2);

   printf("sizeof: a=%d; b=%d; a2=%d\n",
          sizeof(a), sizeof(b), sizeof(a2));

   return 0;
}

```

Héritage polymorphique en mémoire

```cpp
class Mere {
   int u;
   int v;
 public:
   virtual void toto() { std::cout <<"Mere\n";}
};
class Fille: public Mere {
   int x;
   int y;
 public:
   void toto() override { std::cout<<"Fille\n";}
};

int main() {
   Mere a;
   Fille b;
   Mere a2 = Fille();
   Mere* a3 = new Fille();

   a.toto();
   b.toto();
   a2.toto();
   a3->toto();

   std::cout<< "sizeof(a)=" << sizeof(a) <<"\n";
   std::cout<< "sizeof(b)=" << sizeof(b) <<"\n";
   std::cout<< "sizeof(a2)="<< sizeof(a2)<<"\n";
   std::cout<< "sizeof(a3)="<< sizeof(a3)<<"\n";

   return 0;
}

```