

PROGRAMACIÓN ORIENTADA A OBJETOS USANDO C++

1ª EDICIÓN

4ª SESIÓN

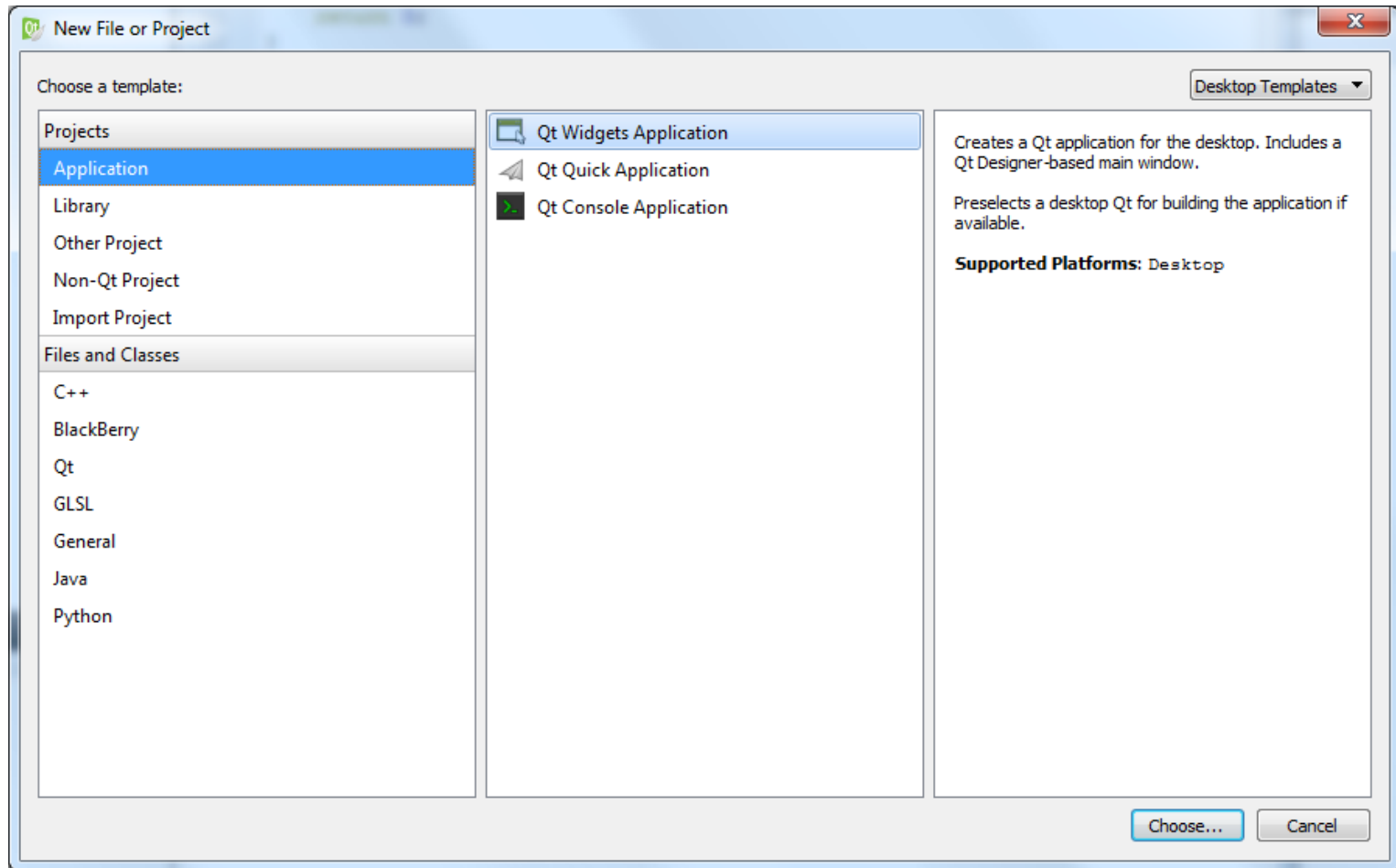
1

Curso 2015-2016

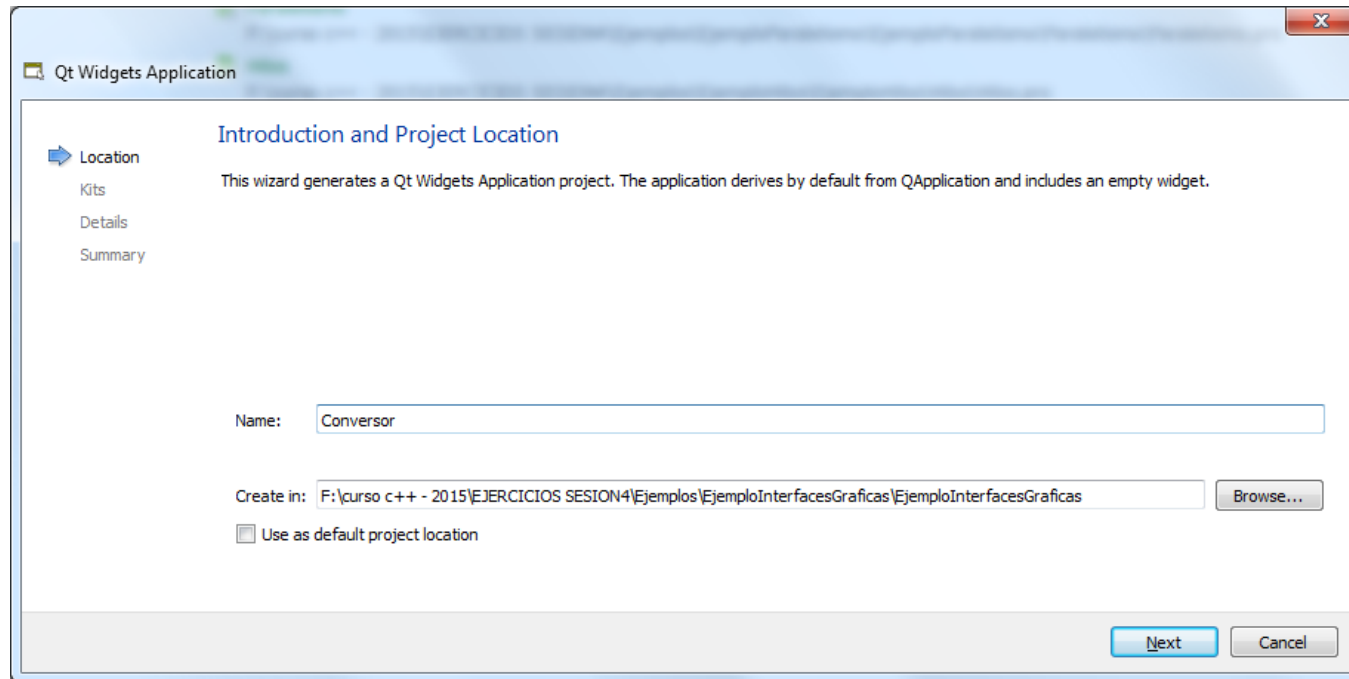
ÍNDICE

1. Interfaces de usuario con Qt
2. Ficheros
3. Plantillas
4. Excepciones
5. Memoria dinámica
6. Preprocesador
7. Espacio de nombres
8. Multihilo
9. Paralelismo
10. Novedades de C++11 y C++14

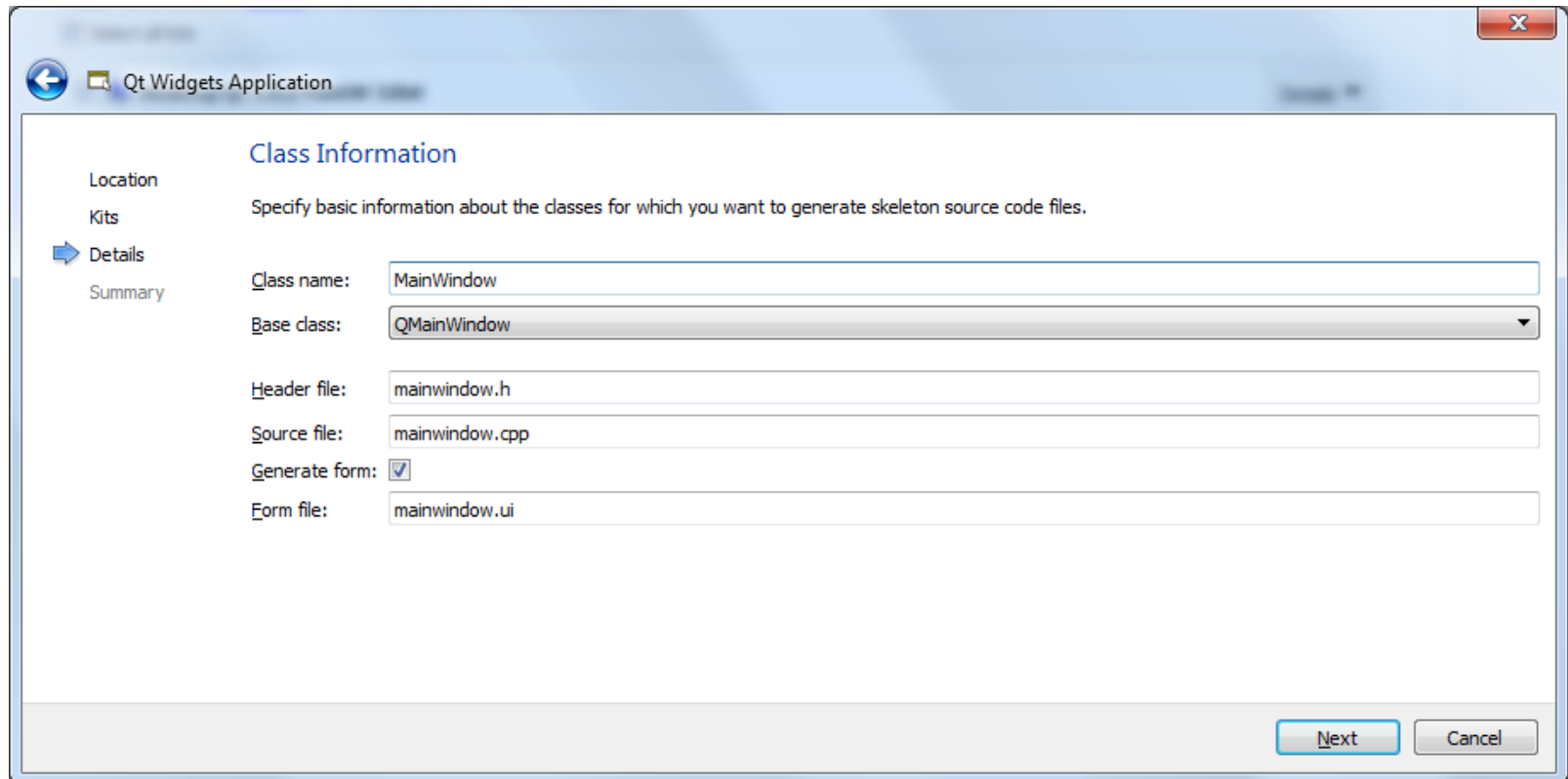
1- INTERFACES GRÁFICAS CON QT



1- INTERFACES GRÁFICAS CON QT



1- INTERFACES GRÁFICAS CON QT



The image shows a Qt Widgets Application dialog box titled "Qt Widgets Application". It has a sidebar on the left with four options: "Location", "Kits", "Details" (which is selected and highlighted with a blue arrow), and "Summary". The main area is titled "Class Information" and contains the instruction "Specify basic information about the classes for which you want to generate skeleton source code files." Below this instruction are several input fields: "Class name:" with the text "MainWindow", "Base class:" with a dropdown menu showing "QMainWindow", "Header file:" with the text "mainwindow.h", "Source file:" with the text "mainwindow.cpp", "Generate form:" with a checked checkbox, and "Form file:" with the text "mainwindow.ui". At the bottom right of the dialog are two buttons: "Next" and "Cancel".

Qt Widgets Application

Location
Kits
Details
Summary

Class Information

Specify basic information about the classes for which you want to generate skeleton source code files.

Class name:

Base class:

Header file:

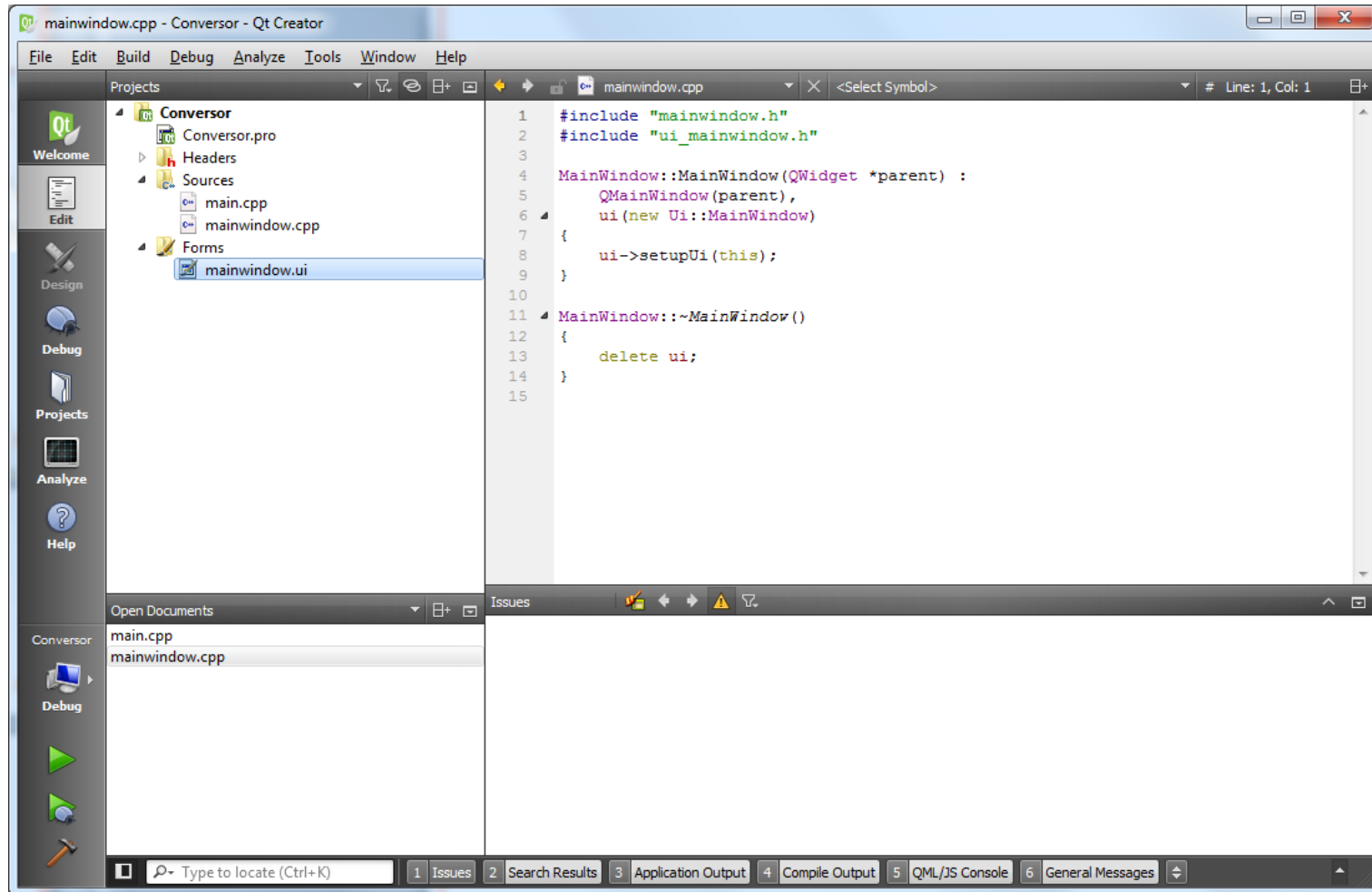
Source file:

Generate form: ☒

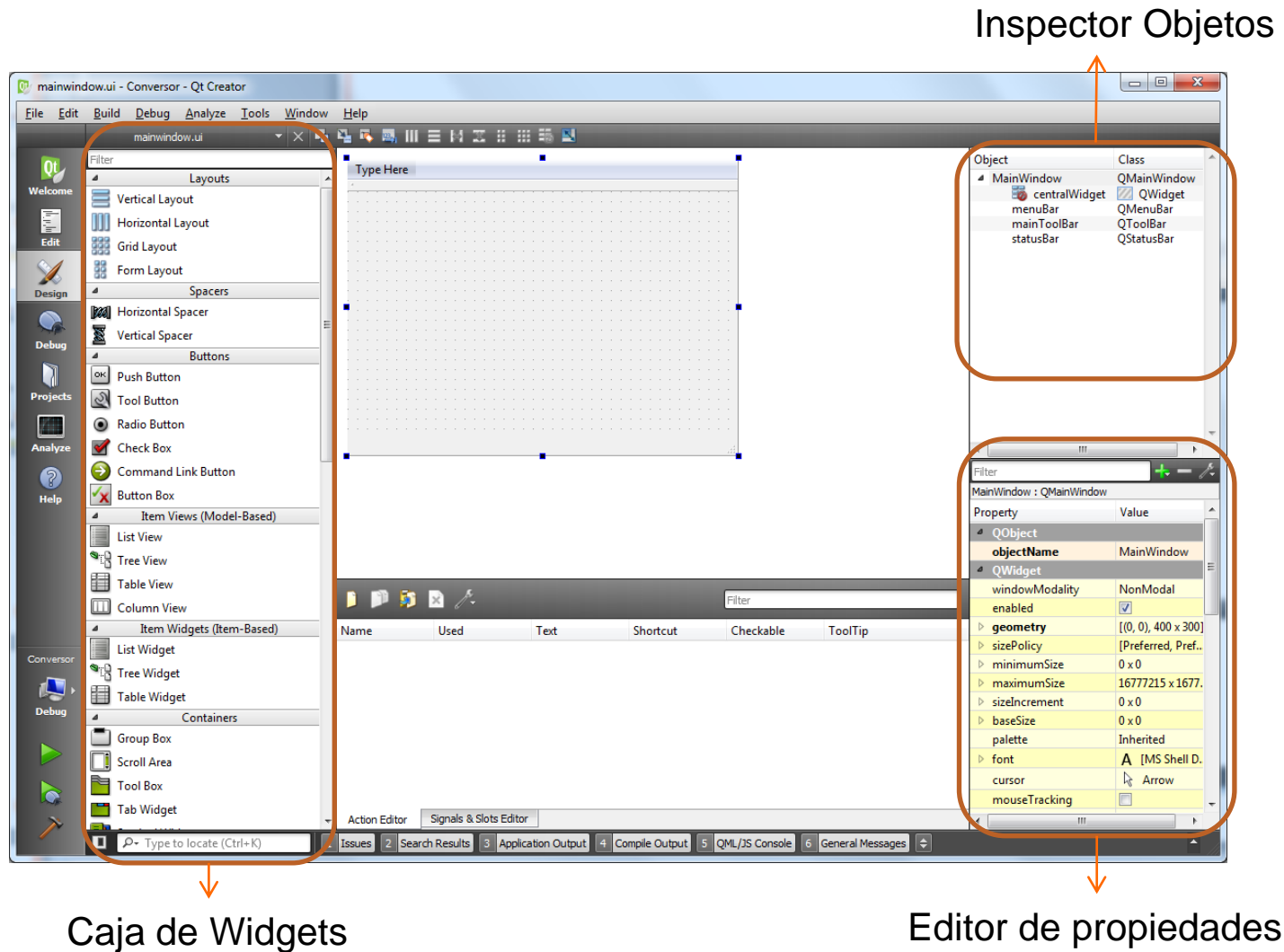
Form file:

Next Cancel

1- INTERFACES GRÁFICAS CON QT



1- INTERFACES GRÁFICAS CON QT



1- INTERFACES GRÁFICAS CON QT

- Queremos realizar un conversor **Decimal – Binario**
- ¿Qué necesitamos? Diseñar nuestra interfaz gráfica....

Diagram illustrating a Qt graphical user interface for a Decimal to Binary converter. The interface includes:

- A text input field for the decimal number.
- Two radio buttons for selecting the conversion direction: **Decimal - Binario** (selected) and **Binario - Decimal** (unselected).
- A second text input field for the resulting binary number.
- A **Convertir** button to perform the conversion.

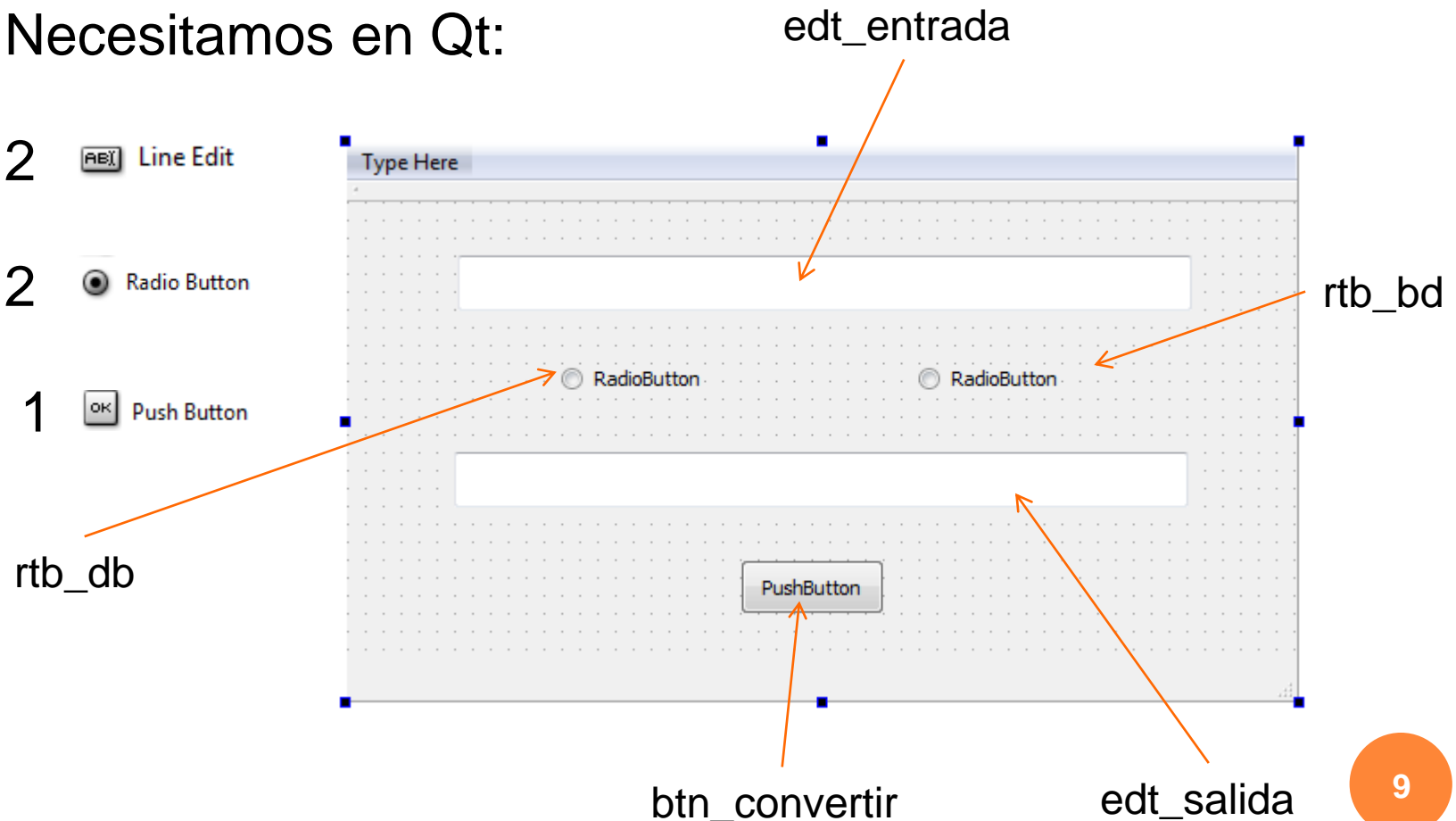
1- INTERFACES GRÁFICAS CON QT

- Necesitamos en Qt:

- 2  Line Edit

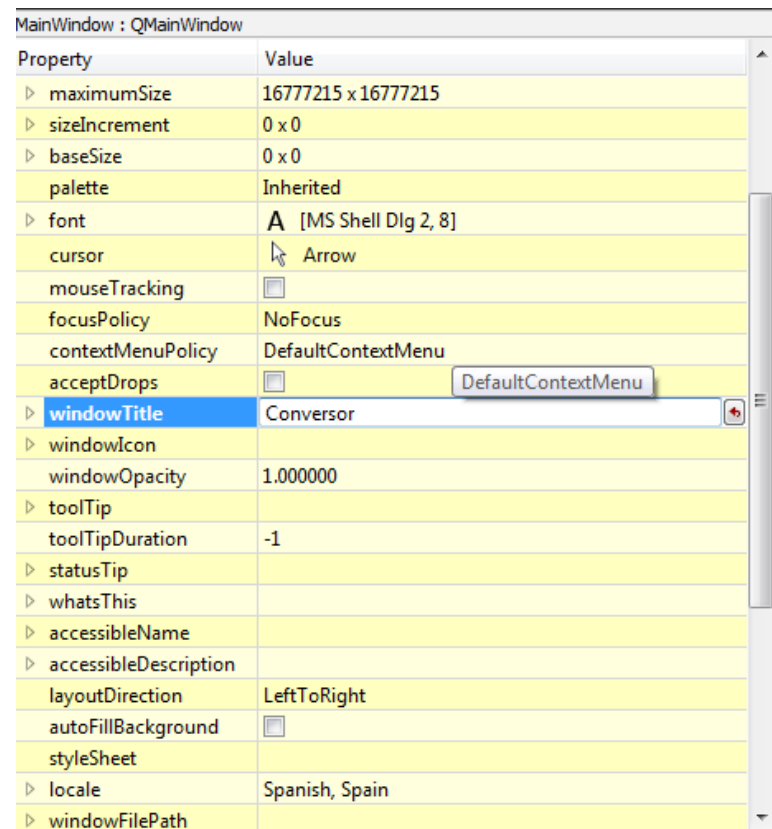
- 2  Radio Button

- 1  Push Button



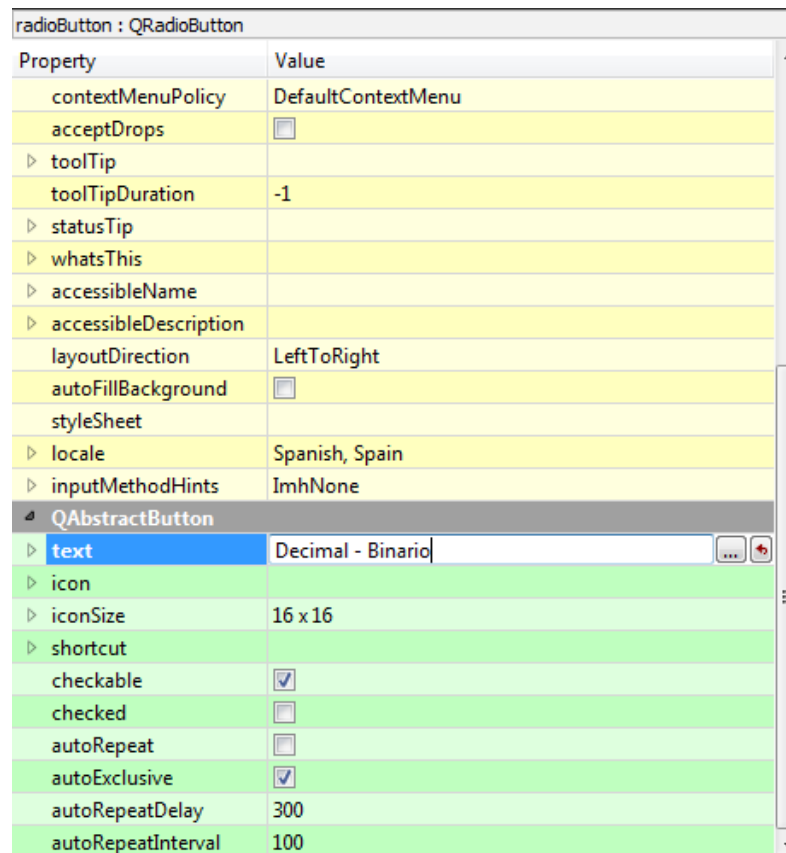
1- INTERFACES GRÁFICAS CON QT

- En el editor de propiedades:
 - Cambiamos el nombre de la ventana de la interfaz por **Conversor**



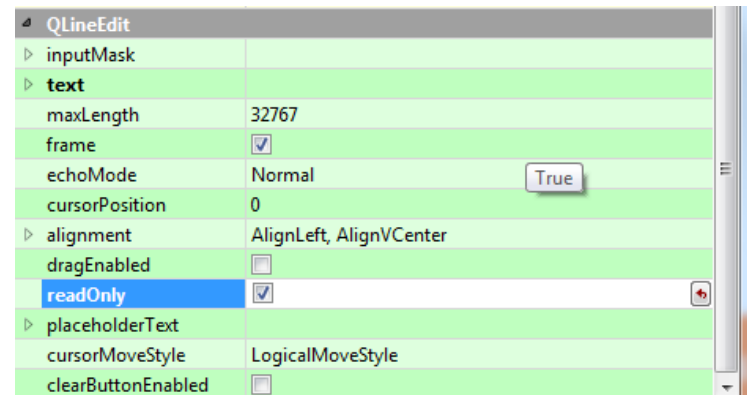
1- INTERFACES GRÁFICAS CON QT

- Cambiamos los textos de los radioButton por **Decimal – Binario y Binario – Decimal**

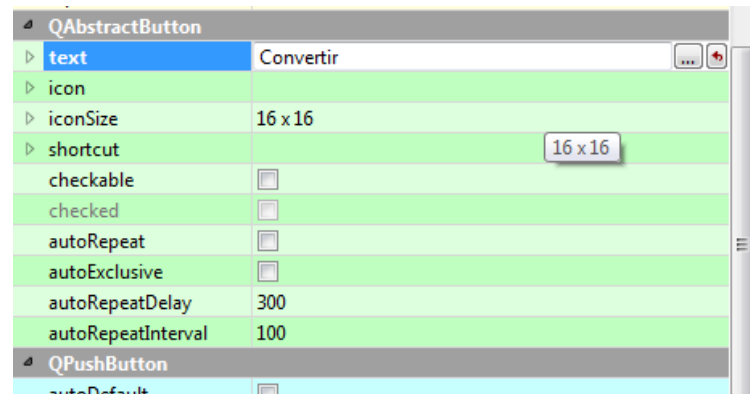


1- INTERFACES GRÁFICAS CON QT

- Cambiamos a **onlyRead** el LineEdit que muestra el resultado de la conversión

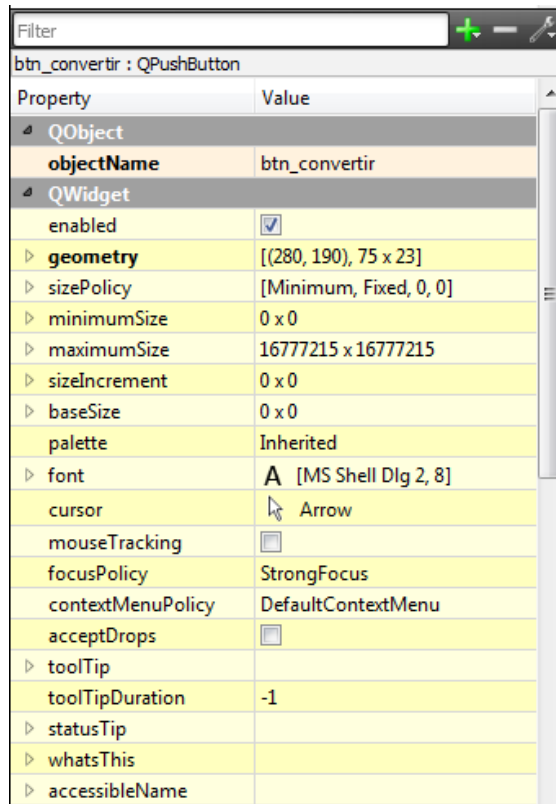


- Cambiamos el texto del botón a **Convertir**



1- INTERFACES GRÁFICAS CON QT

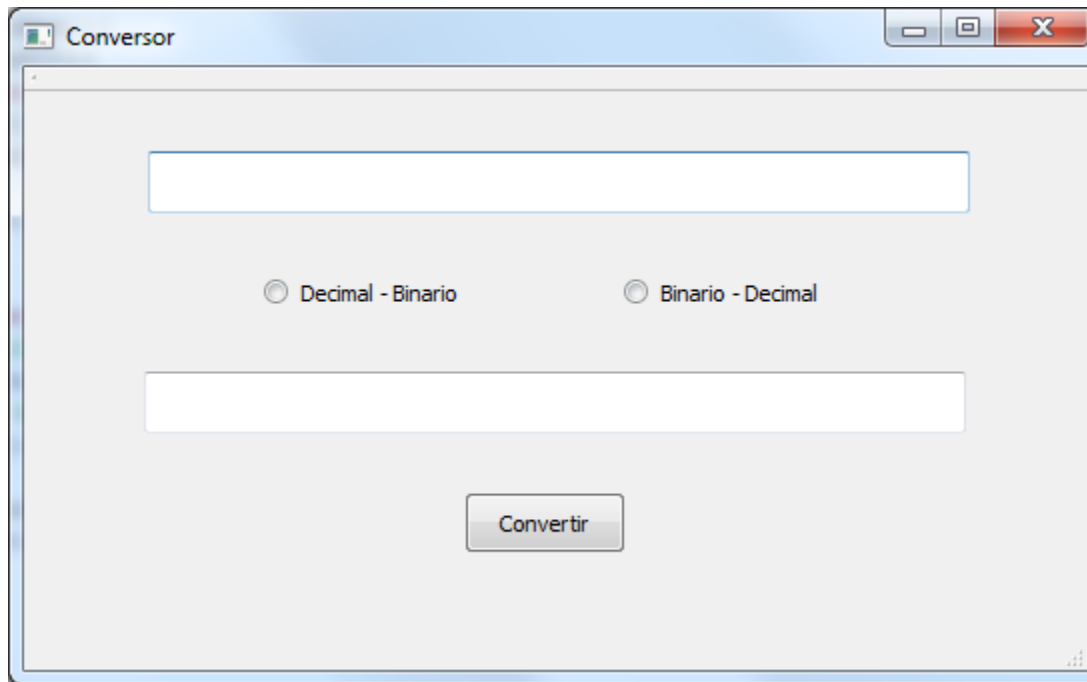
- Cambiamos nombre de las variables de la interfaz para poder distinguirlas a la hora de realizar el programa



Object	Class
MainWindow	QMainWindow
centralWidget	QWidget
btn_convertir	QPushButton
edt_entrada	QLineEdit
edt_salida	QLineEdit
rtb_bd	QRadioButton
rtb_db	QRadioButton
menuBar	QMenuBar
mainToolBar	QToolBar
statusBar	QStatusBar

1- INTERFACES GRÁFICAS CON QT

- Resultado



1- INTERFACES GRÁFICAS CON QT

○ Funciones de conversión Decimal – Binario

```
long int MainWindow::DecimalBinario(string ar){
    unsigned long int n, res=0;
    for(n=0;n<ar.length();n++){
        if(!(ar[n]>='0' && ar[n]<='9')) return res=-1;
    }
    unsigned long int x = atof(ar.c_str());
    unsigned long int y=0,aux=1;
    unsigned long int i[200];
    if (x>0){
        n=x;
        do{
            res=n%2;
            n=n/2;
            i[y]=res; y++;
        }while(n>=2);
        i[y]=n;
    }
    res=0;
    for(int h=0;h<=y;h++){
        res=res+i[h]*aux;
        aux=aux*10;
    }
    return res;
}
```

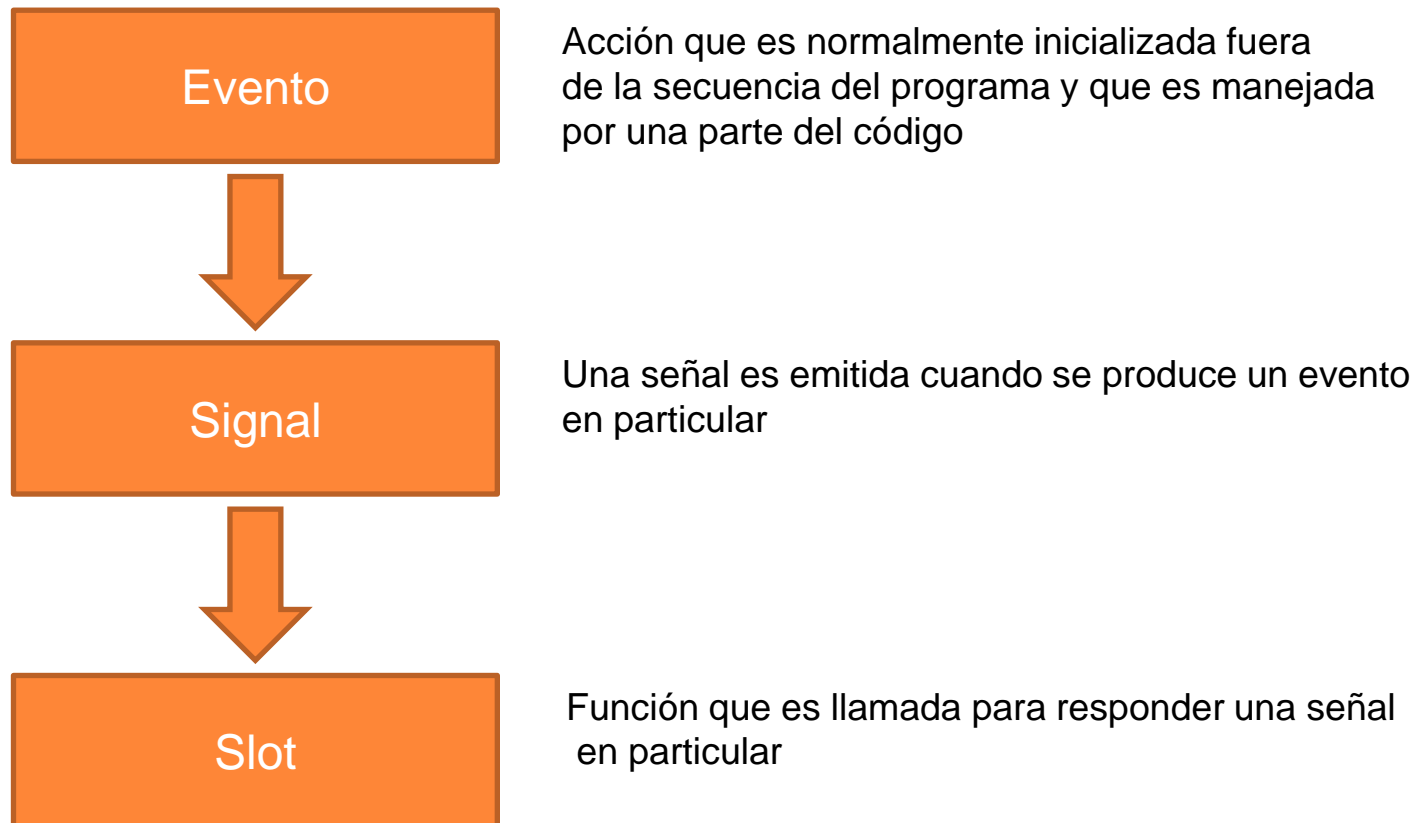
1- INTERFACES GRÁFICAS CON QT

○ Funciones de conversión Binario – Decimal

```
long int MainWindow::BinarioDecimal(string ar){
    long int n, res=0;
    for(n=ar.length()-1;n>=0;n--){
        if(!(ar[n]=='0' || ar[n]=='1')){ return res=-1; }
    }
    unsigned long int p=0,aux;
    for(n=ar.length()-1;n>=0;n--){
        aux=ar[n]-48;
        res=res+aux*pow(2,p); p++;
    }
    return res;
}
```


1- INTERFACES GRÁFICAS CON QT

○ Gestión de eventos con Qt



1- INTERFACES GRÁFICAS CON QT

- Gestión de eventos con Qt

```
connect(emisor, SIGNAL(signal_emitida()), receptor, SLOT(slot_doaction()));
```

- En nuestro caso:

```
connect(ui->convert, SIGNAL(clicked()), this, SLOT(calculate()));
```

```
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new Ui::MainWindow) {  
    ui->setupUi(this);  
    connect(ui->btn_convertir, SIGNAL(clicked()),this,SLOT(calculate()));  
}
```

1- INTERFACES GRÁFICAS CON QT

- A la hora de designar la función en el archivo de cabecera deberemos especificar que es de tipo slot:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <iostream>
#include <math.h>
```

```
using namespace std;
```

```
namespace Ui {
    class MainWindow;
}
```

1- INTERFACES GRÁFICAS CON QT

```
class MainWindow : public QMainWindow {  
    Q_OBJECT  
    public:  
        explicit MainWindow(QWidget *parent = 0);  
        ~MainWindow();  
        long int DecimalBinario(string ar);  
        long int BinarioDecimal(string ar);  
    private:  
        Ui::MainWindow *ui;  
    public slots:  
        void calculate();  
};  
#endif // MAINWINDOW_H
```

1- INTERFACES GRÁFICAS CON QT

```
void MainWindow::calculate(){

    QString numberS=ui->edt_entrada->text();
    QByteArray ba = numberS.toLatin1();
    string numberS2=ba.data();

    if(ui->rtb_db->isChecked()){
        long int numberBinary=DecimalBinario(numberS2);
        ui->edt_salida->setText(QString::number(numberBinary));
    } else{
        long int numberBinary=BinarioDecimal(numberS2);
        ui->edt_salida->setText(QString::number(numberBinary));
    }
}
```

1- INTERFACES GRÁFICAS CON QT

- Otras opciones con eventos → cuando se pulse cualquiera de los Radio Buttons se borre la información tanto de la entrada del numero a convertir como de la salida.

- Conexion:

```
connect(ui->rtb_bd , SIGNAL(clicked()),this,SLOT(deleteInfo()));
```

```
connect(ui->rtb_db , SIGNAL(clicked()),this,SLOT(deleteInfo()));
```

1- INTERFACES GRÁFICAS CON QT

- Función:

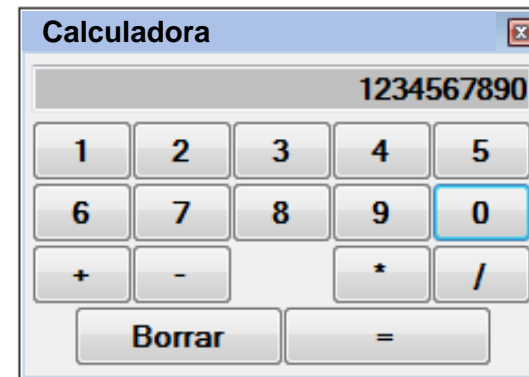
```
void MainWindow::deleteInfo(){  
    ui->edt_entrada->setText("");  
    ui->edt_salida->setText("");  
}
```

- Definición como función SLOT:

```
public slots:  
    void deleteInfo();
```

1- INTERFACES GRÁFICAS CON QT

- Ejercicio: Utilizando las interfaces gráficas con QT crea tu propia calculadora



- Haz las operaciones básicas: suma, resta, multiplicación y división
- Avanzado: raíz cuadrada, cambio de signo, módulo, seno, coseno, tangente,....

2- FICHEROS

- C++ permite el manejo de archivos mediante la librería estándar **fstream**

Tipo de dato	Descripción
ofstream	Tipo de datos que representa el flujo de salida del fichero. Es usado para crear y escribir en los archivos.
ifstream	Tipo de datos que representa el flujo de entrada del fichero. Es usado para leer en los archivos.
fstream	Tipo de datos que representa el flujo de datos en general, es decir, se puede usar tanto para manejar la salida como la entrada de datos (ofstream e ifstream).

2- FICHEROS

○ Abrir un archivo:

```
void open(const char *filename, ios::openmode mode );
```

Mode Flag	Description
ios::app	Modo añadir. La información se concatena al final del archivo.
ios::ate	Abre el archivo y lleva el control de la lectura/escritura al final del archivo.
ios::in	Abre el archivo con opciones de lectura.
ios::out	Abre el archivo con opciones de escritura.
ios::binary	Abre el archivo en modo binario

```
fstream afile;  
afile.open("file.dat", ios::out | ios::in );    // para abrir el archivo con más de una  
                                                // opción se usa el operador OR
```

2- FICHEROS

- **Leer** de un archivo: La lectura en archivos se realiza mediante el operador de extracción (>>), mediante la función **read()** o **getline()**

```
afile >> data; // lee el dato del archivo
```

```
read (char*, size)
```

o

```
getline(infile, data);
```

- **Escribir** en un archivo: La escritura en archivos se realiza utilizando el operador de inserción (<<) o la función **write()**

```
afile << data; // escribe el dato en el archivo
```

o

```
write(char*, size);
```

2- FICHEROS

- **Cerrar** un archivo: Cuando un programa en C++ acaba, éste directamente cierra los archivos y conexiones que haya abierto y libera la memoria del flujo de datos asociado. Sin embargo, es una buena práctica el cerrar los archivos una vez que ya no se quieran volver a utilizar en el programa.

```
afile.close();
```

- **Final de archivo: eof()**. Función cuya salida es un booleano que indica si se ha llegado al final del archivo. Muy útil a la hora de recorrer archivos.

```
while (! afile.eof()) {  
    //Lo que quieras hacer en el archivo  
}
```

2- FICHEROS

- Ejemplo: Crear un fichero llamado ejemplillo.txt que no existía y escribir la típica frase de “Hola mundo”. Dos opciones:

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    fstream myReadFile;
    myReadFile.open("Ejemplillo.txt", ios::out);
    myReadFile.write("\nHola Mundo\n", 12);
    return 0;
}
```

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    fstream myReadFile;
    myReadFile.open("Ejemplillo.txt", ios::out);
    myReadFile << "\nHola Mundo\n";
    return 0;
}
```

2- FICHEROS

- Ejemplo: Leer el contenido del archivo «test.txt»

```
#include <iostream>
#include <fstream>

using namespace std;

int main () {
    fstream myReadFile;
    myReadFile.open("test.txt", ios::in);
    string cadena;

    while(!myReadFile.eof()) {
        myReadFile >> cadena;
        cout << cadena << endl;
    }
    return 0;
}
```

2- FICHEROS

- Ejercicio: Usa el archivo «*EjercicioFicheros1.txt*» para:
 - ❑ Lee su contenido e imprímelo por pantalla (usa >> o getline() para leer el archivo y la función eof() para el control del final de archivo)
 - ❑ Cuenta el número de palabras que contiene
 - ❑ Cambia todas las apariciones de la letra “i” por “u” en el archivo
 - ❑ Avanzado: Cambia al autor de la cita por tu nombre y apellido

3- PLANTILLAS

- Nos permite escribir código genérico que puede ser usado con varios tipos de datos. Sin templates, se tendrían que reescribir muchas funciones y clases.
- Imagina que tenemos estas dos funciones que retornan en ambos casos el máximo de dos valores

```
int maximo(int x, int y) {  
    return (x < y) ? y : x;  
}
```

```
float maximo(float x, float y) {  
    return (x < y) ? y : x;  
}
```


3- PLANTILLAS

- Podríamos crear una función genérica que retornara el máximo de dos valores pero independientemente del tipo de datos de entrada usando una template.

```
#include <iostream>

using namespace std;

template <typename T> T maximo(T x, T y) {
    return (x < y) ? y : x;
}

int main() {
    int a=4, b=5;
    float c=4.5, d=4.6;
    cout << maximo(a,b) << endl;
    cout << maximo(c,d) << endl;
    return 0;
}
```

3- PLANTILLAS

- Además, las plantillas pueden ser también aplicadas a las clases

❑ Sin template

```
class calc {  
public:  
    int multiply(int x, int y);  
    int add(int x, int y);  
};
```

```
int calc::multiply(int x, int y) {  
    return x*y;  
}  
  
int calc::add(int x, int y) {  
    return x+y;  
}
```

❑ Con template

```
template <class A_Type> class calc {  
public:  
    A_Type multiply(A_Type x, A_Type y);  
    A_Type add(A_Type x, A_Type y);  
};
```

```
template <class A_Type> A_Type calc<A_Type>::multiply(A_Type x,A_Type y) {  
    return x*y;  
}  
  
template <class A_Type> A_Type calc<A_Type>::add(A_Type x, A_Type y) {  
    return x+y;  
}
```

3- PLANTILLAS

- Ejemplo con clases. Ejemplo de la clase Punto como template
- Punto.h

```
#ifndef PUNTO_H
#define PUNTO_H

using namespace std;

template <class T> class Punto {
    private:
        T _coorx; T _coory;
    public:
        Punto(T x,T y);
        ~Punto();
        void setCoorX(T x);
        void setCoorY(T y);
        T getCoorX();
        T getCoorY();
};

#endif // PUNTO_H
```

3- PLANTILLAS

○ Punto.cpp

```
#include "punto.h"
```

```
template <class T> Punto<T>::Punto(T x, T y){  
    _coorx=x;  
    _coory=y;  
}
```

```
template <class T> Punto<T>::~~Punto(){}
```

```
template <class T> void Punto<T>::setCoorX(T x){  
    _coorx=x;  
}
```

```
template <class T> void Punto<T>::setCoorY(T y){  
    _coory=y;  
}
```

```
template <class T> T Punto<T>::getCoorX(){  
    return _coorx;  
}
```

```
template <class T> T Punto<T>::getCoorY(){  
    return _coory;  
}
```

3- PLANTILLAS

○ Main.cpp

```
#include <iostream>
```

```
#include "Punto.h"
```

```
#include "Punto.cpp"
```

```
using namespace std;
```

```
int main() {
```

```
    Punto <float> obj1(1.2,2.3) ;
```

```
    cout<<"coordenada x "<<obj1.getCoorX()<<" y coordenada y "<<obj1.getCoorY()<<endl;
```

```
    Punto <int> obj2 (3,7) ;
```

```
    cout<<"coordenada x "<<obj2.getCoorX()<<" y coordenada y "<<obj2.getCoorY()<<endl;
```

```
    Punto <long int> obj3 (1254844444, 1238999999) ;
```

```
    cout<<"coordenada x "<<obj3.getCoorX()<<" y coordenada y "<<obj3.getCoorY()<<endl;
```

```
}
```

3- PLANTILLAS

- Ejercicio. Realiza funciones genéricas para la suma, resta, multiplicación, división y mínimo de dos números.

4- EXCEPCIONES

- Una excepción es un problema que aparece durante la ejecución del programa. C++ permite definir las excepciones para saber la causa del problema (ejemplo: división por cero). Las excepciones proveen una forma de transferir el control de una parte del código a otra.
- Lanzar excepción: **throw**

```
#include <iostream>
using namespace std;

double division(int a, int b) {
    if( b == 0 ) {
        throw "Division by zero condition!";
    }
    return (a/b);
}

int main () {
    int x = 50;
    int y = 0;
    double z = 0;
    try {
        z = division(x, y);
        cout << "llega aqui?"<< endl;
    }catch (const char* msg) {
        cerr << msg << endl;
    }
    cout << "y aqui?"<< endl;
    return 0;
}
```

4- EXCEPCIONES

- Capturar excepción: **try, catch**. Las excepciones a capturar pueden ser especificadas mediante la declaración de excepción que aparece después del **catch**

```
try {  
    // código protegido  
}catch( ExceptionName e ) {  
    // código para manejar la excepción ExceptionName  
}
```

```
try {  
    // código protegido  
}catch(...) {  
    // código para manejar cualquier excepción  
}
```


4- EXCEPCIONES

Exception	Description
std::exception	Clase padre de las excepciones de C++.
	std::bad_alloc, std::bad_cast, std::bad_exception, std::bad_typeid
std::logic_error	Excepción que teóricamente debería ser detectada mirando el código del programa.
	std::domain_error, std::invalid_argument, std::length_error, std::out_of_range
std::runtime_error	Excepción que teóricamente no es detectada mirando el código del programa.
	std::overflow_error, std::range_error, std::underflow_error

4- EXCEPCIONES

○ Excepción:

```
#include <iostream>
using namespace std;

int main() {
    int *x;
    int y = 100000000;
    try {
        x = new int[y];
    }
    catch(std::bad_alloc&) {
        cout << "Memoria insuficiente" << endl;
    }
    return 0;
}
```

4- EXCEPCIONES

○ Excepción:

```
#include <iostream>

#include <string>

using namespace std;

int main() {
    string s = "Hola";
    try {
        cout << s.at(100) << endl;
    }
    catch(exception& e) {
        cout << e.what() << endl;
    }
    catch(...){
    }
    return 0;
}
```

5- MEMORIA DINÁMICA

- A veces se conoce la cantidad exacta, el tipo y duración de la vida de los objetos en un programa, pero no siempre es así.
- The stack (pila): Todas las variables declaradas dentro del programa se llevarán a la memoria de la pila.
- The heap (montículo): Esta es la memoria no utilizada por el programa y se puede utilizar para asignar memoria dinámicamente cuando se ejecuta el programa. Cuidado porque al igual que se reserva memoria, hay que liberarla.
- C: asignar → `malloc()`, `calloc()`, `realloc()` ; liberar → `free()`
- **C++: asignar → `new` ; liberar → `delete()`**

5- MEMORIA DINÁMICA

- En el caso de los punteros a variables, el uso de `new` y `delete` sería:

```
#include <iostream>
```

```
using namespace std;
```

```
int main () {
```

```
    double* pvalue = NULL; // puntero inicializado a nulo
```

```
    pvalue = new double; // Solicita memoria para la variable
```

```
    *pvalue = 29494.99;    // Almacena el valor en la dirección asignada
```

```
    cout << "Value of pvalue : " << *pvalue << endl;
```

```
    cout << "Value of pvalue : " << pvalue << endl;
```

```
    cout << "Value of pvalue : " << &pvalue << endl;
```

```
    delete pvalue; // libera la memoria
```

```
    return 0;
```

```
}
```

5- MEMORIA DINÁMICA

- En el caso de los punteros a objetos, el uso de new y delete sería:

```
#include <iostream>
#include <vector>

using namespace std;

class MusicBox {
private:
    vector<string> _canciones;
public:
    MusicBox() { cout << "Constructor called!" <<endl; }
    ~MusicBox() { cout << "Destructor called!" <<endl; }
    void setCancion(string cancion){ _canciones.push_back(cancion); }
    string getCancion(int position){ return _canciones[position]; }
};

int main( ) {
    MusicBox* myBox = new MusicBox();
    myBox->setCancion("Viva la Vida - Cold Play");
    myBox->setCancion("Uprising - Muse");
    myBox->setCancion("California Dreaming - Sia");
    string cancionSeleccionada=myBox->getCancion(2);
    cout<<cancionSeleccionada<<endl;
    delete myBox; // Delete array
    return 0;
}
```

5- MEMORIA DINÁMICA

- Ejercicio: Vuelve a escribir el método main() con los objetos de la clase Coche reservando memoria dinámicamente.

```
#ifndef COCHE_H
#define COCHE_H

#include <iostream>

using namespace std;

class Coche
{
public:
    Coche( void );
    ~Coche( void );
    int _peso;
    int _n_puertas;
    int _potencia;
    double _precio;
    string _marca;
    string _modelo;
    double getPotenciaPeso( void );
    int getPotencia( void ) { return _potencia; }
    void setPotencia( int potencia ) { _potencia = potencia; }
};

#endif // COCHE_H
```

5- MEMORIA DINÁMICA

```
#include "coche.h"

// Constructor vacío
Coche::Coche( void )
{
    cout << "Estamos creando un coche..." << endl;
    _peso = 0; _n_puertas = 0; _potencia = 0;
    _precio = 0; _marca = "Desconocida";
    _modelo = "Desconocido";
}

double Coche::getPotenciaPeso( void ){
    return _peso / _potencia;
}

Coche::~Coche( void ){
    cout << "Estamos eliminando un coche..." << endl;
}
```


5- MEMORIA DINÁMICA

```
#include <iostream>
#include "coche.h"
using namespace std;

int main(){

    int n_coches = 0;

    Coche coche1; n_coches++;
    Coche coche2; n_coches++;
    Coche coche3; n_coches++;

    coche1._marca = "Seat";
    coche1._modelo = "Ibiza";
    coche1._n_puertas = 5;
    coche1._peso = 1200;
    coche1._potencia = 95;
    coche1._precio = 12000;

    cout << "La marca es " << coche1._marca << endl;
    cout << "El modelo es " << coche1._modelo << endl;
    cout << "Precio " << coche1._precio << " euros" << endl;
    cout << "PotenciaPeso " << coche1.getPotenciaPeso() << endl;

    cout << "Coches= " << n_coches << endl;
    return 0;
}
```

6- PREPROCESADOR

- Las sentencias del preprocesador son directivas que dan instrucciones al compilador para procesar la información antes de que la compilación comience.
- Las directivas del preprocesador empiezan con el símbolo #
- Existen diferentes tipos de directivas en C++ entre ellas, las mas conocidas:
 - #include, #define, #undef, #if, #else, #warning, ...

6- PREPROCESADOR

- La directiva **#include**: la directiva que nos permite incluir ficheros externos dentro de nuestro fichero de código fuente. Estos ficheros son conocidos como ficheros incluidos, ficheros de cabecera o "headers".

6- PREPROCESADOR

- Diferencia entre: **#include < >** e **#include “ “**
- La versión con los paréntesis angulares busca los ficheros en todos los directorios que se han especificado en la llamada al compilador. Cuando se incluye un fichero entre comillas, entonces el compilador busca este fichero primero en el mismo directorio que el fichero actualmente compilado y después en los demás directorios.
- Más significativo es el comportamiento ante ficheros con el mismo nombre en distintos directorios. En este caso la versión con comillas da preferencia sobre el fichero en el mismo directorio y esto suele ser el mejor acertado.

6- PREPROCESADOR

- La directiva **#define**:
 - #define nombre_macro valor

```
#include <iostream>
```

```
using namespace std;
```

```
#define PI 3.14159
```

```
int main () {
```

```
    cout << "Value of PI :" << PI << endl;
```

```
    return 0;
```

```
}
```

6- PREPROCESADOR

○ La directiva **#define** como función:

```
#include <iostream>
```

```
using namespace std;
```

```
#define MIN(a,b) (((a)<(b)) ? a : b)
```

```
int main () {  
    int i = 100, j = 30;  
    cout <<"The minimum is " << MIN(i, j) << endl;  
    return 0;  
}
```

IMPORTANTE !

Correcto:

```
#define MIN(a,b) (((a)<(b)) ? a : b)
```

Incorrecto:

```
#define MIN (a,b) (((a)<(b)) ? a : b)
```

↑
espacio !!

6- PREPROCESADOR

- Las directivas condicionales: **#if**, **#elif**, **#else**, **#endif**, **#ifdef** e **#ifndef**

```
#include <iostream>

using namespace std;

#define PI 3.14159

int main () {
    #ifdef PI
        cout << "Value of PI :" << PI << endl;
    #else
        cout << "PI no esta definido" << endl;
    #endif
    return 0;
}
```

6- PREPROCESADOR

o La directiva **#undef**:

```
#include <iostream>
using namespace std;
#define PI 3.14159
int main () {
    #ifdef PI
        cout << "Value of PI :" << PI << endl;
        #undef PI
    #endif

    double PI=3.14158;
    cout << "Value of PI :" << PI << endl;
    return 0;
}
```


6- PREPROCESADOR

- Los operadores # y ##
- El operador # reemplaza el token de texto en un string entre comillas.

```
#include <iostream>
```

```
using namespace std;
```

```
#define MKSTR( x ) #x
```

```
int main () {  
    cout << MKSTR(HELLO C++) << endl;  
    return 0;  
}
```

¡ Mira a ver cual es el resultado !

6- PREPROCESADOR

- El operador `##` es usado para concatenar dos tokens.

```
#include <iostream>

using namespace std;

#define CONCATENATE(x, y) x ## y

int main() {
    int international=224;
    cout << CONCATENATE(inter, national);
    return 0;
}
```

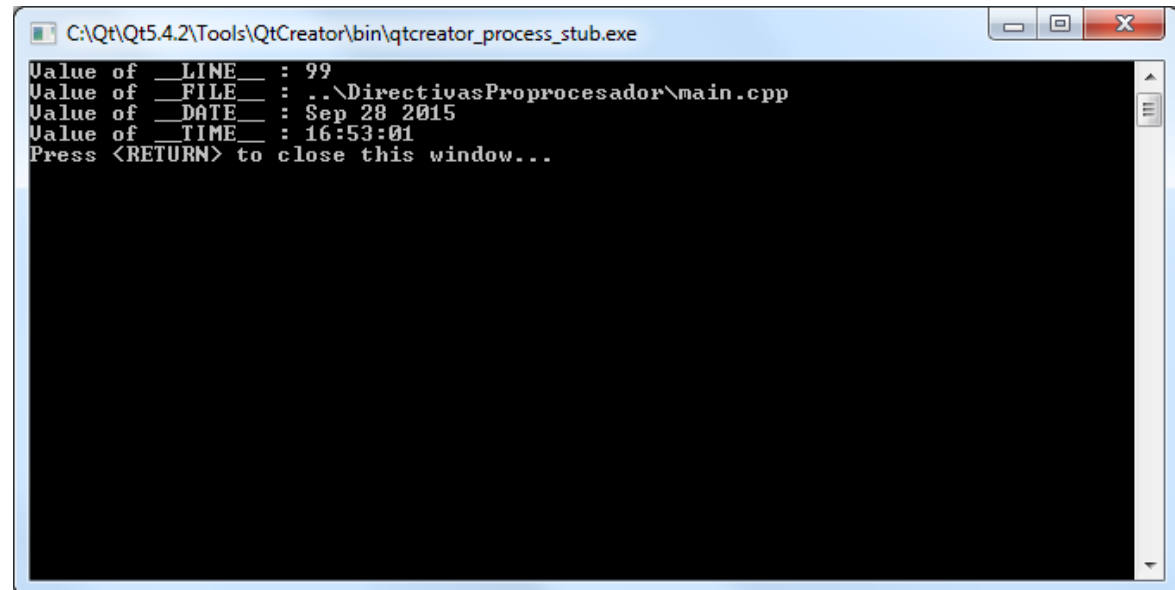
6- PREPROCESADOR

- Macros predefinidas:
- `__LINE__` : línea de compilación
- `__FILE__` : archivo actual que está siendo compilado
- `__DATE__` : Fecha actual
- `__TIME__` : Hora, minuto y segundo en el cual el programa fue compilado

6- PREPROCESADOR

○ Macros predefinidas

```
92
93
94 #include <iostream>
95 using namespace std;
96
97 int main ()
98 {
99     cout << "Value of __LINE__ : " << __LINE__ << endl;
100     cout << "Value of __FILE__ : " << __FILE__ << endl;
101     cout << "Value of __DATE__ : " << __DATE__ << endl;
102     cout << "Value of __TIME__ : " << __TIME__ << endl;
103
104     return 0;
105 }
106
107
108
109
110
```



The screenshot shows a console window titled "C:\Qt\Qt5.4.2\Tools\QtCreator\bin\qtcreator_process_stub.exe". The output text is as follows:

```
Value of __LINE__ : 99
Value of __FILE__ : ..\DirectivasPreprocesador\main.cpp
Value of __DATE__ : Sep 28 2015
Value of __TIME__ : 16:53:01
Press <RETURN> to close this window...
```

7- ESPACIOS DE NOMBRES

- Imagina que generando tu código creas una función que posee el mismo nombre que otra de una librería que has incluido para tu código. Cuando quieras hacer una llamada a dicha función el compilador no sabrá que versión de la función utilizar. La forma de solucionar dicho problema es mediante el uso del espacio de nombres.

- Definir un espacio de nombres:

```
namespace nombre_espacio_de_nombres {  
    // código con las funciones a definir  
}
```

- Llamar a la función definida en un espacio de nombres:

```
name::code; // code podría ser una variable o una función
```

7- ESPACIOS DE NOMBRES

- Ejemplo:

```
#include <iostream>
#include <math.h>

using namespace std;

namespace redondeo_alza{
    int round(double number){
        if((double)(number/(int)number)>1) number++;
        cout << "Estoy usando mi propia funcion" <<endl;
        return number;
    }
}

int main() {
    int a=round(2.3);
    cout<<"Redondeo con funcion de math.h: "<<a<<endl;
    a=redondeo_alza::round(2.3);
    cout<<"Redondeo con mi funcion: "<<a<<endl;
    return 0;
}
```

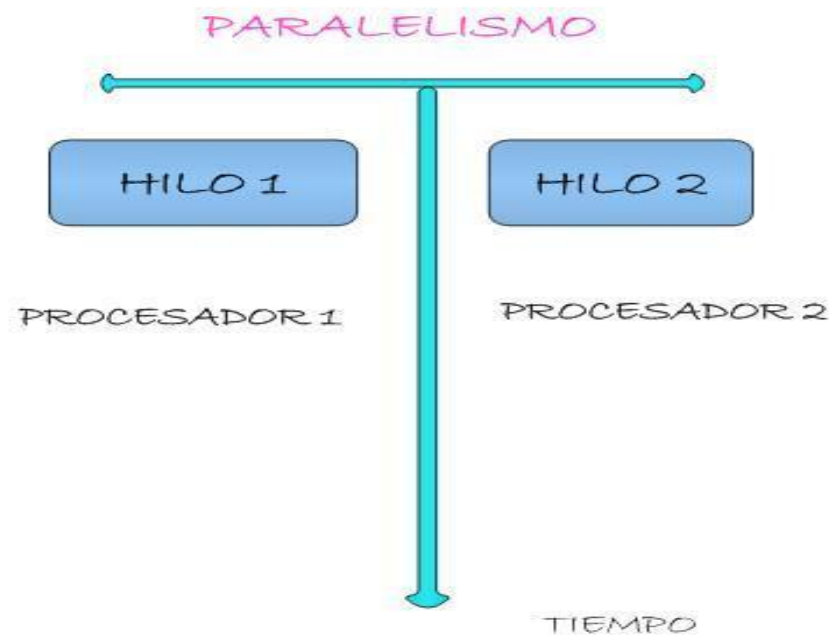
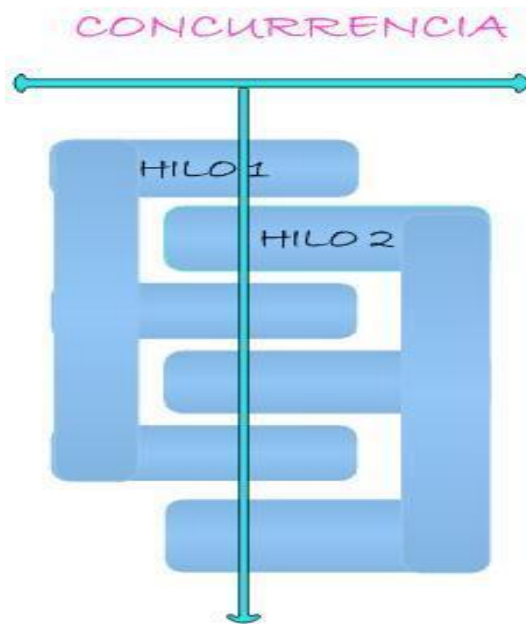
8- HILOS EN C++

- La multitarea es la propiedad que tienen los ordenadores de poder ejecutar dos o más programas a la vez.
- Existen dos tipos de multitarea: la basada en procesos y la basada en hilos. La basada en procesos maneja la ejecución de varios programas a la vez y la basada en hilos permite la ejecución a la vez de varios bloques de código de un mismo programa.

8- HILOS EN C++

- Una de las características más importantes de los hilos es que permiten la ejecución concurrente de instrucciones asociadas a diferentes funciones dentro de un mismo proceso
- Cada hilo es el encargado de realizar una tarea diferente.
- Crear hilos en C++, necesitamos añadir: `#include <pthread.h>`

8- HILOS EN C++



8- HILOS EN C++

- Los hilos de un mismo proceso comparten los siguientes elementos:
 - - Código
 - - Variables de memoria globales
 - - Archivos o dispositivos (entrada y salida estándar) que tuviera abiertos el hilo padre
- Por otro lado, no comparten los siguientes elementos:
 - - Contador de programa
 - - Registros del CPU
 - - Pila
 - - Estado del hilo

8- HILOS EN C++

- Los estados de un hilo son los siguientes:
 - 1. Listo. El hilo ya puede ser ejecutado, pero está esperando a un procesador
 - 2. Ejecutándose/Corriendo. El hilo se está ejecutando; en sistemas multiprocesador pueden estar ejecutándose varios de forma simultánea.
 - 3. Bloqueado. El hilo no puede ejecutarse porque está esperando algo, por ejemplo variables de condición o una operación de E/S para completarse.
 - 4. Terminado. El hilo termina cuando se llamó a la función `pthread_exit` o fue cancelado.

8- HILOS EN C++

- Funciones POSIX de gestión básica de threads

Función	Descripción
<code>pthread_create</code>	Crea un thread para ejecutar una función determinada
<code>pthread_exit</code>	Causa la terminación del thread que lo invoca
<code>pthread_attr_init</code>	Inicializa los atributos del thread a su valor por defecto
<code>pthread_join</code>	Hace que el thread que la invoca espere a que termine un thread determinado
<code>pthread_self</code>	devuelve la identidad del thread que lo invoca
<code>pthread_cancel</code>	Solicita la terminación de otro thread

8- HILOS EN C++

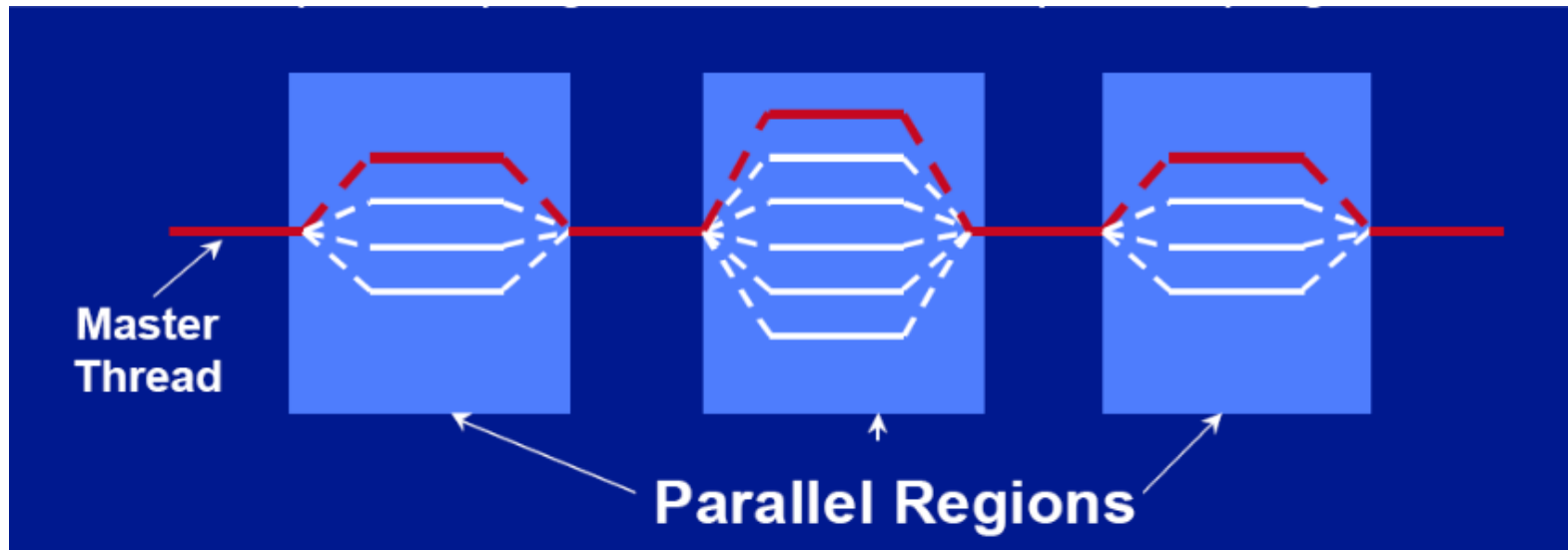
```
#include <iostream>
#include <cstdlib>
#include <pthread.h>
using namespace std;

int lista_ids[10];

void * foo(void * var) { // Función que ejecutará el thread
    int my_id = (int)var;
    printf("Soy el hilo %i!\n", my_id);
    double counter;
    for(int i=0; i < 1000000000; i++)
        counter+=i;
    return NULL;
}

int main() {
    int i;
    pthread_t mythread[4];
    for(i=0; i<4; i++) {
        lista_ids[i] = i+1;
        printf("Creando thread %i!\n", i+1);
        pthread_create(&(mythread[i]), NULL, foo, (void *) i);
        pthread_join(mythread[i], NULL); //Se espera a que se acaben los hilos
    }
    pthread_exit(NULL);
    return 0;
}
```

9- PARALELISMO CON OMP



9- PARALELISMO CON OMP

- La mayoría de las construcciones en OpenMP son directivas de compilación o pragmas. Se deben añadir dentro de nuestro «proyecto.pro».

```
QMAKE_CXXFLAGS+= -fopenmp
```

```
QMAKE_LFLAGS += -fopenmp
```

- En C y C++, los pragmas tienen la forma:
 - **#pragma omp construct [clause [clause]...]**

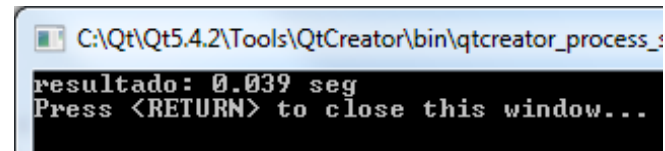
9- PARALELISMO CON OMP

```
#include <iostream>
#include <ctime>
#include <vector>
#include <omp.h>
```

```
using namespace std;
```

```
int main() {
    vector<int> a,b,c;
    a.resize(10000000);
    for(int i=0; i<10000000; i++){
        b.push_back(10);
        c.push_back(5);
    }
    clock_t c_start= clock();
    #pragma omp parallel for
    for (int i = 0; i< 10000000; i++){
        a[i] = b[i] + c[i];
    }
    clock_t c_end= clock();
    cout<<"resultado: "<<(((float)c_end-c_start)/(CLOCKS_PER_SEC))<<" seg"<<endl;
}
```

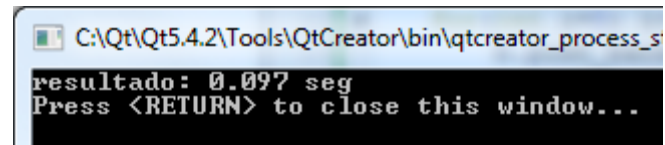
Con paralelización:



C:\Qt\Qt5.4.2\Tools\QtCreator\bin\qtcreator_process_s
resultado: 0.039 seg
Press <RETURN> to close this window...

Sin paralelización:

// #pragma omp parallel for



C:\Qt\Qt5.4.2\Tools\QtCreator\bin\qtcreator_process_s
resultado: 0.097 seg
Press <RETURN> to close this window...

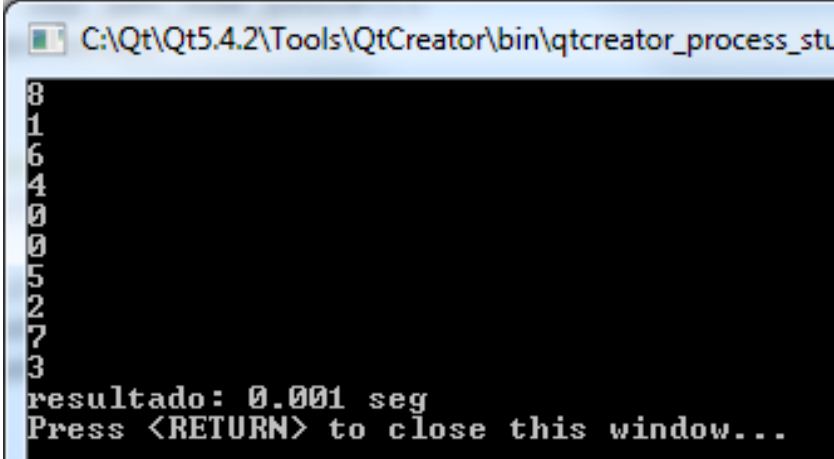
9- PARALELISMO CON OMP

- `int omp_get_num_procs(void);`
 - Devuelve nº de procesadores físicos disponibles para el programa paralelo
- `int omp_get_thread_num(void);`
 - Devuelve número de hilos:0,...,num_hilos-1

9- PARALELISMO CON OMP

```
#include <iostream>
#include <ctime>
#include <vector>
#include <omp.h>
using namespace std;
int main() {

    int num= omp_get_num_procs(); cout<<num<<endl;
    vector<int> a,b,c; a.resize(9);
    for(int i=0; i<9; i++){
        b.push_back(10);
        c.push_back(5);
    }
    clock_t c_start= clock();
    #pragma omp parallel for
    for (int i = 0; i < 9; i++){
        a[i] = b[i] + c[i];
        cout<<omp_get_thread_num()<<endl;
    }
    clock_t c_end= clock();
    cout<<"resultado: "<<(((float)c_end-c_start) /(CLOCKS_PER_SEC))<<" seg"<<endl;
}
```



C:\Qt\Qt5.4.2\Tools\QtCreator\bin\qtcreator_process_stu

```
8
1
6
4
0
0
5
2
7
3
resultado: 0.001 seg
Press <RETURN> to close this window...
```

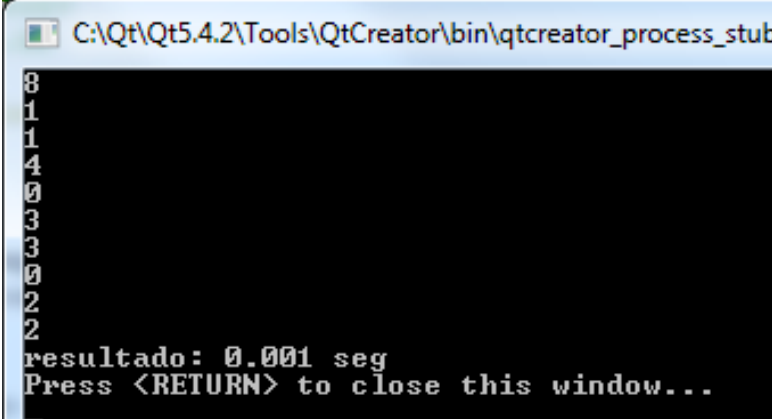
9- PARALELISMO CON OMP

- Variables de entorno OMP_NUM_THREADS
 - Número de hilo por defecto que se crearan en los bucles.
- Void omp_set_num_threads(int num_hilos);
 - Fija el número de hilos en secciones paralelas

9- PARALELISMO CON OMP

```
#include <iostream>
#include <ctime>
#include <vector>
#include <omp.h>
using namespace std;
int main() {

    int num= omp_get_num_procs(); cout<<num<<endl;
    vector<int> a,b,c; a.resize(9);
    for(int i=0; i<9; i++){
        b.push_back(10);
        c.push_back(5);
    }
    clock_t c_start= clock();
    omp_set_num_threads(5);
    #pragma omp parallel for
    for (int i = 0; i < 9; i++){
        a[i] = b[i] + c[i];
        cout<<omp_get_thread_num()<<endl;
    }
    clock_t c_end= clock();
    cout<<"resultado: "<<(((float)c_end-c_start)/(CLOCKS_PER_SEC))<<" seg"<<endl;
}
```



```
C:\Qt\Qt5.4.2\Tools\QtCreator\bin\qtcreator_process_stub
8
1
1
4
0
3
3
0
2
2
resultado: 0.001 seg
Press <RETURN> to close this window...
```

10- NOVEDADES DE C++11

- Para usar C++11 en Qt deberemos añadir en el archivo .pro:

- **QMAKE_CXXFLAGS += -std=c++11**

ó

- **CONFIG += c++11**

10- NOVEDADES DE C++11

- Deducción de tipo: **auto** → No declarar de que tipo son las variables

```
int y=5;  
Cout << y << endl;
```

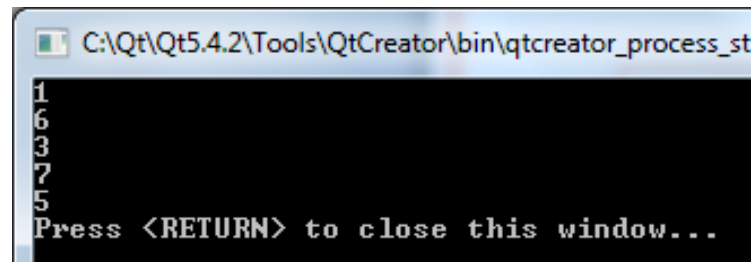
```
auto y=5;  
Cout << y << endl;
```

```
vector<int> v(5,10);  
for (vector<int>::iterator it = v.begin(); it != v.end(); ++it) {  
    cout<<"Elememto "<< *it <<endl;  
}
```

```
vector<int> v(5,10);  
for (auto it = v.begin(); it != v.end(); ++it) {  
    cout<<"Elememto "<< *it <<endl;  
}
```

- Bucle **foreach**

```
int numeros[] = {1,6,3,7,5};  
for (auto valor : numeros){  
    cout << valor <<endl;  
}
```



```
C:\Qt\Qt5.4.2\Tools\QtCreator\bin\qtcreator_process_st  
1  
6  
3  
7  
5  
Press <RETURN> to close this window...
```

10- NOVEDADES DE C++11

- **Punteros inteligentes:** Permiten manejar el tiempo de vida de los objetos, sin embargo, a diferencia de los objetos creados con memoria dinámica, los punteros inteligentes añaden características como el recolector de basura automático o el comprobador de límites
 - Cabecera: **#include <memory>**

10- NOVEDADES DE C++11

- **unique_ptr**: no es copiable, dos instancias de **unique_ptr** no puede administrar el mismo objeto. Además no se puede pasar por valor a una función ni utilizar en ningún algoritmo de la Biblioteca de plantillas estándar (STL) que requiera hacer copias. Un **unique_ptr** solo se puede mover. Esto significa que la propiedad del recurso de memoria se transfiere a otro **unique_ptr** y el **unique_ptr** original deja de poseerlo.

```
void foo(int* p) {  
    cout << *p << endl;  
}  
int main() {  
    unique_ptr<int> p1(new int(42));  
    unique_ptr<int> p2 = move(p1); // transfer ownership  
    if(p1)  
        foo(p1.get());  
    (*p2)++;  
    if(p2) foo(p2.get());  
    return 0;  
}
```


10- NOVEDADES DE C++11

- **shared_ptr**: está diseñado para escenarios en los que más de un propietario tendrá que administrar la duración del objeto en memoria.
- Después de inicializar **shared_ptr**, puede copiarlo, pasarlo por valor en argumentos de función y asignarlo a otras instancias de **shared_ptr**.
- Todas las instancias apuntan al mismo objeto y el acceso compartido a un "bloque de control" aumenta o disminuye el recuento de referencias siempre que un nuevo **shared_ptr** se agrega, se sale del ámbito o se restablece.
- Cuando el recuento de referencias llega a cero, el bloque de control elimina el recurso de memoria y se elimina a sí mismo.

10- NOVEDADES DE C++11

```
void foo(int* p) {  
    cout << "foo = " << *p << endl;  
}  
void bar(shared_ptr<int> p) {  
    cout << "Estamos en el bar ... biennn " << endl;  
    ++(*p);  
}  
int main() {  
    shared_ptr<int> p1(new int(42));  
    shared_ptr<int> p2 = p1;  
  
    bar(p1);  
    foo(p2.get());  
  
    return 0;  
}
```

10- NOVEDADES DE C++14

- Para usar C++14 en Qt deberemos añadir en el archivo .pro:
 - **QMAKE_CXXFLAGS += -std=c++14**
 - ó
 - **CONFIG += c++14**
- Más información sobre las novedades soportadas en el nuevo estándar para las versiones de GCC
 - <https://gcc.gnu.org/projects/cxx1y.html>

10- NOVEDADES DE C++14

○ Plantillas para las variables

```
template<typename T> constexpr T pi = T(3.1415926535897932385);  
// Usual specialization rules apply:  
template<>  
constexpr const char* pi<const char*> = "pi";
```

○ Literales estandar

- "s", para crear strings.
- "h", "min", "s", "ms", "us", "ns", para crear intervalos de tiempo (std::chrono::duration)

```
auto str = "hello world"s; // auto deduces string  
auto dur = 60s; // auto deduces chrono::seconds
```

10- NOVEDADES DE C++14

- **Separadores de dígitos**
- En C++14, el carácter comilla simple, también puede usarse como separador de millares en los literales numéricos, ambos entero literal y puntos flotantes literales. Esto puede hacer que sea más fácil analizar grandes cantidades.

```
auto integer_literal = 1'000'000;  
auto floating_point_literal = 0.000'015'3;  
auto binary_literal = 0100'1100'0110;
```

- Probar que novedades del C++14 están soportadas en nuestra versión de QtCreator (normalmente MinGW).

86