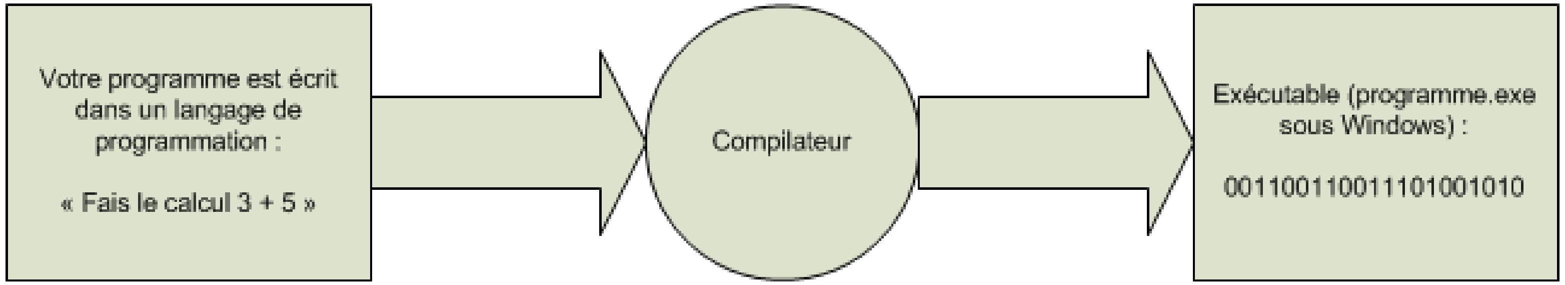
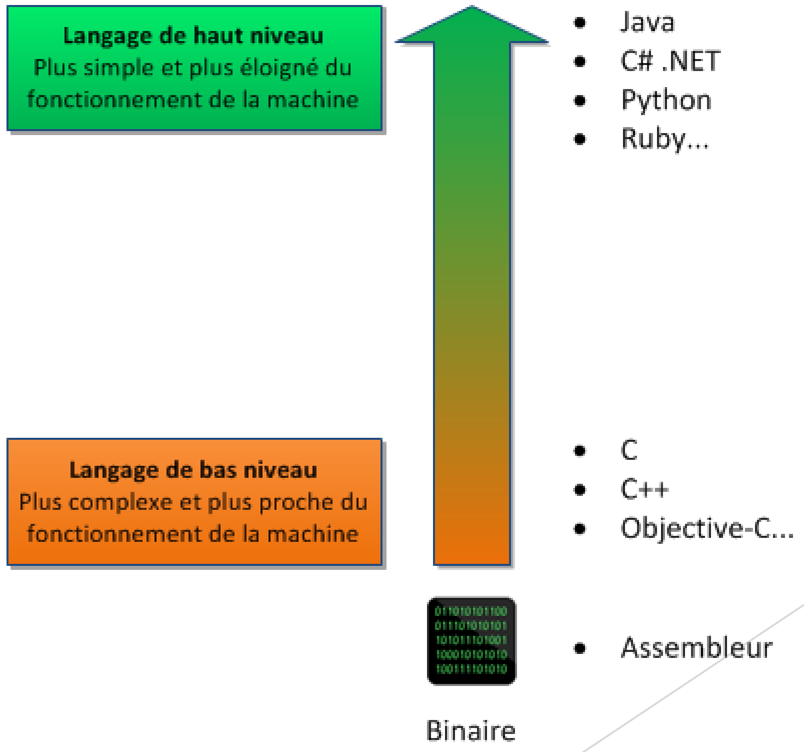
C++ Programming Language

By: Mohamed Aziz Tousli

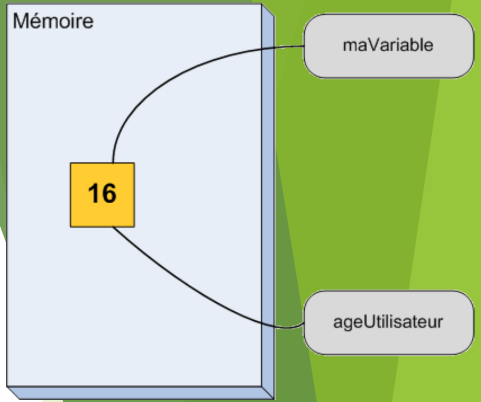
About


Votre programme est écrit dans un langage de programmation :
« Fais le calcul 3 + 5 »
Compilateur
Exécutable (programme.exe sous Windows) :
0011001100111101001010

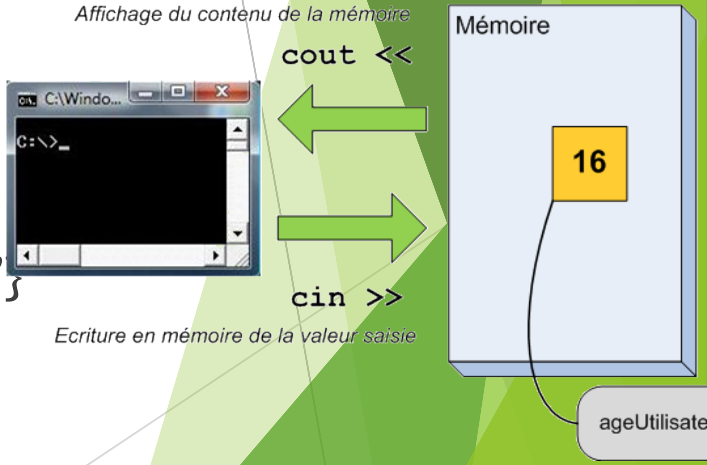
- C++ advantages:
  - Widespread
  - Fast
  - Portable (Many OS's)
  - Many libraries
  - Multi-paradigm (Many ways to program


Langage de haut niveau
Plus simple et plus éloigné du fonctionnement de la machine
Langage de bas niveau
Plus complexe et plus proche du fonctionnement de la machine
Binaire
Java
C# .NET
Python
Ruby...
C
C++
Objective-C...
Assembleur

Basics (1)



- //This is a short comment
- /* This is a long comment */
- #include <iostream> //Include "Input and Output Stream" library (preprocessor directive)
- #include <bits/stdc++.h> //Include all libraries
- using namespace std; //Important to avoid "std::instructionHere" for <u>standard </u>libraries
- cout << "Insert message here" << variable << endl << "Insert message in another line here"; //Write
- int main() { //Main function → Necessary in every C++ program
  - /* Instructions here */
  - return 0; }


Affichage du contenu de la mémoire
cout <<
C:\Windo...
C:\>_
cin >>
Ecriture en mémoire de la valeur saisie
Mémoire
16
ageUtilisate

- <u>Variable types</u>: bool{true/false}, char{'x'}, int, unsigned int, double, string["Hi"]}
  - type name (value); type name = value; //Declare a variable
  - type name1(value), name2(value) //Declare multiple variables
    - PS: It Is highly recommended to do initializations of new variables
- variableType const variableName(value); //Declare a constant
- <u>Reference </u>= <u>Bias</u>: type name(value); sameType& reference(name); //Create a reference to variable

Basics (2)

- cin >> variable; //Read content and put it in variable
- getline(cin,stringVariable); //Read string and put it in stringVariable
  - PS: cin >> can't deal with 'spaces' for strings, we use getline() instead
  - If we want to use cin >> and getline(), we must add cin.ignore() after every cin >>
- Algebraic operations: +, -, *, /, %
- Incrementation: i=i+1 ⇔ i++ ⇔ ++i ⇔ i+=1
- #include <string> //Call string library
- #include <cmath> //Math library
  - sqrt(), fabs(), floor(), ceil(), pow(x,n)
- Boolean operations: ==, >, >=, <, <=, !=
- Condition operations: &&, ||, !
- PS: if (value) ⇔ if (value==true) ⇔ if (value==1)

```
//Generate a random number
#include<ctime>
#include<cstdlib>
srand(time(0)); //ONCE
randomNumber = rand() % N;
```

Control Structures

```
if (/*condition 1*/)
        {/*code*/}
else if (/*condition 2*/)
        {/* code */}
else
        {/* code */}
```

```
switch (intVariable)
{case value1:
        /*code*/
        break;
default:
        /* code */}
```

```
while (/*condition*/)
        {/* code */}
```

```
do {/* code */}
while (/*condition*/)
```

```
for (initialization; condition ; incrementation)
        {/* code */}
```

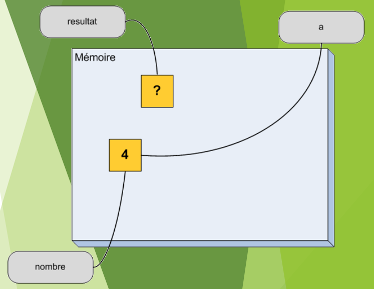
Functions

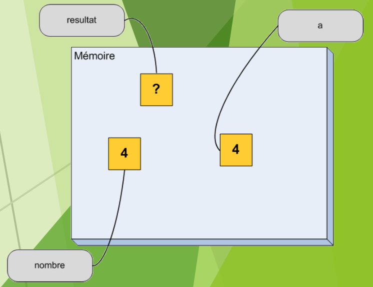
- typeReturn functionName (argument1Type argument1Name, arg2Type arg2Name) //Create a function
- { /* code */
- return something}
  - Overload: We can have two functions with the same name, if they don't have the same arguments
- PS: typeReturn = void, in a function that has no return
- **Passage by value**: typeReturn functionName (argType argName)

  → Copy argumentName !
- **Passage by reference**: typeReturn functionName (argType& argName)

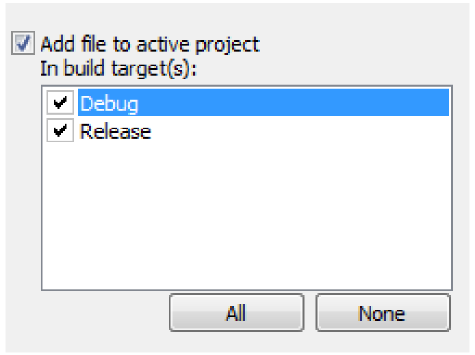
  → Modify argumentName !
- **Passage by constant reference**: typeReturn functionName (argType const& argName)

  → Doesn't copy & doesn't modify argumentName 😊
- type Fname(argument1, argument2=default); //Default value for argument2 in function prototype
  - Fname(argument1); or Fname(argument1, argument2); //Call Fname in main
  - Only function prototype contains default values
  - Default values must be in the end of list of arguments, i.e., to the right





Work Organization


Add file to active project [✓]

In build target(s):

- [✓] Debug
- [✓] Release

Fname.c

```
#include “Fname.h”

type Fname(arguments)
{
/* code */
}
```

Fname.h

```
#ifndef LIBRARY_NAME_H_INCLUDED
#define LIBRARY_NAME_H_INCLUDED

/* Comment about file */
type Fname(arguments); //Function prototype

#endif
```

main.c

```
#include “Fname.h”
using namespace std

int main()
{ Fname();
Return 0}
```

It is not recommended to use **using namespace std** in .h file

→ For complicated types (i.e. string / array / vector), we use **std::string** in argument instead of **string**

Arrays

- type array[constSizeArray]; //Declare a **static array**
- array[i] = value; //Insert value at the ith position
- Arrays vs functions:
  - Functions can't return arrays
  - Functions modify arrays by reference without the '&', type funcName(type array[], int sizeArray)
- array[i] = value; //Insert value at the ith position
- #include <vector> //Import vector library (vector = **dynamic array**)
- vector<type> array; vector<type> array(sizeArray); //Declare a vector
- vector<type> array(sizeArray, value0); //Declare a vector ; array=[value0, value0, …, value0]
- array.push_back(newValue); //Add newValue to array
- array.pop_back(); //Delete last value from array
- array.size(); //Give size of array
- Vectors vs functions: Vector is used in same way of normal variables
- type staticMatrix[sizeX][sizeY]; //Declare a **dynamic matrix** / **multidimensional static array**
- vector<vector<type> > dynamicMatrix; //Declare a **dynamic matrix** (not recommended)
  - dynamicMatrix.push_back(vector<int>); //Add line to dynamicMatrix
  - dynamicMatrix[y].push_back(); //Add element x to line y in dynamicMatrix

Files

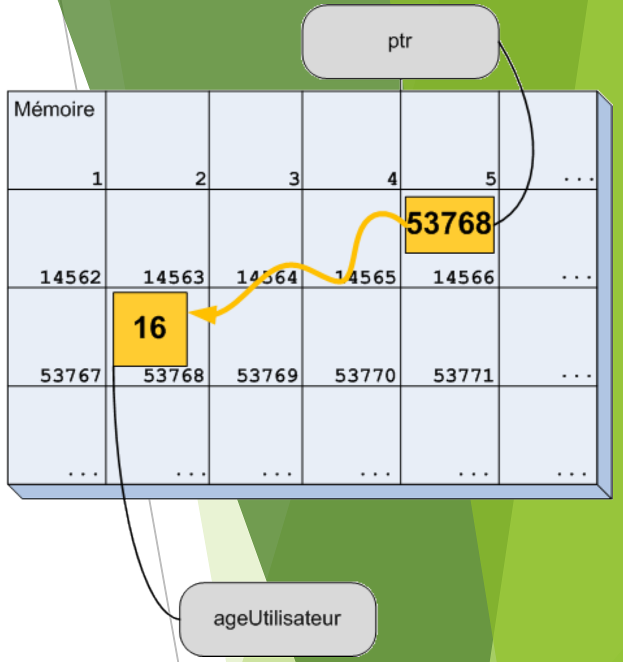
Cursor functions	ofstream	ifstream
Get position	myFile.tellp()	tellg
Move to position	myFile.seekp(numberOfChar, position);	seekg

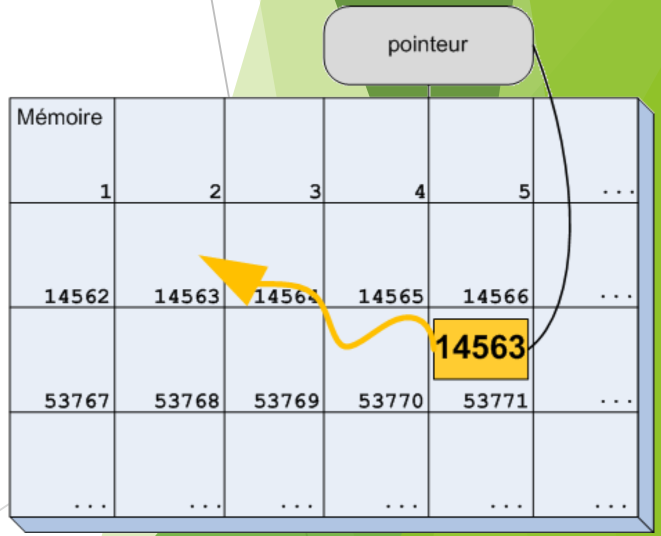
position =
- Beginning: ios::beg
- End: ios::end
- Cursor: ios::cur

- Stream to files ⬄ Read and modify files
- #include <fstream> //File library
- ofstream myFile("C:/fileDirectory.txt", ios::app); //Open file to append
- ofstream myFile("C:/fileDirectory.txt"); //Open file to write (absolute path/relative path)
  - If file doesn't exist, it will be created. If folder doesn't exist, it will give an error
  - If (!myFile) → error
- myFile << messageToWriteInFile; //Write in file
- ifstream myFile("C:/fileDirectory.txt"); //Open file to read
- string line; getline(myFile, line); //Read file line by line, return **false** if the end is reached
- int variable; myFile >> variable; //Read file word by word (space ' ' is the seperator)
- char a; myFile,get(a); //Read file character by characte (all type of characters: '\n', ' '…)
  - PS: If we use >> method, and we want to change the reading mode, we use: myFile.ignore(); between them
- myFile.close(); //Close file; It automatically closes in C++
- myFile.open(); //Open file after declaration of myFile as ofstream or ifstream

Pointers

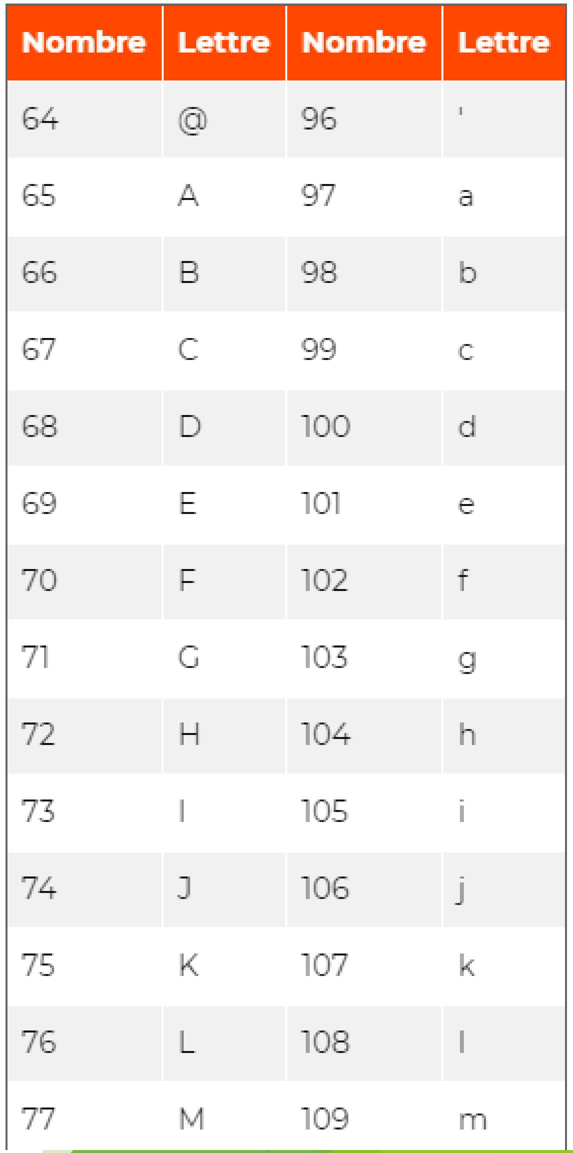
- A pointer is a variable that contains the address of another variable
- variableType *pointerOnType(0); //Create a pointer
  - Address 0 doesn't exist
- &variable; //Get address of variable
- *pointer; //Get value of addressed variable
- pointer = new variableType; //Allocate a memory cell
  - Memory leak: When you lose the value of a pointer
  - ***Important***: Every 'new' needs a 'delete'!
- delete pointer; pointer = 0; //Free memory cell
  - PS: When pointer is deleted, it still points on the address, so we have to do =0
- When to use pointers?
  - Manage the creation and the destruction of the memory cells
  - Share a variable in several pieces of code
  - Select one of several options



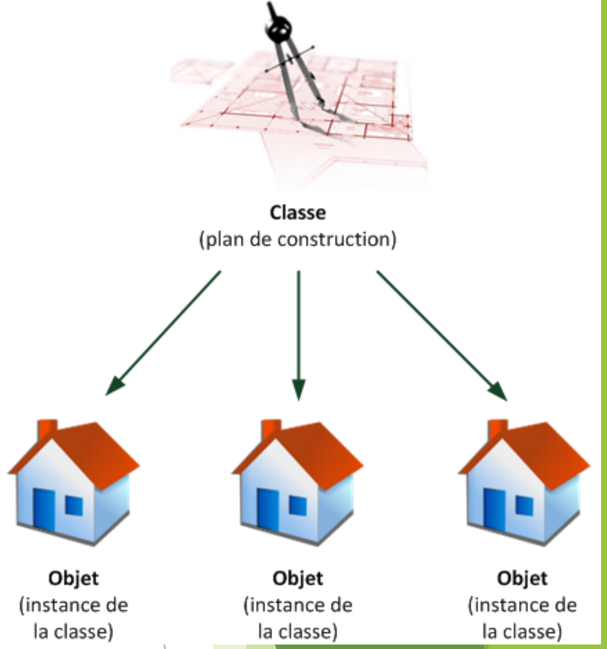


String Object

- string text(“”) object ⬄ char text[100]; ⬄ vector<char> text;
- #include <string> //Call string library
- string str; //Create string object
  - PS: It should have been “String” instead of “string”
- str[i]=char; //Replace ith letter with char
- str.size(); //Return size of str
- str1+=str2; //Concatenation
- str.erase(); ⬄ str=“” //Erase whole str
- str.erase(pos, nbOfChar); //Erase chars
- pointer = str.c_str(); //Get pointer on table of char of str
- str.substr(pos, nbOfChar); //Substring str



Nombre	Lettre	Nombre	Lettre
64	@	96	'
65	A	97	a
66	B	98	b
67	C	99	c
68	D	100	d
69	E	101	e
70	F	102	f
71	G	103	g
72	H	104	h
73	I	105	i
74	J	106	j
75	K	107	k
76	L	108	l
77	M	109	m



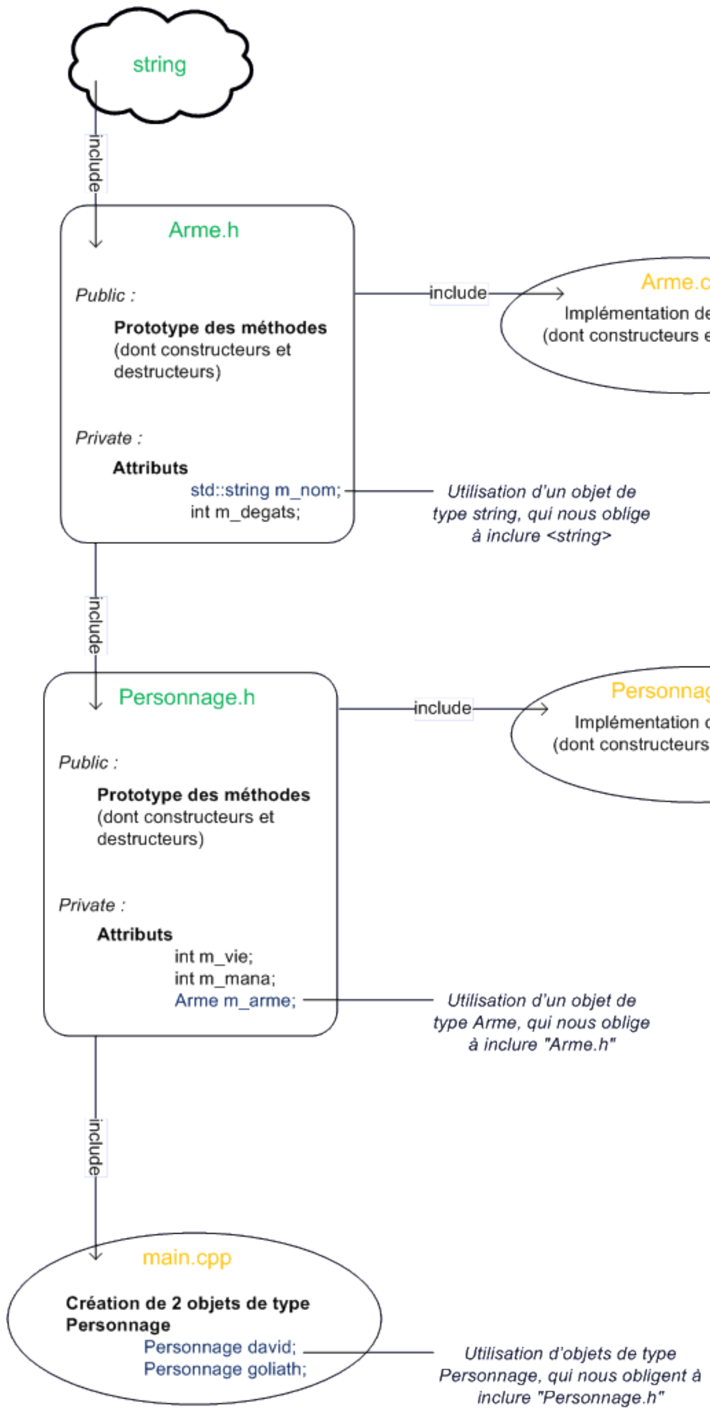
OOP (1)

- Inside an object:
  - Variables = **Attributes** = Member variables
  - Functions = **Methods** = Member functions
- object.method(); //Call a method of an object
- Object is instance of a class
- class ClassName {
- Public: returnType methodName(args) { /* code here */ } //Methods
- Private: attributeType m_attributeName; //Attributes; //m as member (organizational purposes)
- }; //Create a class
  - We can write 'struct' instead of 'class'. 'struct' → Public by default; 'class' → Private by default
  - Public and private are called 'scope' or 'access permission'
- **Encapsulation : All the attributes of a class must be always private!**

  → Users shouldn't modify the values of the attributes
- Organization:
  - header (*.h) for attributes and prototypes of methods of class
  - code source (*.cpp) for methods
    - returnType className::methodName (args) { /* code here */ } //Define a method outside of class
- PS: Methods can use attributes without having them in the arguments

OOP (2)

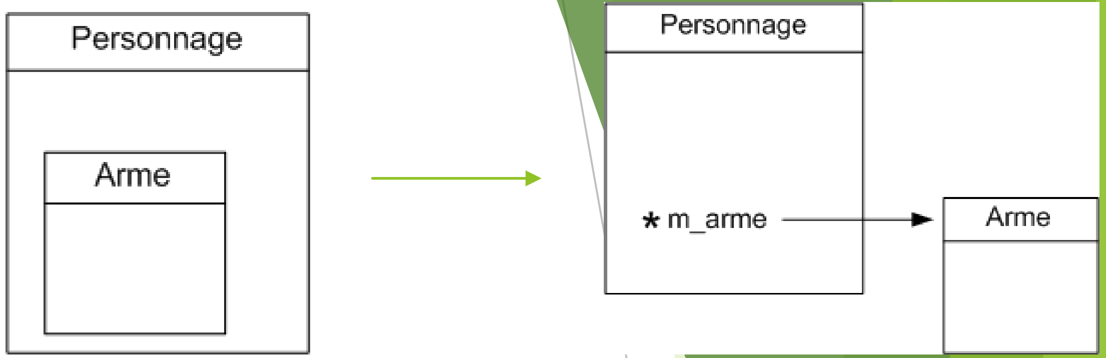
- Classical types get a random value when created
- Object types (for exp string) get a fixed value by C++ when created
- className() { m_attributeName = initialization; } //Constructor by default
- className() : m_attributeName(initialization), … { /* no code here */ } //Initialization list
  - PS: Prototype of function doesn’t change
- className(argType argName) : m_attName(argName), … { /* no code here */ } //Overload constructor
  - className objectName(argName); //Creation of an object with an overload constructor
- className(className const& object2) : m_attName(object2.m_attName);
  - className object1(object2); //Copy constructor (by default it copies the contents of object2)
- ~className(); //Create a destructor (used when ‘new’ is called to deallocate from memory)
- returnType methodName(args) const { /* code here */ } //Constant method (read only, no modifications)
- Since we can’t get attributes by doing className.attributeName, we will need to:
  - typeAttribute getAttribute () const { return m_AttributeName; } //Accessor to get attributes
  - Void setAttribute () const { /* code here */ } //Accessor to set attributes


string
include
Arme.h
Public :
Prototype des méthodes
(dont constructeurs et destructeurs)
Private :
Attributs
std::string m_nom;
int m_degats;
include
Arme.cpp
Implémentation des méthodes
(dont constructeurs et destructeurs)
Utilisation d'un objet de type string, qui nous oblige à inclure <string>
include
Personnage.h
Public :
Prototype des méthodes
(dont constructeurs et destructeurs)
Private :
Attributs
int m_vie;
int m_mana;
Arme m_arme;
include
Personnage.cpp
Implémentation des méthodes
(dont constructeurs et destructeurs)
Utilisation d'un objet de type Arme, qui nous oblige à inclure "Arme.h"
include
main.cpp
Création de 2 objets de type Personnage
Personnage david;
Personnage goliath;
Utilisation d'objets de type Personnage, qui nous obligent à inclure "Personnage.h"

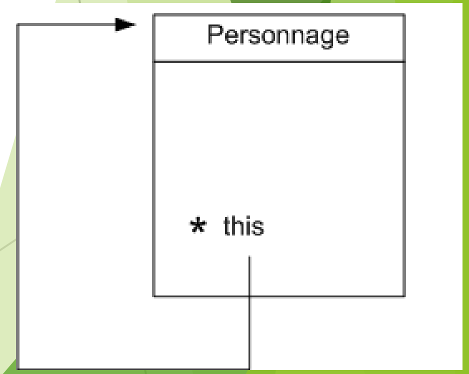
Operator Overload

- Comparison operators:
  - bool operator==(className const &a, className const& b) ⇔ a == b
  - → This is not a method, this is a function located outside of the class
  - → We create isEqual() method inside class, and operator==() function outside class that calls it
  - ❑ Other operators: !=, <, <=, >, >=
- Arithmetic operators:
  - className operator+(className const &a, className const& b) ⇔ c = a + b
  - ❑ Other operators: *, -, /, %
  - ❑ Shortcut operators: +=, *=, -=, /=, %=
    - They must be written inside the class since they change the value of the attribute
    - They must return a reference on *this, and their prototype should start with className&
- Flow operators:
  - ostream& operator<<(ostream &flow, className const& classObject) { /* code here */ }
    - → cout << classObject; //Code in main
  - PS: When we include <iostream>, **cout** object is created from class **ostream**

Pointers & Classes (1)



- Pointers are used in OOP to associate classes between each other
- Instead of creating classB inside classA, we'll create a pointer to classB inside classA
  - class classA{ /* code here */ classB *objectB(0); }
  - objectB = new classB(); //Dynamic allocation in constructor of classA //classB() calls the constructor of classB
  - ~classA() {delete objectB; } //Desallocation in destructor of classA //Avoid memory leak problems
  - PS: Do not forget that objectB.method(); ➔ objectB->method(); (object B is a pointer now)
- 'this' is a pointer on the object itself
  - *this is the object itself
- classA(classA const& objectToCopy): attA(objectToCopy.attA) //Copy constructor
- {attB = new classA(*objectToCopy.attB;}
  - Problem with copy constructor is that objectA and objectB will point on the same thing

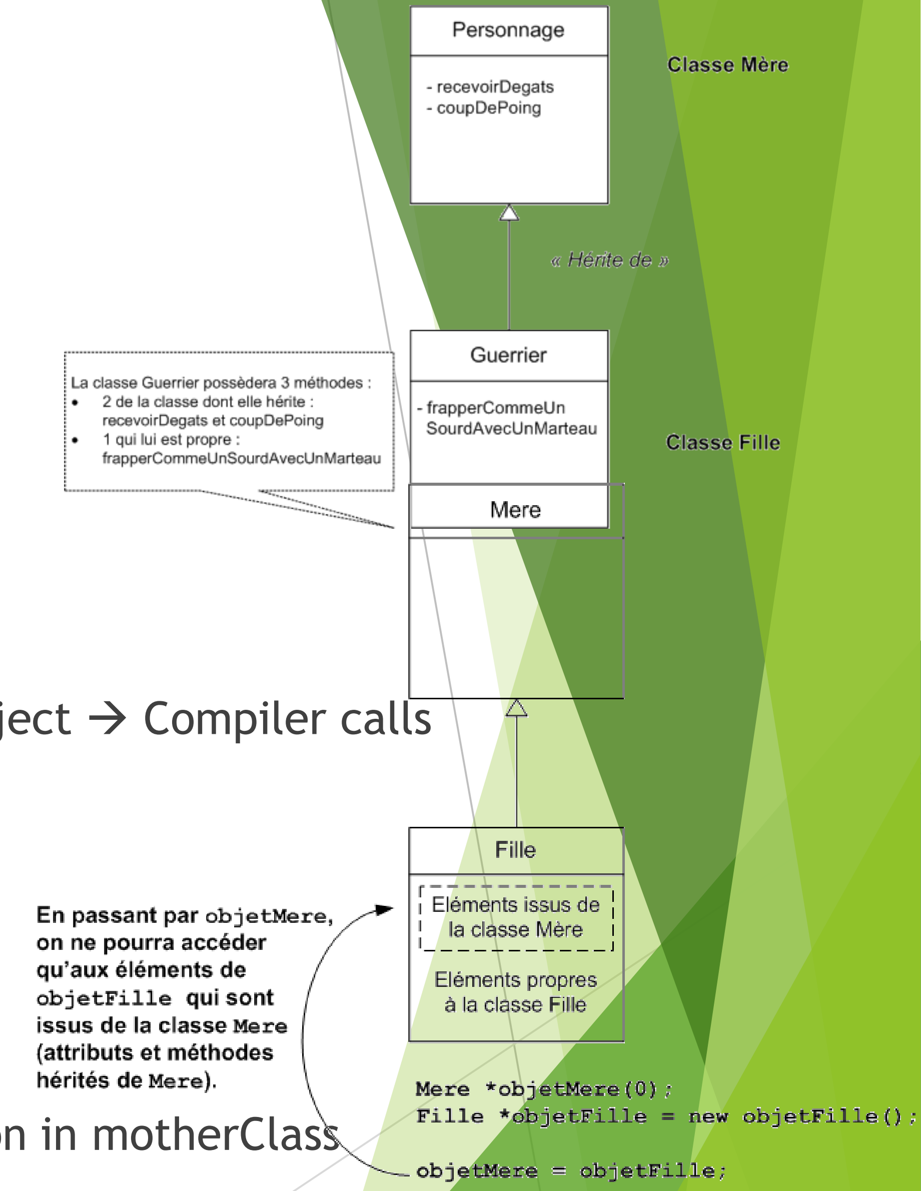


Pointers & Classes (2)

- classA& operator={classA const& objectToCopy}
- { if (this!=&objectToCopy) //Verify object=object
- { attA = objectToCopy.attA;
- delete attB; //If this value existed before
- attB = classB(*(objectToCopy.attB)); }
- return *this; //Return object itself
- PS: className obj1 = obj2; //Copy constructor
- obj1 = obj2; //operator=
  - ➔ It is preferable to write the copy constructor and the operator= together

Heritage

- Heritage is possible when we can say “daughterclass **IS** motherClass”
- #include “motherClass.h”
- class daughterClass : public motherClass{};
  - daughterClass has the same attributes and methods of motherClass
  - daughterClass is specialization of motherClass
- Create daughterObject → Compiler calls standard constructor of motherObject → Compiler calls standard constructor of daughterObject
  - PS: We can have the same with no standard constructor:
  - daughterClass() : motherClass() …
- PS: **Important**: We can only do motherObject = daughterObject;
- protected : attributes //It can only be accessible by daughter classes
- **Masking**: Write a function in daughterClass with the same name of a function in motherClass
- **Unmasking**: daugherClass::functionName() { motherClass::functionName(); /* code here */ }
  - Unmasking must be done in .h file!
  - ‘::’ is called **Scope Resolution Operator**


Personnage
- recevoirDegats
- coupDePoing
Classe Mère
« Hérite de »
Guerrier
- frapperCommeUn SourdAvecUnMarteau
Classe Fille
Mere
Fille
Eléments issus de la classe Mère
Eléments propres à la classe Fille

La classe Guerrier possèdera 3 méthodes :
- 2 de la classe dont elle hérite : recevoirDegats et coupDePoing
- 1 qui lui est propre : frapperCommeUnSourdAvecUnMarteau

En passant par `objetMere`, on ne pourra accéder qu’aux éléments de `objetFille` qui sont issus de la classe `Mere` (attributs et méthodes hérités de `Mere`).

```
Mere *objetMere(0);
Fille *objetFille = new objetFille();

objetMere = objetFille;
```

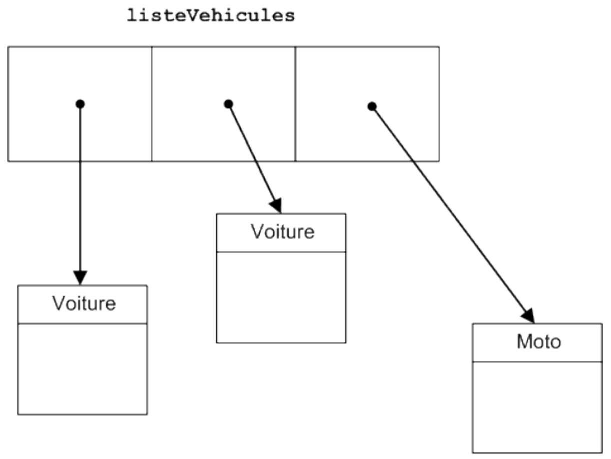
Polymorphism

- Polymorphism: Same code that generates different results according to type passed in input

→ We can manipulate daughterObject via pointers/references on motherClass

- Static resolution of links:
  - **In main**: functionName(motherClass object); //Function gets motherClass → Function uses methods of motherClass
- Dynamic resolution of links:
  - Virtual methods:
    - **In .h file**: virtual functionName(); //Add ‘virtual’ to prototype of function in .h file (not .cpp file)
    - PS: Not necessary to do it for daughterClass, but preferable for organizational purposes
  - Using pointers (1):
    - **In main**: functionName(motherClass const& object); //If object is daughterClass → Function uses methods of daughterClass
  - Using references (2):
    - **In main**: motherClass* object = new className; object->method();
- PS: Constructor never be ‘virtual’ because we already know the type when we created it
- PS: Destructor must be ‘virtual’ when we use polymorphisms
- PS: Even if destructor does nothing, we add it in .h and .cpp files

Heterogeneous Collection



- <vector motherClass*> listOfObjects; //Contains objects of “different” types
- listOfObjects.push_back (new daughterClass()); //Append an object
- listOfObjects[i]->method(); //Use a method on an object
- delete listOfObjects[i]; listOfObjects[i] = 0; //Delete an object
- Purely virtual method (PVM):
- **In .h file**: virtual funcName() const = 0; //Create a PVM
- → If a function is useful only for daughterClass but has to be defined for motherClass without doing anything
- Abstract class: Class that has atleast one PVM
  - **Problem**: We can’t create objects from abstract classes because we can’t call PVMs (since they don’t exist)
  - We can only ‘manipulate’ it if motherClass * motherObject = &daughterObject;
  - (VM) **Can** be redefined in daughterClass vs (**PVM**) **Must** be redefined in daughterClass

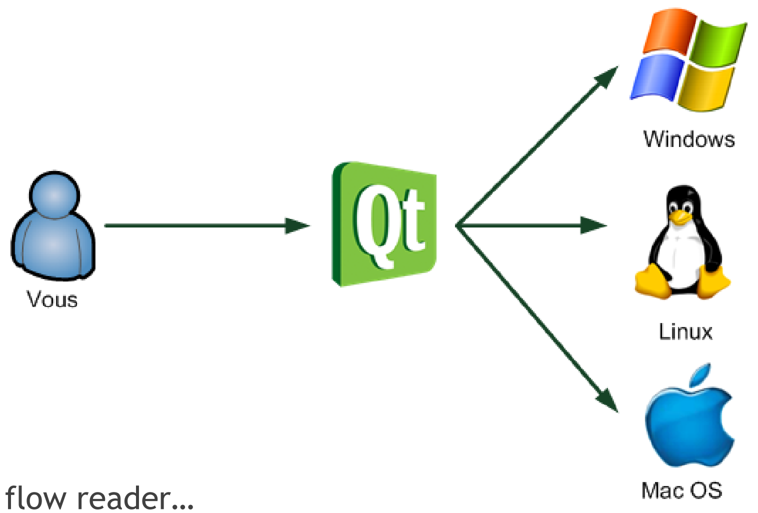
Static Methods / Static Attributes / Friendship

- Static method:
  - **In .h file**: static functionName();
  - **In main**: className::functionName();
  - Static method = Classic function
- Static attribute:
  - **In .h file**: static attributeType attributeName;
  - **In main** (rather before main) / **In .cpp file**: attributeType className::attributeName = value;
  - Static attribute = Global variable / May be useful to count class objects
- Friendship:
  - → We would like to declare methods that are used by some functions in private section
  - **In .h file, inside class{}**: friend functionPrototype;
  - → Function is friend with class → Function can get in private section of class

Qt – About

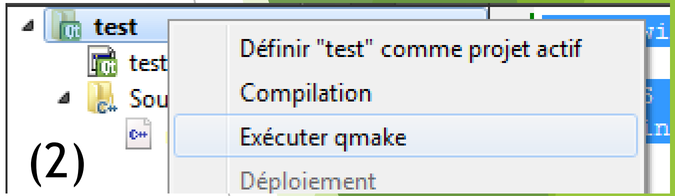
https://doc.qt.io/

- Qt = “Cute”
- GUI = Graphical User Interface
  - Windows: .NET
  - Mac OS X: Cocoa
  - Linux: Xlib, GTK+ (Gnome), Qt (KDE)
  - Portable multiplatform: .NET, GTK+, Qt, wxWidgets, FLTK
- Qt is a framework, a set of libraries, a set of modules
  - **Module GUI**: Window creation
  - **Module OpenGL**: 3D window creation
  - **Draw module**: 2D window drawing
  - **Network module**: Chat software, FTP client, Bittorent client, RSS flow reader…
  - **SVG module**: Flash and vector image creation
  - **Script module**: JS applications
  - **XML module**: Data exchange
  - **SDL Module**: Database modules (MySQL, Oracle, PostgreSQL…)
- Qt is LGPG liscence (free use)
- Qt Creator = C++ IDE & Window editor & Documentation
- Qt users = : Adobe, Archos, Boeing, Google, Skype, NASA, Google Earth



Qt - Basics

- #include <QApplication> //Include library
- QApplication app(argc, argv); //Create application object
- return app.exec(); //Execute application, program really starts at '.exec'
- Widget = Element of window: Buttons, images, cases…
- (1): QT += widgets //Must be added in .pro file before SOURCES and (2)
- Add DLL files situated in "C:\Qt\5.1.0\mingw48_32\bin" to .exe in order to send the file
- Compile with "Release" instead of "Debug" to have a .exe file with lower size
- QWidget window; //Create a window; Window = Widget that is not contained in another widget
- window.setFixedSize(width, height); //Set fixed size of window
- window.show(); //Show window
- PS: Methods of QWidgetName are just part of QWidget methods
- className::methodName(); //Call a static method → No need to create an object
- Good organization: .cpp file for every window
- We don't have to 'delete' after 'new' becausewhen widgetParent is deleted, all widgets inside it are deleted



(2)

Qt – Widgets

- **#include <QtWidgets>** //Include everything related to widgets
- QWidget widget(widgetParameters,&widgetParent); //Create widget object inside widgetParent
  - Widget container: Place widgets inside other widgets
  - widget.Attribute(); //Method to get widget
  - widget.setAttribute(); //Method to set widget
- Properties for all widgets:
  - setCursor(Qt::PointingHandCursor); //Change cursor when it is hovered over a widget
  - setEnabled(false); //Indicates whether the widget is enabled/can be changed
  - ToolTip; //Help text on widget when cursor is hovered
  - Height, Width, Size, Visible, Move(x,y), Geometry(x,y,w,h) //Indicates dimensions and visibility of widget
  - quit() //Slot that quits widget
- Properties for widow widget:
  - setWindowsFlags(Qt::WindowType); //Series of options controlling the behavior of the window
  - setWindowIcon(QIcon("Icon.*")); //Change icon of window
  - setWindowTitle("newWindowTitle"); //Change title of window
- QDialog = Dialog box = Small secondar window:
  - exec(); //Slot that opens dialog box in modal way

Qt – Widgets – Buttons

- Buttons:
  - QPushButton: Classic button
    - clicked(); //Signal when button is activated
    - pressed(); //Signal when button is pressed
    - released(); //Signal when button is released
  - QCheckBox: Checkbox button
    - QCheckBox *checkbox = new QCheckBox("checkBoxLabel", &window); //Create a checkbox
    - stateChanged(bool); //Signal when button state is changed
    - isChecked(); //Signal when button is checked
  - QRadioButton: Radio button
    1. QGroupBox *groupbox = new QGroupBox(“groupBoxName", &window); //Create a group box to group radio buttons
    2. QRadioButton *button = new QRadioButton("buttonName"); //Create a radio button
    3. button->setChecked(true); //Make radio button ‘checked’ by default
    4. groupbox->setLayout(layoutThatContainsButtons); //Add radio button to group box

Qt – Widgets – Text Fields (1)

- Text fields:
  - QLineEdit: Single-line text field
    - Text; //Recover and modify the text contained in the field
    - Alignment; //Edit alignment of the text inside
    - setEchoMode(QLineEdit::Password); //Type of text display
    - InputMask; //Define an input mask (intege, double…)
    - MaxLength; //Maximum number of characters that can be entered
    - ReadOnly; //Contents of the text field cannot be modified
      - Difference with 'enabled', we can still copy-paste the contents with 'readOnly'
    - returnPressed(); //Signal when the user presses 'Enter'
    - textChanged(); //Signal when the user has changed the text
  - QTextEdit: Multi-line text field
    - plainText, HTML //Retrieve and edit the content as plain text or HTML-enriched text
  - QSpinBox: integer input text field
    - Accelerated; //Allow the spinbox to accelerate the modification of the number if you press the button a long time
    - Minimum; //Minimum value that the spinbox can take
    - Maximum; //Maximum value that the spinbox can take
    - SingleStep; //No increment (default of 1) //If you want the buttons to vary the spinbox from 100 to 100, this is the property you have to change!
    - Value; //Value contained in the spinbox.
    - Prefix; //Text to display before the number
    - Suffix; //Text to display after the number

- QFont **Font** //QFont(fontName, fontSize, boldness{0→99}, italic{true/false})
  - **QFont::Bold** = 75 (Enumeration = Predefined constant from Qt library)

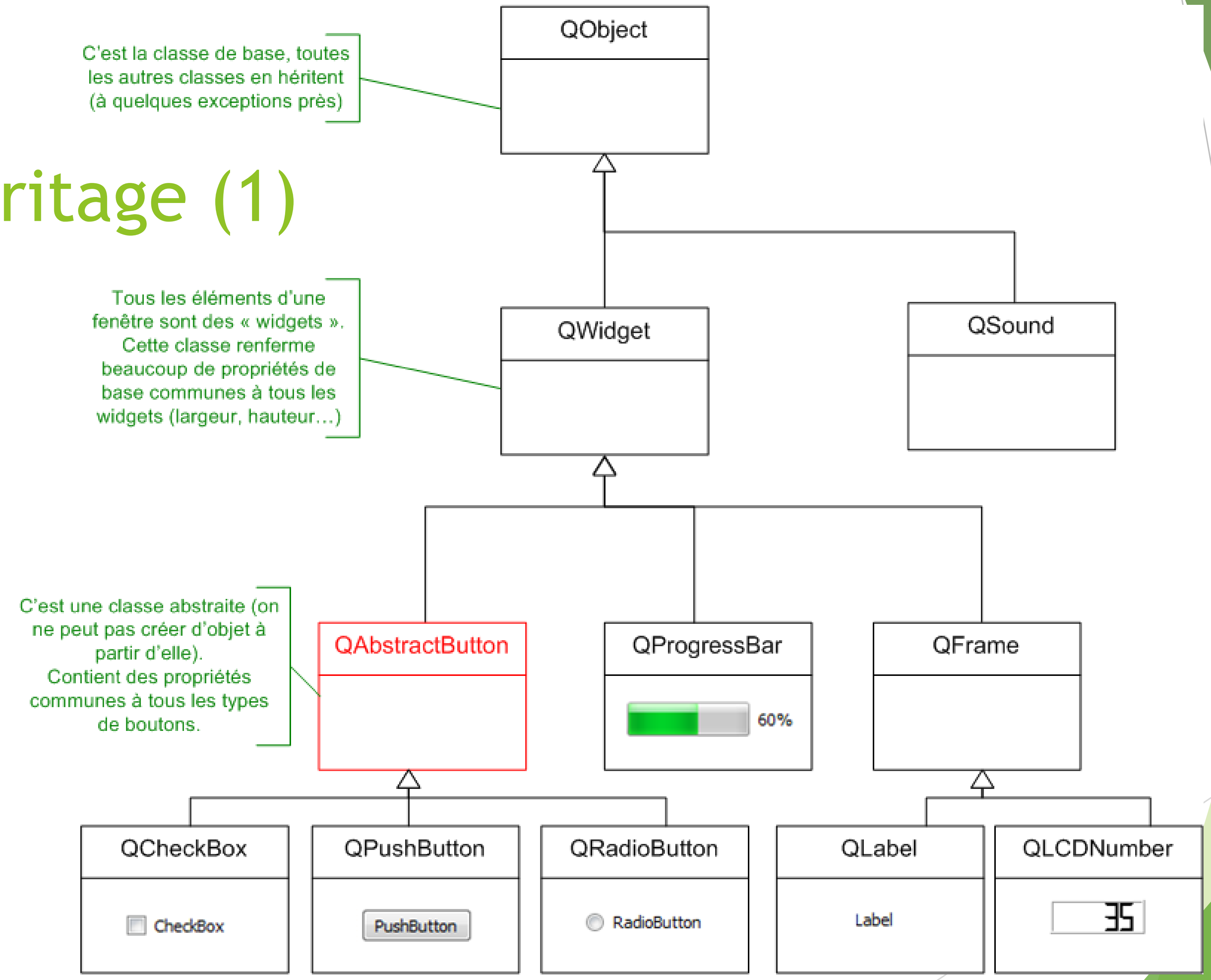
Qt – Widgets – Text Fields (2)

- QDoubleSpinBox: Single-line text field for float numbers
  - Same properties as QSpinBox
  - Decimals; //Handle the number of digits after the decimal point
- QSlider: Cursor to select a value
  - Same properties as QSpinBox
  - Range; //Set range of values for slider
  - Orientation; //Define the orientation of the slider (vertical or horizontal)
  - ValueChanged; //Signal if value is changed
- QComboBox: a drop-down list
  1. QComboBox *list = new QComboBox(&window); //Create a drop-down list
  2. list->addItem("itemName"); //Add item to list
  - Count; //Number of items in the drop-down list
  - CurrentIndex; //Index number of the currently selected element. (0→…)
  - CurrentText; //Text corresponding to the selected item
  - Editable; //Indicates whether the widget allows adding custom values or not (like a text field) //By default, adding new values is prohibited
  - CurrentIndexChanged(); //Signal when a new element is selected
  - Highlighted(); //Signal when element is overflowed by mouse (return int or string)

Qt – Widgets – Displayers & Containers

- Displayers:
  - QLabel: Show text or image
    - QLabel *label = new QLabel("labelText", &window); //Create a text label
    - setText("newLabelText); //Change text in text label
      - Alignment property allows you to set the alignment of text in the label
      - You can write HTML code in the label to apply formatting (bold text, hyperlinks, etc.)
    - QLabel *label = new QLabel(&window); label->setPixmap(QPixmap("icon.png")); //Create an image label
  - QProgressBar: Show progression bar
    - Maximum; //The maximum value that the progress bar can take
    - Minimum; //The minimum value that the progress bar can take
    - setValue(valueBetween0and100); //Change the value of the progress bar
    - ValueChanged(); //Signal when value is changed
  - QLCDNumber: Show LCD Number
- Containers:
  - QFrame: A widget that can have a border → Used to group other widgets inside
  - QGroupBox: A widget to manage checkboxes and radio buttons
  - QTabWidget: A widget that creates tabs (Can contain only one widget at a time)
    1. QTabWidget *tabs = new QTabWidget(&window); //Create a QTabWidget
    2. QWidget *page1 = new QWidget; //Create a QWidget for each of tab of QTabWidget, without giving them a parent widget
    3. //Place child widgets in each of these QWidget to populate the content of each page
    4. tabs->addTab(page1, "tabName"); //Create the tab pages by specifying the address of the QWidget that contains the page

Qt – Heritage (1)

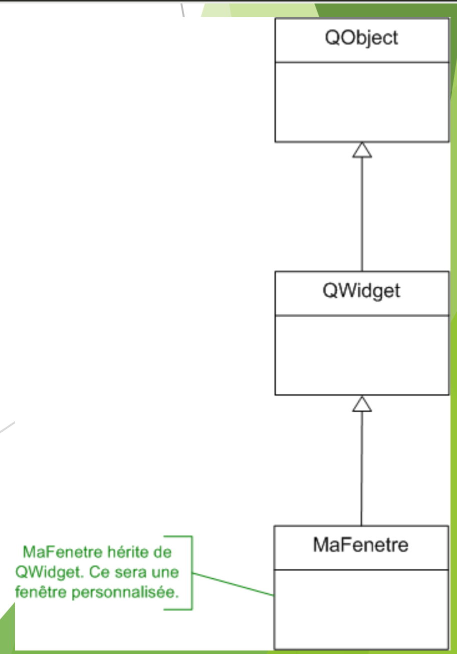

C'est la classe de base, toutes les autres classes en héritent (à quelques exceptions près)
QObject
Tous les éléments d'une fenêtre sont des « widgets ». Cette classe renferme beaucoup de propriétés de base communes à tous les widgets (largeur, hauteur…)
QWidget
QSound
C'est une classe abstraite (on ne peut pas créer d'objet à partir d'elle). Contient des propriétés communes à tous les types de boutons.
QAbstractButton
QProgressBar
60%
QFrame
QCheckBox
CheckBox
QPushButton
PushButton
QRadioButton
RadioButton
QLabel
Label
QLCDNumber
35

Qt – Heritage (2)

```cpp
#ifndef DEF_MAFENETRE
#define DEF_MAFENETRE

#include <QApplication>
#include <QWidget>
#include <QPushButton>

class MaFenetre : public QWidget // On hérite de QWidget
{
    public:
    MaFenetre();

    private:
    QPushButton *m_bouton;
};

#endif
```

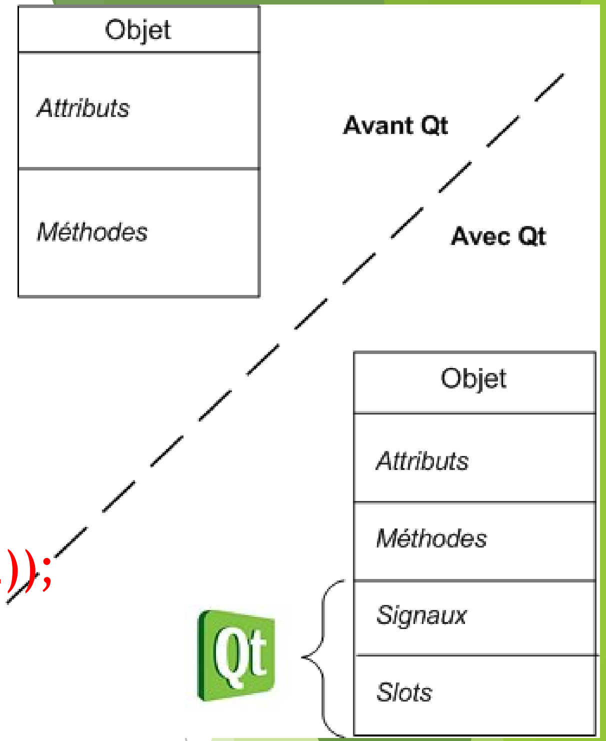
```cpp
#include "MaFenetre.h"

MaFenetre::MaFenetre() : QWidget()
{
    setFixedSize(300, 150);

    // Construction du bouton
    m_bouton = new QPushButton("Pimp mon bouton !", this);

    m_bouton->setFont(QFont("Comic Sans MS", 14));
    m_bouton->setCursor(Qt::PointingHandCursor);
    m_bouton->setIcon(QIcon("smile.png"));
    m_bouton->move(60, 50);
}
```

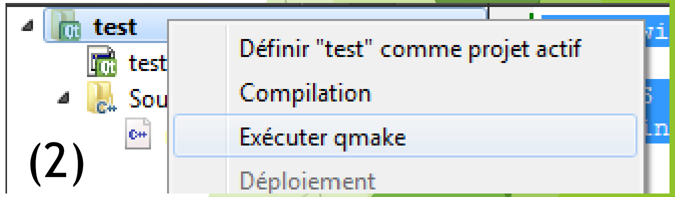
```cpp
#include <QApplication>
#include "MaFenetre.h"


int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    MaFenetre fenetre;
    fenetre.show();

    return app.exec();
}
```


QObject
QWidget
MaFenetre
MaFenetre hérite de QWidget. Ce sera une fenêtre personnalisée.

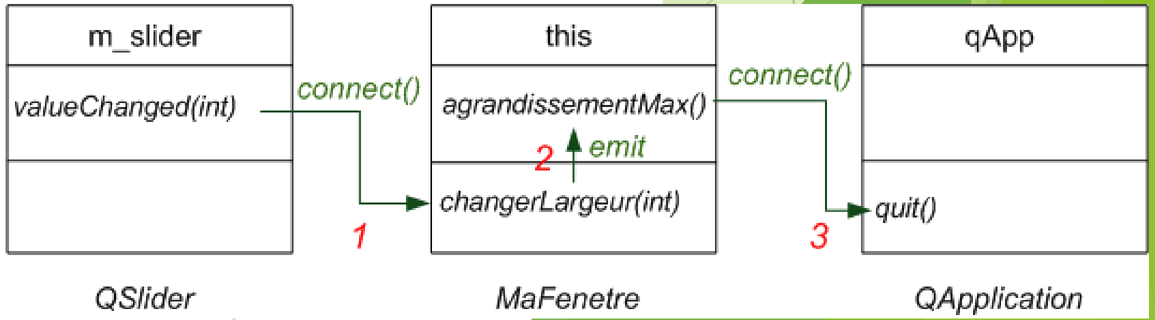
Qt – Signals & Slots

- Signal = Message sent by widget when an event happens
  - Signal can be called as a normal method: object.signal();
- Slot = Function called when an event happens
  - → Signal calls slot, they must work with the **SAME** type
- QObject::connect(senderWidget, SIGNAL(method1), recieverWidget, SLOT(method2));
  - Static method to connect between two widgets
  - SIGNAL and SLOT are preprocessing stuff done by Qt
  - recieverWidget = qApp //Pointer on QAppliccaiton created when we #include<QApplication>
- <u>Create your own slot</u>:
  - (1) Write Q_OBJECT in the start of the class and (2)
  - public slots:
  - slotFunction();
- <u>Create your own signal</u>:
  - **In .h file:** signals:
  - void signalFunction(); //Signal always return 'void'
  - **In a slot function in .cpp file:** emit signalFunction();



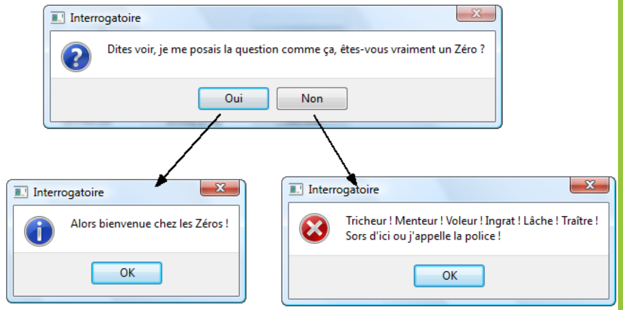


(2)



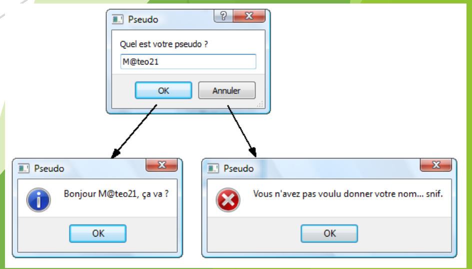
Qt - Common Dialogs (1)

- Dialog is a modal window → window that “blocks” temporarily its parent waiting a response from user
- #include <QMessageBox> //Class to show message dialog
  - QMessageBox::information(this, "windowTitle", "windowText <HTML>");
  - QMessageBox::warning(this, "windowTitle", "windowText <HTML>");
  - QMessageBox::critical(this, "windowTitle", "windowText <HTML>");
  - Int answer QMessageBox::question(this, "windowTitle", "windowText", QMessageBox::Yes | QMessageBox::No);
    - Choose between two buttons (predefined values)
    - Returns an integer that can be equal to QMessageBox::Yes or QMessageBox::No
  - QMessageBox::critical(this, "windowTitle", "windowText <HTML>");
- #include <QInputDialog> //Class to show text dialog
  - QString QInputDialog::getText(this, "windowTitle", "windowText", QLineEdit::Normal, QString(), &ok));
    - QLineEdit=type of writing{normal, password…}; //QString=text by default in dialog; ok=bool value to tell if button is clicked or not
    - str.isEmpty(); //Return if str is empty or not
  - QInputDialog::getInteger();
  - QInputDialog::getDouble();
  - QInputDialog::getItem(); //Choose an item from a list


Interrogatoire
Dites voir, je me posais la question comme ça, êtes-vous vraiment un Zéro ?

Interrogatoire
Alors bienvenue chez les Zéros !

Interrogatoire
Tricheur ! Menteur ! Voleur ! Ingrat ! Lâche ! Traître !
Sors d'ici ou j'appelle la police !

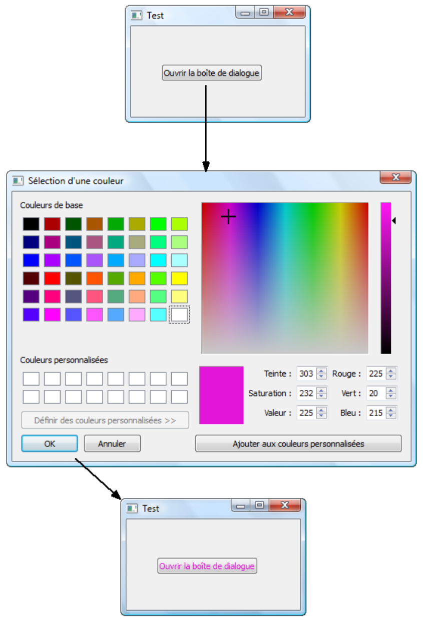

Pseudo
Quel est votre pseudo ?
M@teo21

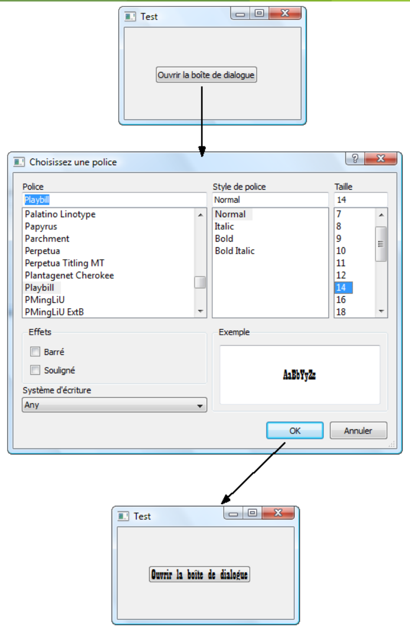
Pseudo
Bonjour M@teo21, ça va ?

Pseudo
Vous n'avez pas voulu donner votre nom... snif.

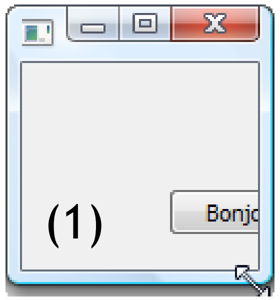
Qt - Common Dialogs (2)

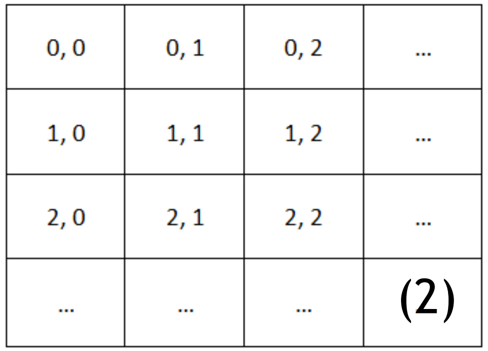
- include <QFontDialog> //Class to show font dialog
  - QFont QFontDialog::getFont(&ok, standardFont, this, "textWindow");
- include <QColorDialog> //Class to show color dialog
  - QColor color QColorDialog::getColor(Qt::white, this);
  - QPalette palette; palette.setColor(QPalette::ButtonText, color);
  - widget.setPalette(palette);
- include <QFileDialog> //Class to show file dialog
  - QString folderDirectory = QFileDialog::getExistingDirectory(this);
  - QString fileDirectory = QFileDialog::getOpenFileName(this, "textWindow", QString(), “type(*.extension)");
  - QString file = QFileDialog::getSaveFileName(this, "textWindow", QString(), "type(*.extension)");

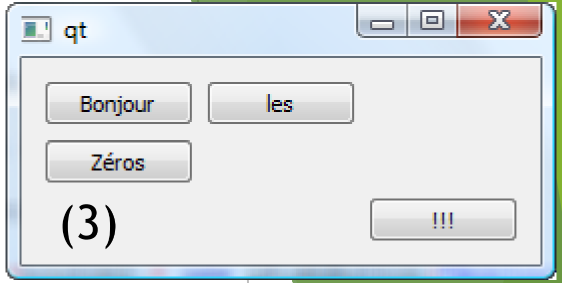


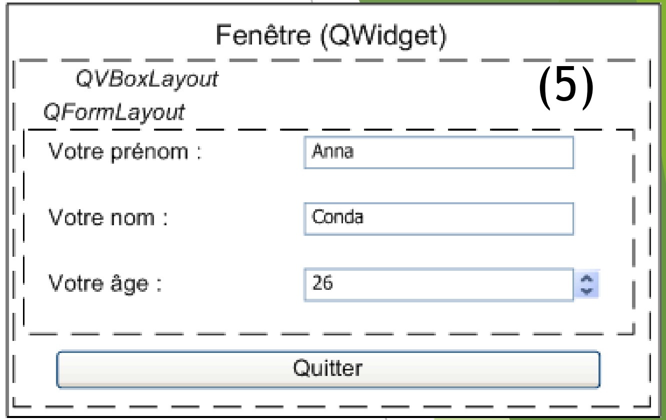


Qt - Layouts









- Absolute positioning: (1)
  - Problem: Widgets don't change when window is changed
  - Soltuion: setFixedSize();
    - Problem: Varies from a screen to another
- Relative positioning: Done with widget containers : layouts
- #include <QLayoutName> //LayoutName = HBoxLayout, VBoxLayout, QGridLayout (2,3), QFormLayout
- **Layout structure:**
  1. QWidget *widget = new QWidget(); //Create widget
  2. QLayoutName *layout = new QLayoutName; //Create layout
  3. layout->addWidget(widget, x, y, rowSpan, columnSpan); //Place widget in layout //(x,y,r,c) for QGridLayout

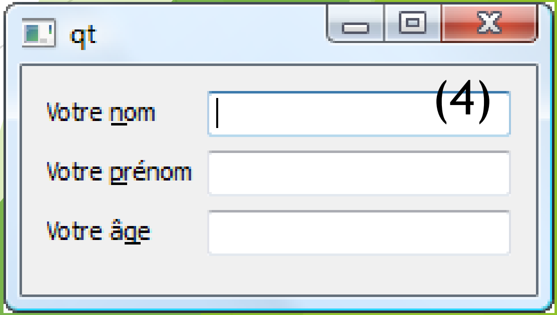
     PS: rowSpan and columnSpan won't work if numbers don't fit

     layout->addRow("textLabel", textWidget); //Place widget in form layout and use "Alt" to use it (4)

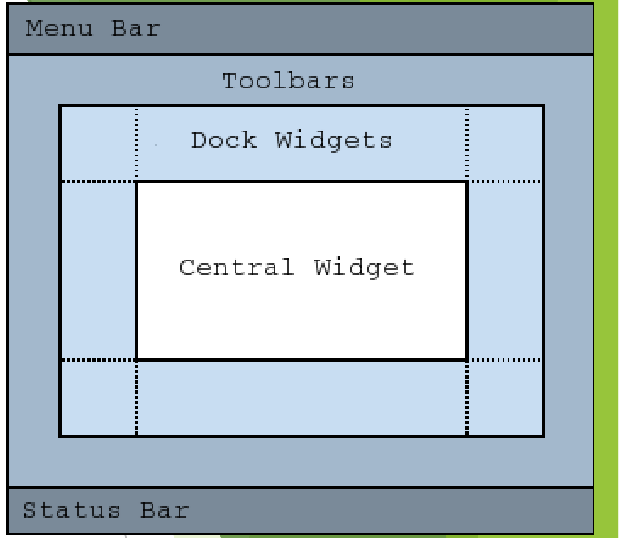
     PS: Place a "&" symbol in front of the letter of the label you want to turn into a shortcut

     PS: Place a "&&" symbol to write the "&" symbol in textLabel
  4. window.setLayout(layout); //Tell window to use created layout

  PS: principalLayout.addLayout(layout); //Add layout inside principalLayout (5)



Qt – Main Window (1)

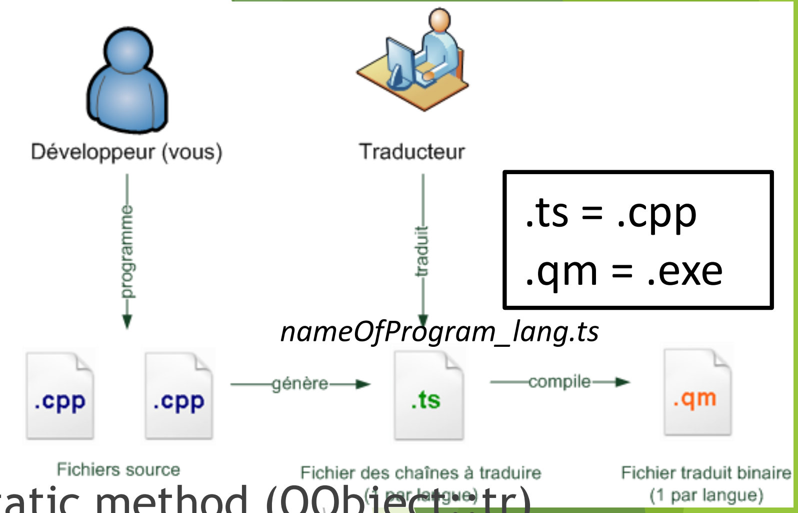


- class myWindow : public QMainWindow //Create a class from Main Window
  - Central widget contain one and only one widget (like in tabs)
  - SDI (Single Document Interface) ◻ Display one document at a time
    1. QWidget *centralWidget = new QWidget; setCentralWidget(centralWidget); //Create SDI
  - MDI (Multiple Document Interface) ◻ Display multiple documents at once (Subwindows)
    1. QMdiArea *centralWidget = new QMdiArea; //Create MDI
    2. QMdiSubWindow *subWindow1 = centralWdiget->addSubWindow(widgetName); //Create sub window
    3. subWindow1.removeSubWindow(); //Remove subwindow
    4. subWindowList(); //Show list of sub windows in centralWidget
    5. setCentralWidget(centralWidget); //Show MDI
  - Menus:
    - QMenu *menuName = menuBar()->addMenu("menuName"); //Create menu object in menu bar
    - QAction *actionName = new QAction("actionName", &mainWindow); //Create an action object
    - menuName->addAction(actionName); //Add action object to menu object
    - QMenu *subMenuName = menuName->addMenu("subMenuName"); //Create sub menu object in menu object
    - subMenuName->addAction("subSubMenuName"); //Create sub sub menu object in sub menu object in menu object
  - You can create custom contextual menus. A contextual menu is a menu that appears when you right-click a widget.

Qt – Main Window (2)

- Actions:
  - triggered() & hovered() //Use actions as signals (triggered = chose by user)
  - actionName->setShortcut(QKeySequence("Ctrl+letter")); //Set shortcut to action
  - actionName->setIcon(QIcon("icon.png")); //Set icon to action
  - actionGras->setCheckable(true/false); actionGras->isChecked(); //Make action checkable and check its value
- Toolbars:
  - QToolBar *toolBarName = addToolBar("toolBarName"); //Create toolbar
  - toolBarName->addAction(actionName); //Add action object to toolbar
  - toolBarName->addWidget(widgetName); //Add widget object to toolbar
  - toolBarName->addSeparator(); //Add separator between toolbar objects

Qt – Translation

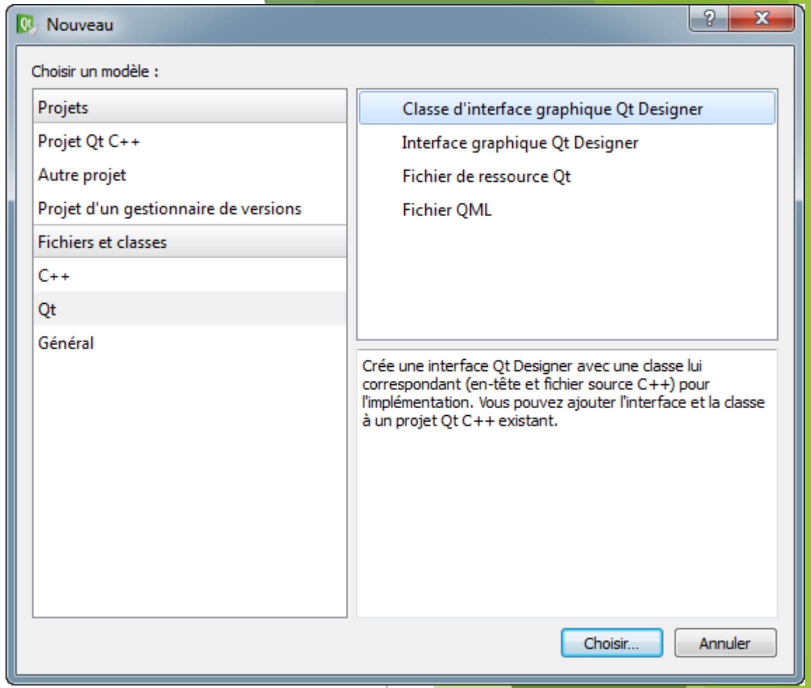


- Unicode = Norm that indicates how characters are edited inside a computer
  - Qstring → Adapted to translation
  - char[] → Not adapted to translation
- ("message") → tr("message"); //Indicate the string that should be translated //Static method (QObject::tr)
- tr("message", "messageToExplain"); //Add an explaining message to the translator
  - PS: Useful for "ctrl+letter"
- tr("pluralMessage %n", " ", number); //Edit translation of plural sentences

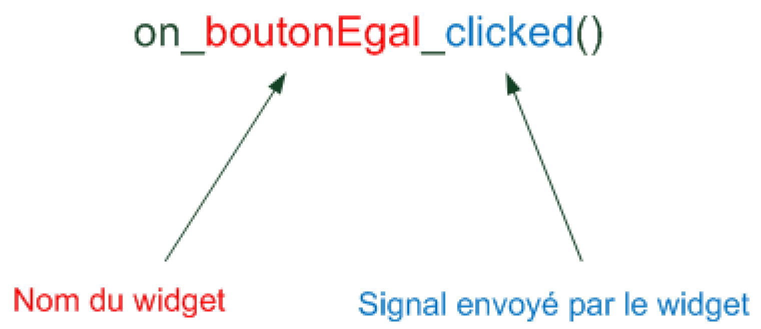
1. TRANSLATIONS = zeroclassgenerator_lang.ts //Need to be added in .pro file
2. lupdate NomDuProjet.pro //Update .ts file in Qt Command Prompt
3. Open Qt Linguist and make the necessary translations
4. lrelease nomDuProjet.pro //Update .ts file in Qt Command
   - **In main:** QString local = QLocale::system().name().section('_', 0, 0); //Get PC language from Windows bar
   - QTranslator translator; //Create translator object
   - translator.load(QString("zeroclassgenerator_") + local); //Load .ts file
   - app.installTranslator(&translator); //Add translator to application

Qt – Designer

- dialog.ui: This is the file that will contain the GUI (XML type). It is this file that we will modify with the Qt Designer editor.
- className::className() : … ui(new Ui::className)
- { ui->setupUi(this);
- connect(ui->widgetName, SIGNAL(clicked()), this, SLOT(slotName())); }
- #include "ui_personalDesign.h" in .h file

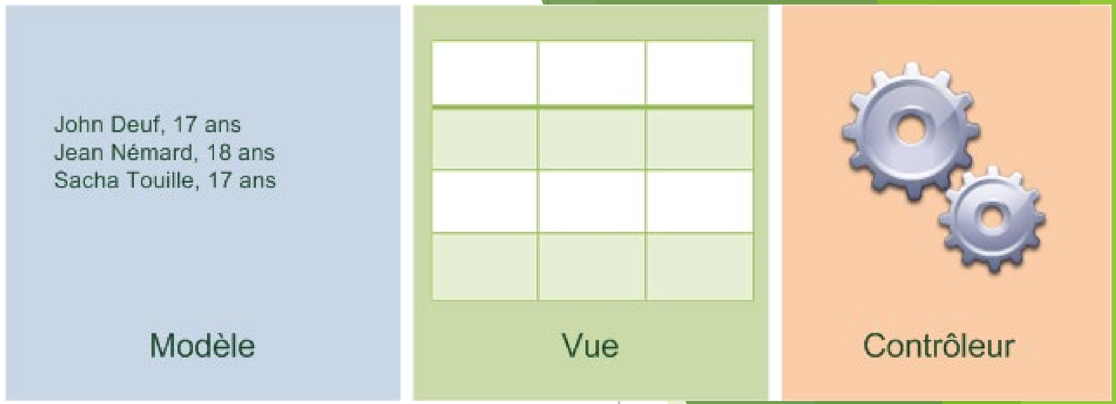


Crée une interface Qt Designer avec une classe lui correspondant (en-tête et fichier source C++) pour l'implémentation. Vous pouvez ajouter l'interface et la classe à un projet Qt C++ existant.

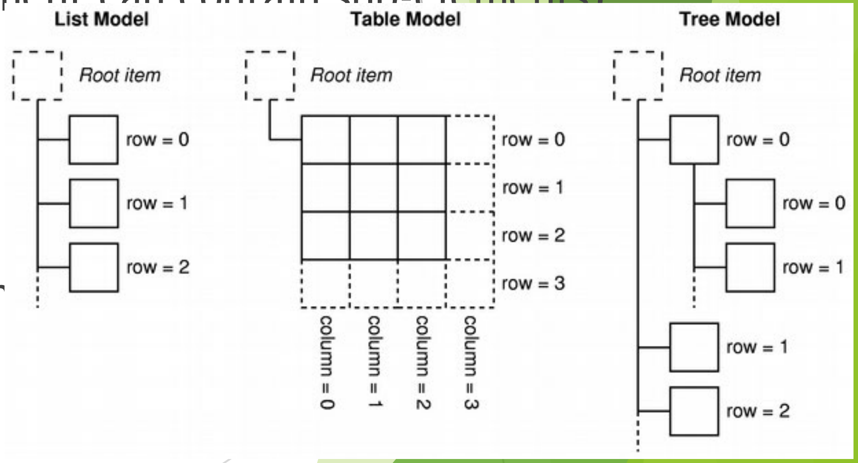


Give your slot a precise name and the connection will be automatic!

Qt – MVC Architecture (1)



- MVC = Model-View-Controller
  - Model: Contains the data
  - View: Deals with the display → Displays what the model contains
  - Controller: “Reflection" part of the program
- Models:
  - **QStringListModel**: A list of character strings, of type QString
  - **QStandardItemModel**: A list of elements organized as a tree (each element can contain sub-elements)
  - **QDirModel**: The list of files and folders stored on your computer
- Views:
  - **QListView**: A list of elements
  - **QTreeView**: An element tree, where each element can have child elem
  - **QTableView**: An array
- QDirModel *model = new QDirModel; //Create directory model
- QTreeView *view = new QTreeView; //Create tree view
- view->setModel(model); //Set model to view



Qt – MVC Architecture (2)

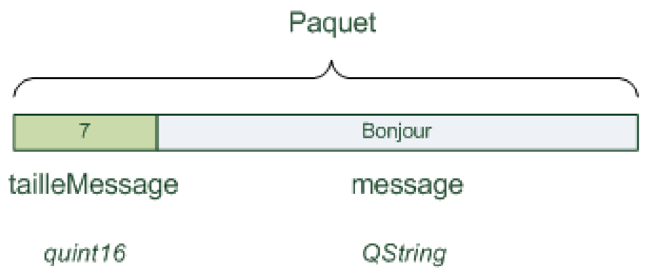
- QStringList list; list << element1 << element2; list.append("element"); //Create list and add elements
- QStringListModel *model = new QStringListModel(list); //Convert list to ListModel
- QStandardItemModel *model = new QStandardItemModel(rows, columns); //Create ItemModel (rows and columns are optional)
- QStandardItem *item = new QStandardItem("itemName"); //Create item
- model->appendRow(item); model->appendColumn(item); //Append rows and columns to ItemModel
- item->appendRow(new QStandardItem("itemName")); //Append row to item (sub item)
- model->setItem(i, j, new QStandardItem("text")); //Add text to ItemModel
- view->header()->hide(); //Hide header

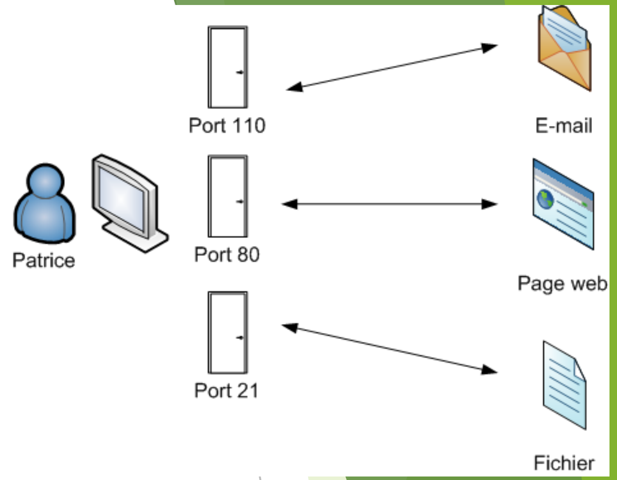
1. QItemSelectionModel *selection = view->selectionModel(); //Get what is selected on view
2. QModelIndex indexElementSelected = selection->currentIndex(); //Get index of selected item

   QModelIndexList listeSelected = selection->selectedIndexes(); //Get indexes of selected items
3. QVariant elementSelected = model->data(indexElementSelected, Qt::DisplayRole); //Get content from index
4. elementSelected.toString(); //Convert selected element into string

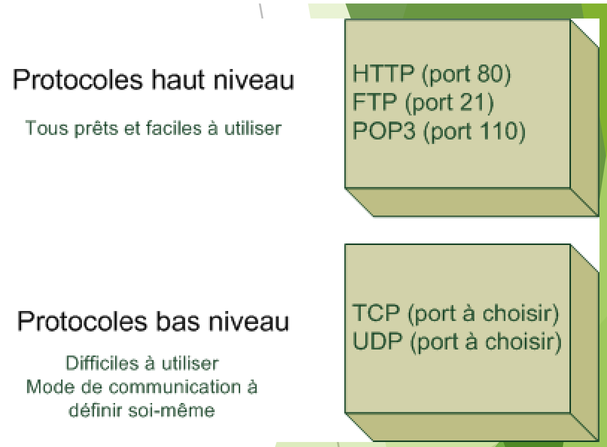
- view->setSelectionMode(QAbstractItemView::ExtendedSelection); //Select more than one element

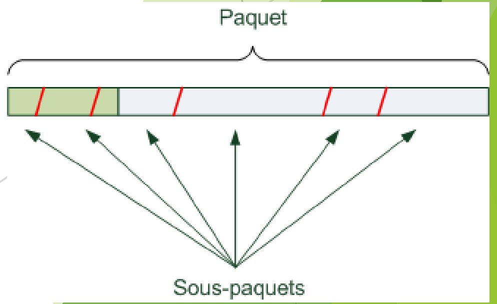
Qt – Networking





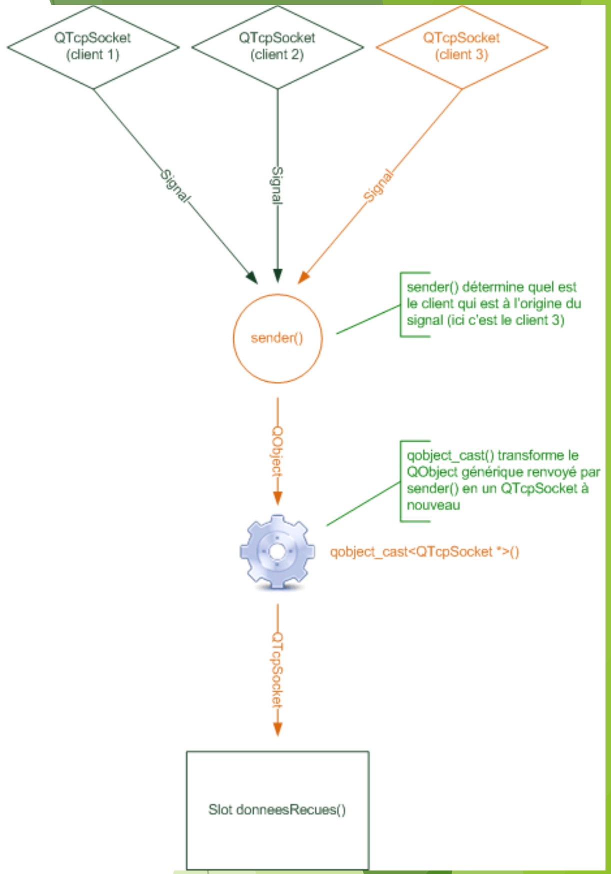
- To make 2 programs communicate with each other via the network:
  1. Know the <u>IP address</u> identifying the other computer
  2. Use a free and open <u>port</u>
  3. Use the same data transmission <u>protocol</u>
- Different IP addresses:
  - <u>Internal IP</u>: Localhost/Loopback → Communicate to oneself (127.0.0.1)
  - <u>Local network IP</u>: Communicate to the local network (192.168.0.3) (ipconfig / ifconfig)
  - <u>Internet IP</u>: Communicate to the global network (86.79.12.105) (internet website)
- PS: Common ports are from 1 to 1024, and uncommon ports are from 1024 to 65535 (be careful from firewalls)
- <u>A protocol</u> is a set of rules that allow 2 computers to communicate
  - <u>TCP protocol</u>: Control system + Slow
  - <u>UDP protocol</u>: No control system + Fast
- <u>A client / server architecture</u>: Server distribute the communications between the clients
- <u>Peer-To-Peer Architecture</u>: Each customer communicate directly with another customer
- PS: lengthOfMessage will be of type quint16, not int, because int changes from a machine to another, quint16 doesn't

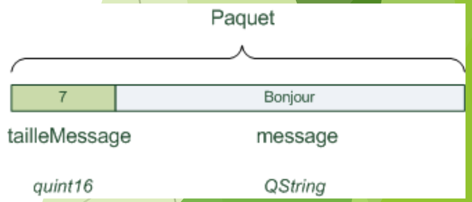




Qt – Servor Code

- QT += widgets network //Add network to .pro file
- QTcpServer *servor //In .h file → Object to present the server on the nework
- QList<QTcpSocket *> clients //In .h file → Array to present the connections with clients on the network
- Signals: servor → newConnection(); client → readyRead(); client → disconnected();
- bool servor->listen(QHostAddress::Any, portNumber); //Listen to clients
- QString::number(servor->serverPort()); //Get portNumber
- QTcpSocket *newClient = servor->nextPendingConnection(); //Accept new connection, i.e. new client
- bool QTcpSocket *socket = qobject_cast<QTcpSocket *>(sender()); //Find QTcpSocket of client
- QDataStream in(socket); //Receive the message
- socket->bytesAvailable() < (int)sizeof(quint16) //Compare and recieve the quint16 header
- in >> lengthOfMessage; //Add the whole lengthOfMessage when recieving is finished
- socket->bytesAvailable() < lengthOfMessage //Compare and recieve the whole message
- in >> message; //Add the whole message when recieving is finished
- clients.removeOne(socket); socket->deleteLater(); //Remove client from list and delete its socket later
- QByteArray paquet; QDataStream out(&paquet, QIODevice::WriteOnly); //Prepare the paquet
- out << (quint16) 0; out << message; //Reserve place to add lengthOfMessage //Add message to out
- out.device()->seek(0); out << (quint16) (paquet.size() - sizeof(quint16)); //Set cursor in the beginning //Erase 0 with length of size
- client->write(paquet); //Send paquet to client


QTcpSocket (client 1)
QTcpSocket (client 2)
QTcpSocket (client 3)
Signal
Signal
Signal
sender()
sender() détermine quel est le client qui est à l'origine du signal (ici c'est le client 3)
QObject
qobject_cast() transforme le QObject générique renvoyé par sender() en un QTcpSocket à nouveau
qobject_cast<QTcpSocket *>()
QTcpSocket
Slot donneesRecues()



Qt – Client Code’

- QTcpSocket *socket; //In .h file → Represent the servor in the network
- Signals: socket → connected(), disconnected(), readyRead(), error(QAbstractSocket::SocketError)
- socket->abort(); socket->connectToHost(servorIP->text(), serverPort->value()); //Connect to servor
- message->clear(); message->setFocus(); //Clear message and set focus on message box
- Type of errors: QAbstractSocket::HostNotFoundError, QAbstractSocket::ConnectionRefusedError, QAbstractSocket::RemoteHostClosedError

Standard Library

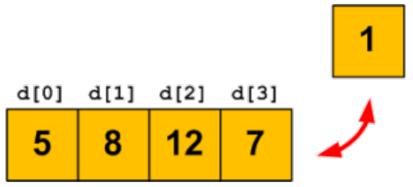
- Heritage from C: 15 C header files → C++
  - cmath //Math library
  - cctype //Char library
    - isalpha() //Check if the character is a **letter**
    - isdigit() //Check if the character is a **number**
    - islower() //Check if the character is a **lowercase**
    - isupper() //Check if the character is an **uppercase**
    - isspace() //Check if the character is a **space** or **a line break**
    - tolower() //**Convert** character to a **lowercase**
    - toupper() //**Convert** character to an **uppercase**
  - ctime //Time library
    - time(0) //Return the number of seconds that have elapsed since January 1, 1970 (UNIX time)
  - cstdlib //Standard library
    - rand() //Generate random number between 0 and 10^9 (Use % to obtain a random in a specific range)
    - srand(time(0)) //Initialize a sequence of random numbers (Must be called only one time)
- Streams: Make program communicate with exterior (write **cout**, read **cin**, file **fstream**)
- Standard Template Library: Containers (vectors…)

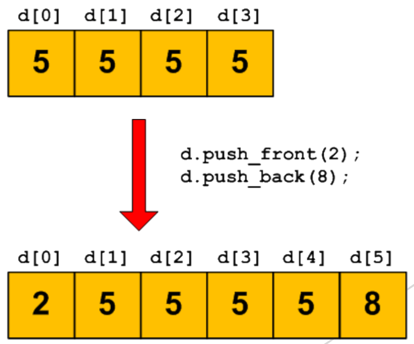
Standard Template Library (1)

- Container: Object that stock other objects → Basic element of STL
  - Sequences: vector, deque, list, stack, queue, priority_queue
    - sequenceName<type> T; //Declare a sequence

    PS: All the cases are arranged **contiguously** in the memory
  - Associative containers: set, multiset, map, multimap
    - associativeContainerName<keyType, valueType> T; //Declare an associative container

  PS: Do not forget #include <containerName>
- Common methods: size(); empty(); clear(); swap(); //Swap container A with container B (Must be with same type)
- Vectors:
  - push_back(newValue); //Add newValue in the end
  - pop_back(); //Delete last value
  - forhead(); //Get the first element
  - back(); //Get the last element
  - assign(value); //Fill all the elements with the same value
- Deques:
  - Same as vectors but you can add and delete from the beginning
  - push_front(newValue); // Add newValue in the beginning
  - pop_front(); //Delete first value





```
d.push_front(2);
d.push_back(8);
```

Standard Template Library (2)

- Stacks: (LIFO – Last In First Out)
  - push(newValue); //Add element
  - top(); //Get last added element
  - pop(); //Remove last added element
- Queues: (FIFO – First In First Out)
  - push(newValue); //Add element
  - front(); //Get first added element
  - pop(); //Remove last added element
- Priority_Queues: Organized queues
  - Elements in a priority_queue must be surcharged with the operato[illegible]n
- Maps: Dictionnary with keys and values
  - T[key] = value; //Add element to T

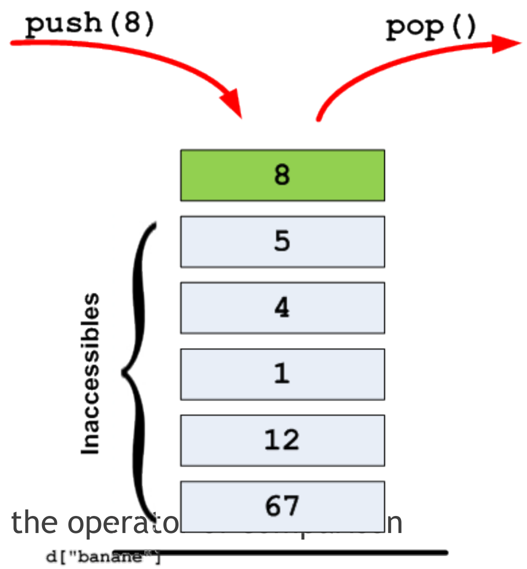
  PS: Items are sorted with their keys

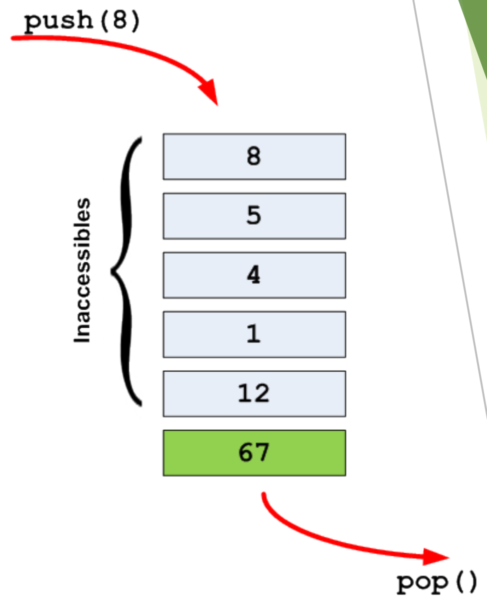
  PS: Used when we want to use keyType ≠ integer

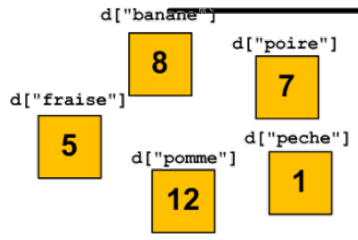
  PS: The [ ] operator gives access to a given element. If the element does not exist, the [ ] operator creates it
- Sets: Maps with keys=values

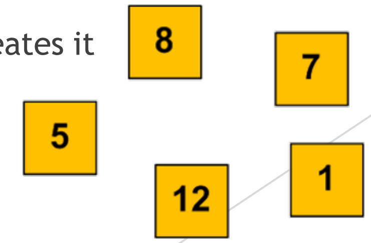
  Items are **unique** and **sorted** automatically

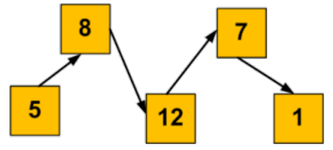
  PS: We can't work with []
- Multisets, multimaps: Sets and maps having more than one key
- Lists: Linked lists

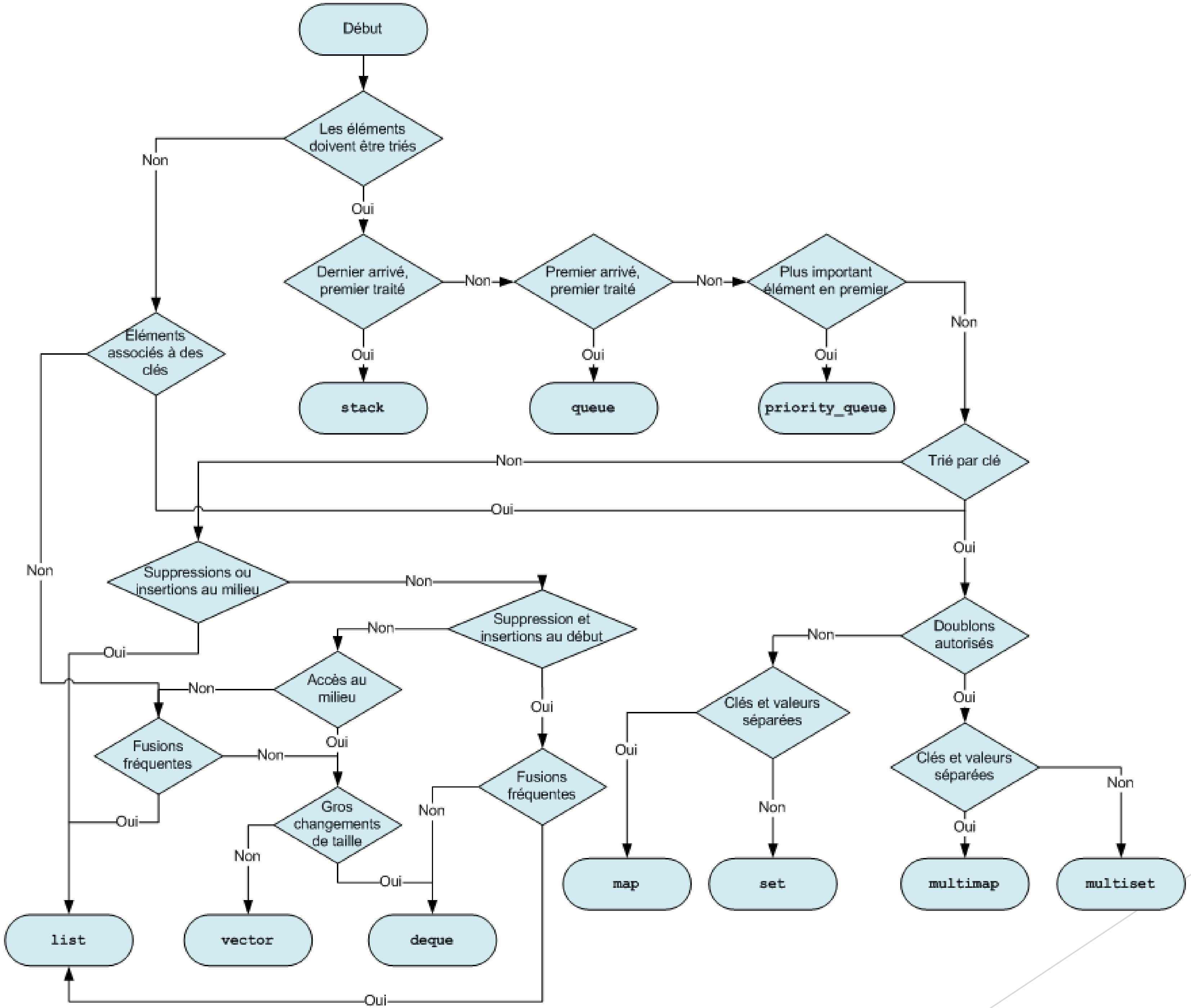











Début
Les éléments doivent être triés
Non
Oui
Dernier arrivé, premier traité
Non
Premier arrivé, premier traité
Non
Plus important élément en premier
Non
Oui
Oui
Oui
stack
queue
priority_queue
Éléments associés à des clés
Trié par clé
Non
Oui
Oui
Non
Suppressions ou insertions au milieu
Non
Oui
Suppression et insertions au début
Non
Oui
Accès au milieu
Non
Oui
Fusions fréquentes
Non
Oui
Gros changements de taille
Non
Oui
Fusions fréquentes
Non
Oui
Doublons autorisés
Non
Oui
Clés et valeurs séparées
Oui
Non
Clés et valeurs séparées
Oui
Non
list
vector
deque
map
set
multimap
multiset

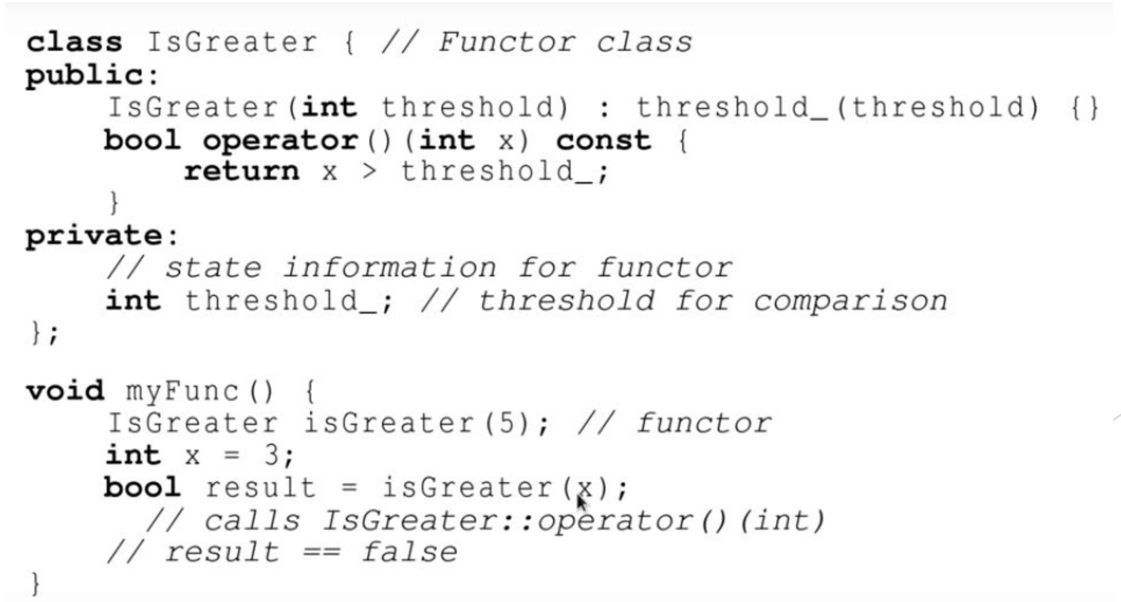
Iterators

- #include <iterator> //Include iterator library
- Iterators are pointer-like objects that will allow us to navigate through containers → Pointers abstractions
  - Bidirectional iterators: Advance only one item at a time
  - Random access iterators: Advance while jumping alot of items at a time
    - it = container.begin() + jumpingValue;

  Ps: Useful for **map** and **list**
- containerType <type>::iterator it; //Create an iterator
- for(it = container.begin(); it! = container.end(); ++it) //Increment with iterator
- *it //Get content of iterator
- tab.insert(it, newValue); //Insert newValue at iterator position
- tab.erase(); //Erase value at iterator position
- #include <utility>; pair<type1, type2> p(value1, value2); //Declare a pair
- p.first(); p.second(); //Get value1 and value2
- PS: map contains pairs (key, value)
- it = container.find(key); //Return value of key in container; return container.end() if key doesn't exist

Functors

- Functors are objects with operator overload () → Functions abstractions
  - class functorName{ public: returnType operator()(args) { /* code */ } }
  - In main: functorName f; f(args); //Create f object as functor name
- Functors are objects. They can therefore use attributes like any other class. This allows us to create a function with a memory. It can therefore perform a different operation on each call
- Predicates: Functors with 1 argument and return a Boolean → To test a particular property of the argument
- #include <functional> //Use predefined functors
- PS: map<type1, type2, comparisonFunctor> mapName; //Use a comparison functor to sort map elements



```
class IsGreater { // Functor class
public:
    IsGreater(int threshold) : threshold_(threshold) {}
    bool operator()(int x) const {
        return x > threshold_;
    }
private:
    // state information for functor
    int threshold_; // threshold for comparison
};

void myFunc() {
    IsGreater isGreater(5); // functor
    int x = 3;
    bool result = isGreater(x);
      // calls IsGreater::operator()(int)
    // result == false
}
```

Algorithms

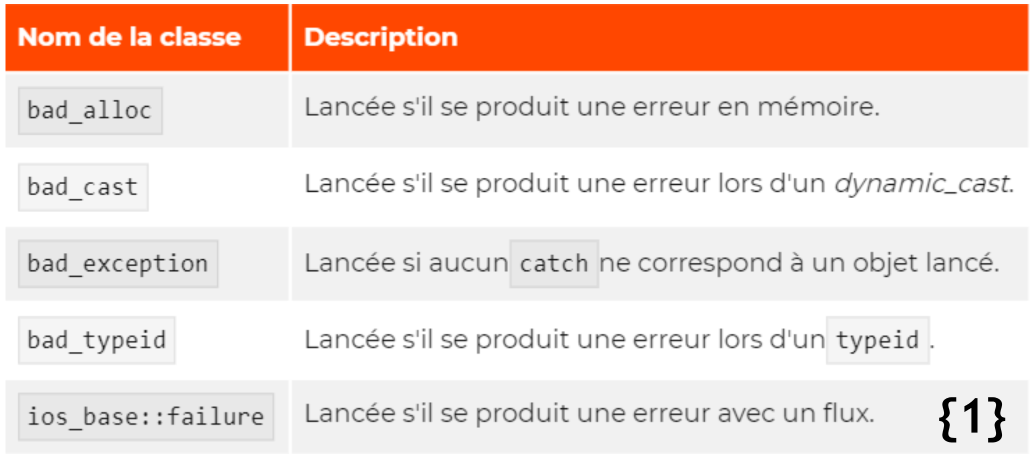
- #include <algorithm> //Include algorithm library
- generate(T.begin(), T.end(), functorName()); //Generate numbers in container
- count(T.begin(), T.end(), elementToCount); //Count element in container
- count_if(T.begin(), T.end(), predicateName()); //Count elements that verify predicate
- find(T.begin(), T.end(), elementToFind); //Find element in container (return iterator on end if it doesn't exist)
- find_if(T.begin(), T.end(), elementToFind); //Find elements that verify predicate
- min_element(); max_element(); //Return minimum and maximum value in container
- sort(T.begin(), T.end()); //Sort elements in container (Good with **vector** and **deque**, useless with **map**)
- sort(T.begin(), T.end(), comparisonFunctor()); //Sort elements using comparison functor
- for_each(T.begin(), T.end(), functorName()); //Iterate through all the elements in container and apply functor
- transform(T1.begin(), T1.end(), T2.begin(), T3.begin(), <double>()); //Add elements from T1 and T2 in T3

Iterators and Flows

- PS: Iterators on flows increment only (++). They do not decrement
- #include <iterator> //Do not forget to include iterator library
- ostream_iterator<type> it(cout/file, “delimeter”); //Create ostream iterator (and use iterator methods)
- istream_iterator<type> it(cin/file); //Create istream iterator (It needs to be advanced every step)
- istream_iterator<type> end; //Create istream that points on end (EOF)
- copy(it, end, T.begin()); //Copy content in T
- back_insert_iterator<vector<string> > it2(tableau); //Create an iterator that augments the container
  - The only difference is with the operator *. Instead of changing a case, the iterator adds a new case at the end of the table
- count(); min_element(); max_element(); //Some algorithms that exist for these iterators
- String flows: **ostringstream** and **istringstream**
- stringFlow << element; //Add element to flow(element can be string, integer, …)
- stringFlow.str(); //Get string from flow
- string::iterator it = str.begin(); //Create an iterator on string
- transform(str1.begin(), str1.end(), str2.begin(), functorName()); //Transform a string into another
- insert(); erase(); //We can use these methods too
- Static arrays: type* beg(array); type* end(array+size); //Create beginning and ending iterators
  - We can use algorithms now because pointers behave like random access iterators
- complex<type> c(realValue,imaginaryValue); //Create a complex object
- valarray<type> T(); //Create a **valarray** → vector that have the ability to perform mathematical operations directly with the entire array
- T.apply(functorName); //Apply functor name on valarray T

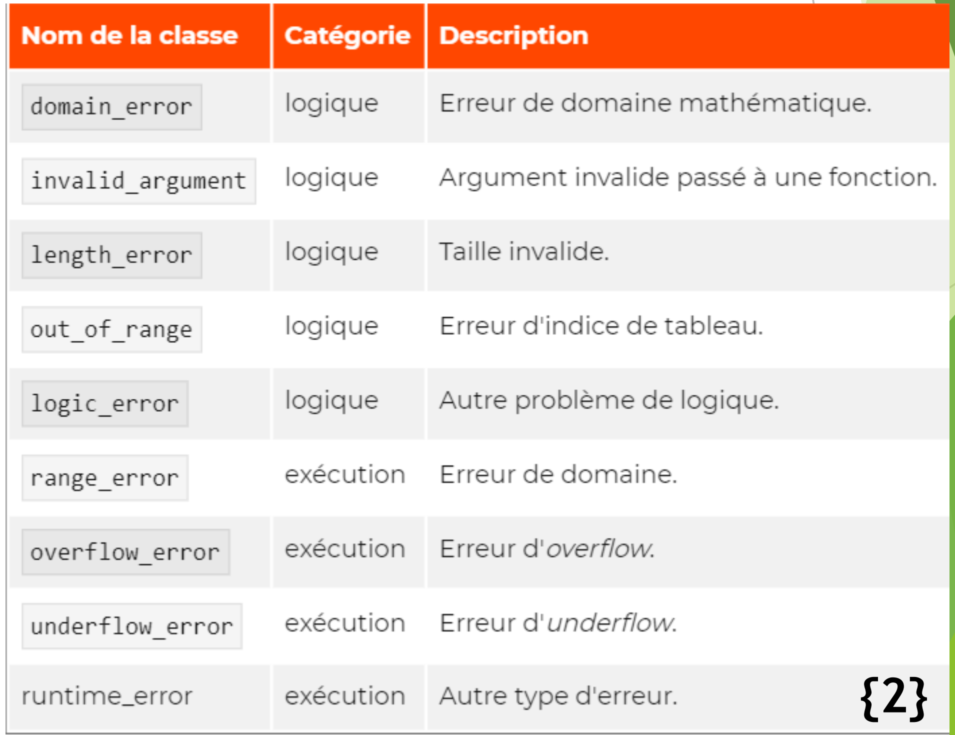
Exceptions

- In function: throw value
- In main:
  - try { /* Instruction to try that calls function that contains 'throw' */ }
  - catch(valueType const& variableName) { cerr << variableName << endl; }
- #include <exception> //Call class that deals with exceptions
- {1} catch(std::exception const& e) { cerr << "ERREUR : " << e.what() << endl; } //Use standard exceptions
- {2} throw className("errorMessage"); //Use standard class exceptions
- assert (/*expression*/); //Test whether an expression is true or not. (If true, program continues | If false, program stops and return error)



Nom de la classe	Description
`bad_alloc`	Lancée s'il se produit une erreur en mémoire.
`bad_cast`	Lancée s'il se produit une erreur lors d'un *dynamic_cast*.
`bad_exception`	Lancée si aucun `catch` ne correspond à un objet lancé.
`bad_typeid`	Lancée s'il se produit une erreur lors d'un `typeid`.
`ios_base::failure`	Lancée s'il se produit une erreur avec un flux.

{1}



Nom de la classe	Catégorie	Description
`domain_error`	logique	Erreur de domaine mathématique.
`invalid_argument`	logique	Argument invalide passé à une fonction.
`length_error`	logique	Taille invalide.
`out_of_range`	logique	Erreur d'indice de tableau.
`logic_error`	logique	Autre problème de logique.
`range_error`	exécution	Erreur de domaine.
`overflow_error`	exécution	Erreur d'*overflow*.
`underflow_error`	exécution	Erreur d'*underflow*.
runtime_error	exécution	Autre type d'erreur.

{2}

Templates

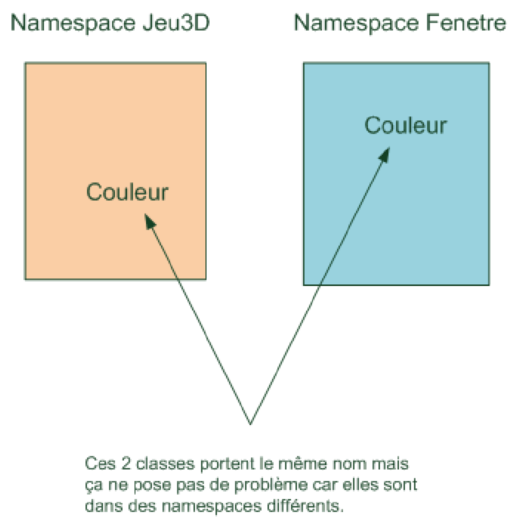
- Goal: Allow a function or class to use different types
- template <typename T, typename S> //Add S if we want to use another type
- T functionName(const T& argument) { /* code here */ }
- In main: functionName<**typeName1**, typeName2>(argument); //typeName2 if we want to use another type
  - PS: EVERYTHING must be in the .h file
- Specialization: //If we want to create a special behavior for a specific type
  - template <>
  - specificType functionName<specificType>(const specificType& arg) { /* code here */ }

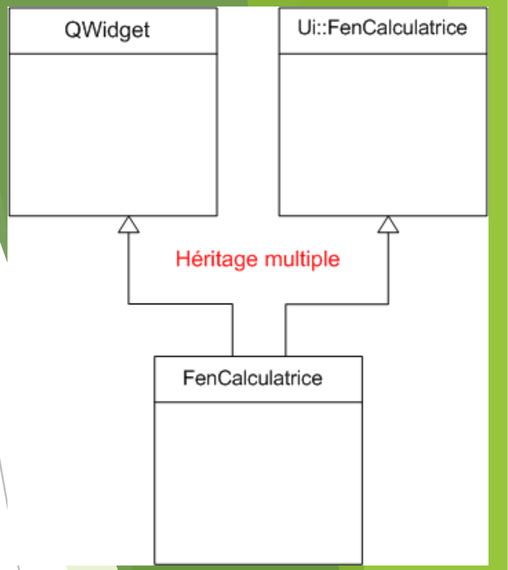
PS: It is used in the same way for classes

- T className<**T**>::functionName() { /* code here */ } //Be careful while creating the function in .c file
- In main: className<**typeName**> objectName; //Be careful while creating an object in main

More

- Multiple heritage: class className: public classMother1, public classMother2 {};
- Namespaces:
  - nameSpace1::className object;
  - nameSpace2::className object;
- Enumerated type: enum enumName{enum1, enum2, enum3…};
- Create a new type: typedef oldTypeLongName newTypeName;
- More libraries:
  - **2D Games**: Allegro, SFML,
  - **3D Games:**
    - API: DirectX, OpenGL → Basic functions
    - 3D Engine: Irrlicht, Ogre3D → Complex functions
  - **GUI**: wxWidgets, .NET
  - **Sound**: FMOD EX
  - **SL Extension**: Boost


Namespace Jeu3D
Namespace Fenetre
Couleur
Couleur
Ces 2 classes portent le même nom mais ça ne pose pas de problème car elles sont dans des namespaces différents.


QWidget
Ui::FenCalculatrice
Héritage multiple
FenCalculatrice