## QT & C++

**Install QT and pre-requisites:**

WINDOWS: Download and install QT Creator (install in C:\\Qt)

<https://www.qt.io/download>

WINDOWS: Add qmake to environment variables

# locate your qmake.exe

C:\Program Files\Microsoft SQL Server\140\PYTHON\_SERVER\Library\bin

Check that qmake is installed and which version is being used

qmake -v

If successful, you should see the following:

A close up of a logo

Description automatically generated

WINDOWS: Install g++ and add to environment variables

<http://www.cs.utah.edu/~alee/g++/g++old.html>

Assuming the following file structure…

./

├── main.cpp

└── main.h

And assuming terminal application (no qt) Compile using the following:

g++ -c -ansi -pedantic -Wall .\main.cpp -I main.h ; g++ main.o; ./a.exe

Directives/Headers

|  |  |  |
| --- | --- | --- |
| **Directive** | **Purpose** | **Usage** |
| QTextStream | Streams | cin(stdin); cout(stdout);  cin >> d; cout << “Hello World”; |
| QApplication | Execute application | a(argc, argv);  a.exec(); |
| QMessageBox | Modal dialog | QMessageBox::StandardButton reply;  reply = QMessageBox::question(0, "Coordinate", "Enter another coordinate?",QMessageBox::Yes|QMessageBox |
| QInputDialog |  | QInputDialog::getText(0,"Input","Please enter stuff”), QLineEdit::Normal, QString(""), &ok); |
| QDebug | Debugging to console | qDebug() << "Hello World"; |
| QString | Strings | QString::number(degrees);  myString.toUpper(); |
| QStringList | String Objects | inputCoordinates.split(" ") |
| QChar | Characters | QChar(0260); |
|  |  |  |
|  |  |  |

**Lesson 1**

Types & Classes

How to read UML diagrams:

A close up of a logo

Description automatically generatedA screenshot of a cell phone

Description automatically generated

**Lesson 2**

OOP

**Inheritance**

#indef CHILD\_H

#define CHILD\_H

#include “parent.h”

// Sub class inheriting from Base Class(Parent)

class Child: public Parent {

public :

int i\_c;

};

#endif // CHILD\_H

#indef PARENT\_H

#define PARENT\_H

class Parent {

public :

int id\_p;

};

#endif // PARENT\_H

#indef CHILDLIST \_H

#define CHILDLIST\_H

#include “child.h”

// Container class that manages a list of child

class ChildList: public QList<Child\*>{

public :

int i\_cl;

};

#endif // CHILDLIST \_H

#include <QTextStream>

#include <QApplication>

//main function

int main()

{

QApplication a;

a(argc, argv);

cout(stdout);

Child obj1;

// child has all data members and member functions of class parent

obj1.id\_c = 7;

obj1.id\_p = 91;

cout << "Child id is " << obj1.id\_c << endl;

cout << "Parent id is " << obj1.id\_p << endl;

a.exec();

}

**Polymorphism**

#indef PARENT\_H

#define PARENT\_H

class Parent {

public :

int id\_p;

//Constructor

Parent(int id);

//Add virtual destructor to make polymorphic

virtual ~Parent();

};

#endif // PARENT\_H

Change into a specific object type at runtime using dynamic\_cast function

#include <QTextStream>

#include <QApplication>

//main function

int main()

{

QApplication a;

a(argc, argv);

cout(stdout);

Parent c;

//Run-time Type Identification (RTTI), cast to a specific type

if(dynamic\_cast<Child\*>(c)->id\_c == “C49”){

cout << “Child ID is ‘C49’\n”;

}

a.exec();

}

In class definition:

#include "journalarticle.h"

//constructor for child class

JournalArticle::JournalArticle(QString \_title,

QStringList \_authors,

int \_year,

QString \_refID,

QString \_journalName,

int \_volume,

int \_number):

//constructor for parent class

Reference(\_title, \_authors, \_year, \_refID),volume(\_volume), number(\_number){

}