**Qt / C++ CheetSheet**

**Problem zur Lösung**

* In Teilprobleme aufteilen
* Funktionen oder Klassen finden
* Zwischenergebnisse testen
* Verfeinerung (Datenspeicher, Verzweigung, Schleife …)

**Datenspeicher**

**Variable im RAM (temorär)**

int, short, xx – Ganzzahl

boolean – true oder false

double, float – Kommazahl

char - Zeichen

QString - Zeichenkette

**Wertzuweisung**

Int i = 5;

**String-Funktionen**

**Konvertierung**

**Zahl nach String**

// int als String

QString s = QString::number(ganzZahl);

// double mit 6 Nachkommastellen

QString sd = QString::number(kommaZahl, 'f', 6);

**String nach Zahl**

double kommaZahl = eingabe.toDouble();

bool ok;

int ganzZahl = eingabe.toInt(&ok);

**Aus Datei lesen**

**Text-Datei einlesen**

QFile file("/home/gst/daten.txt");

if( !file.open(QIODevice::ReadOnly) )

{

QMessageBox::information(0, "error",   
 file.errorString());

}

QTextStream in(&file);

while( !in.atEnd() )

{

QString line = in.readLine(); // Zeile einlesen

QStringList fields = line.split(","); // aufsplitten

// Hier die fields weiterverarbeiten.

}

file.close();

**In Datei speichern (dauerhaft) speichern**

**Binärdateien**

**Directory-Funktionen**

**Wiederholung**

* Passende Schleife suchen

**For**: Anzahl bekannt. for( int i=0; i<10; i++) {}

**While**: Wiederhole Block, solange Bedingung true ist. Prüft am Anfang while (a < 4) {}

**Do**/**while**: Wiederhole Block mind. Einmal. Prüft Bedingung am Ende. do { } while (a < 5)

**Foreach**: Für jedes Element einer Liste

**Verzweigung**

If ( Bedingung ) // true oder false  
{ // Block für Bedingung == true  
}  
else  
{ // Block für Bedingung == false  
}

**Achtung:** In Bedingung == verwenden.

**Binäre Logik**

Int zahl=6;   
zahl == 7 ist false

(n>= 0) && (n<= 5) // beides muss erfüllt sein

True && false gibt false -> UND

True || false gibt true -> ODER

! (n<7) – Negation (aus true wird false und umgekehrt)

**Funktionen**

Rückgabewert, Funktionsname, Parameterliste

int berechne(short a, int b)  
{  
 return b/a;  
}

**Klassen**

**Klassenname**

**Konstruktor/Destruktor**

**Header: testclass.h**

#ifndef TESTCLASS\_H

#define TESTCLASS \_H

class TestClass

{

public:

// Konstruktor:

TestClass(const QString &xmlFilename);

// Destruktor:

~ TestClass ();

// Öffentliche Funktion:

void create\_CFile(); // Methode deklarieren

private: // Private Funktionen und Member-Variablen

QString createInitData();

int meineDaten;

};

#endif // TESTCLASS \_H

**Source: testclass.cpp**

#include "testclass.h" // Header einbinden

TestClass:: TestClass (const QString &xmlFilename)

{

meineDaten = 42; // Konstruktor implementieren

}

TestClass::~ TestClass ()

{

}

void TestClass::create\_CFile() // Öffentliche Methode implementieren

{

std:cout << „Hallo Welt\n“;

}

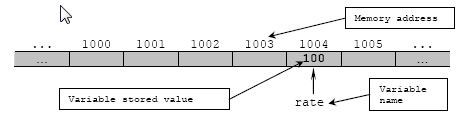
QString TestClass:: createInitData () // Private Methode implementieren

{

std:cout << „Hallo Privat\n“;

}

**C++ - Pointer:**



**Beispiel:**

* Variable „rate“ liegt an Speicheradresse 1004 im RAM-Speicher
* Größe=1byte -> Datentyp unsigned char
* Pointer „ptrRate“ auf „rate“ enthält Adresse (hier 1004)

1. Computerspeicher (RAM):  
   Jede Speicherstelle ist über eine Speicheradresse (Nummer) ansprechbar. Im Beispiel 1004.   
   An jeder Speicherstelle kann ein Wert abgelegt und später wieder gelesen werden.
2. Variablen können eine oder mehrere Speicherstellen benutzen.  
   In Programmiersprachen wird der Variablenname benutzt, da für Menschen leichter lesbar als Adressen.
3. Variable im Speicher anlegen:  
    unsigned char rate = 100;  
   Variable „rate“ hat eine Adresse im Speicher. Im Beispiel 1004.  
   Dort wird der Wert (hier 100) abgespeichert.
4. Pointer (Zeiger) auf „rate“ anlegen:

unsigned char\* ptrRate = &rate; // Weise Adresse von Variable „rate“ zu

1. Zugriff auf den Wert über den Pointer:  
   unsigned char bs = \*ptrRate; // Pointer dereferenzieren (bs ist 100)

<https://debugmode.net/2013/12/20/pointers-in-c/>

Array-Variable kann als Pointer verwendet werden.

**Sonstiges**

**Hilfe**

F1 im Qt Creator oder Google („qt convert int to string“)

**Vorsicht**

Bei Pointern besondere Vorsicht.

**Debugzeile auf Commandline**

#include <QtDebug>

qDebug() << "javaCPU=" << timestamp << " // " << javaCpu ;

**Exceptions**

**Threads**

**GUI (Graphical Unser Interface)**

**Textfeld einzeilig**

QString eingabe = ui->lineEdit->text();

ui->lineEdit->setText("Eingabe=" + eingabe);

**Textfeld mehrzeilig**

ui->textEdit->toPlainText();

ui->textEdit->setPlainText();

**QMessageBox**

QMessageBox messageBox;

messageBox.critical(0,"Error","Startproblem !");

messageBox.setFixedSize(500,200);

messageBox.show();

**Timer**#include <QTimer>   
QTimer::singleShot(startDelay\*1000, this,   
 SLOT(checkAlarm()));

private slots:  
 void checkAlarm();

**Qt Signals und Slots**

**Pfluecker-Objekt**

**pfluecker.h**

#include <QObject>

class Pfluecker : public QObject

{

Q\_OBJECT

…

signals:

void sendInc(int increment); // Definiere Signal

…

};

**pfluecker.cpp**

…

emit sendInc( incWert ); // Verschicke Signal

…

**Mainwindow-Objekt**

**mainwindow.h**

#include <QObject>

class MainWindow : public QMainWindow

{

Q\_OBJECT

…

private slots:

void incCounter(int increment); // Definiere Slot

…  
};

**mainwindow.cpp**

// Verbinde Signal und Slot

QObject::connect(&meinPflue, SIGNAL( sendInc(int) ),

this, SLOT ( incCounter(int) ) );

// meinPflue = Instanz von Objekt Pfluecker

// this = aktuelles Objekt, hier mainwindow

…

// Implementiere Slot:

// Verarbeitet empfangenes Signal

void MainWindow::incCounter(int increment)

{

…

}

Verweise:

<https://vorkurs.informatik.rwth-aachen.de/wordpress/wp-uploads/2016/09/Java-Cheatsheet.pdf>

Todo:

PropertyFile