

**Pokročilé programování
v jazyce C pro chemiky
(C3220)**

Knihovna Qt - část 1.

Implicitní parametry funkce

- Pro funkce (a metody) které přijímají parametry můžeme definovat implicitní hodnoty parametrů
- Funkci potom můžeme volat jako kdyby byla bez parametrů, přičemž parametrům je přiřazena implicitní hodnota

```
// Parametru funkce priradime implicitni hodnotu 10
void printValue(int val = 10)
{
    cout << "Hodnota: " << val << endl;
}

int main()
{
    int a = 3;
    // Pokud funkci zavolame bez parametru, pouzije se implicitni hodnota
    // v tomto pripade se tedy na obrazovku vypise hodnota 10
    printValue();

    // Pokud funkci parametr predame, pouzije se predana hodnota.
    // Nasledujici volani funkce zpusobi vypis hodnoty 7
    printValue(7);
    // Nasledujici volani funkce zpusobi vypis hodnoty v promenne a tj. 3
    printValue(a);
    return 0;
}
```

Implicitní parametry funkce

- Pokud má funkce více parametrů, můžeme definovat implicitní hodnoty jen pro některé parametry. Nejdříve uvádíme parametry bez implicitních hodnot a až potom parametry s implicitními hodnotami.
- Při volání funkce musíme povinně předat všechny parametry které nemají specifikovány implicitní hodnoty. Parametry, které mají specifikovány implicitní hodnoty předávat nemusíme (v takovém případě se použijí jejich implicitní hodnoty).

```
// Některým parametrům funkce přiřadíme implicitní hodnoty
void printValue(int n1, int n2, int n3 = 5, int n4 = 10)
{
    cout << "Hodnoty: " << n1 << n2 << n3 << n4 << endl;
}

int main()
{
    int a = 3, b = 9, c = 1, d = 2;
    // Všechna následující volání jsou platná
    printValue(a, b);
    printValue(a, b, c);
    printValue(a, b, c, d);
    return 0;
}
```

Nepojmenované proměnné

- Někdy potřebujeme vytvořit proměnnou pouze za účelem předání do volané funkce, v takovém případě ji můžeme definovat až v okamžiku předávání a nemusíme ji nijak pojmenovávat

```
// Tady bude definována trída Vector3D, viz. předchozí přednasky

void printVector(const Vector3D &v)
{
    cout << "Vektor: " << v.getX() << v.getY() << v.getZ() << endl;
}

int main()
{
    // Do funkce printVector() předáme nepojmenovanou proměnnou, která
    // bude vytvořena až v okamžiku volání funkce; hodnotu této
    // nepojmenované proměnné nastavíme předáním příslušných hodnot
    // do konstruktoru
    printVector(Vector3D(1.0, 2.0, 3.0));

    // Také můžeme udělat následující: dvě nepojmenované proměnné se
    // předají do operátoru + a ten vrátí hodnotu, která se
    // předá do printVector();
    printVector(Vector3D(4.0, 5.0, 4.0) + Vector3D(1.0, 2.0, 3.0));
    return 0;
}
```

Grafické objektové knihovny

- Pro tvorbu grafických aplikací v C++ se používají objektové knihovny, které obsahují třídy pro práci s okny, interaktivními prvky (tlačítka, seznamy, editační políčka, ...) a podpůrné třídy
- Nejznámější knihovny:
 - MFC** (Microsoft Foundation Classes) – knihovna od firmy Microsoft pro tvorbu programů pro MS Windows (postupně nahrazovaná novější .NET Framework)
 - .NET Framework** – novější rozsáhlá knihovna od firmy Microsoft pro tvorbu grafických i negrafických programů
 - Qt** – grafická multiplatformová knihovna dostupná pod open-source licencí
 - GTK+** - open-source knihovna původně určená pro Unixové systémy, nyní multiplatformová
 - wxWidgets** – open-source multiplatformová knihovna
 - FLTK** – malá jednoduchá open-source multiplatf. knihovna
- Více informací: http://en.wikipedia.org/wiki/List_of_widget_toolkits

Knihovna Qt

- Vlastnosti knihovny Qt:
 - Dostupná jako open-source pod GPL licencí (a dalšími licencemi vč. komerčních)
 - Multiplatformová, podporuje operační systémy Unix a Linux, MS Windows, Mac OS X
 - Kvalitní implementace tříd pro tvorbu grafických rozhraní
 - Obsahuje také třídy pro práci s databázemi, soubory v XML formátu, síťové aplikace atd.
- Více informací na <http://www.qt.io/>
- Dokumentace ke Qt knihovně: <http://doc.qt.io/qt-5/>
- Kniha o Qt: [C++ GUI Programming with Qt4](#)
 - 1. vydání je volně k dispozici v PDF formátu na:
<http://www.qtrac.eu/C++-GUI-Programming-with-Qt-4-1st-ed.zip>
 - 2. vydání v HTML formátu (pod záložkou *Sample Content*):
<http://www.informit.com/store/product.aspx?isbn=0132354160>

Kompilace programu používajícího Qt

- Knihovna Qt vyžaduje složitější překlad, pro snadnější práci je s knihovnou dodáván program *qmake*, který slouží k automatickému generování souboru *Makefile*
- Na počítačích používajících systém modulů (např. v poč. učebně) je potřeba zpřístupnit Qt knihovnu příkazem: *module add qt*
- Každý program umístíme do samostatného adresáře, jehož jméno by mělo odpovídat jménu projektu
- V adresáři vytvoříme soubory **.cpp* se zdrojovým kódem programu
- V adresáři spustíme *qmake -project* čímž vygenerujeme soubor projektu (má koncovku *.pro* a jméno je totožné s názvem adresáře)
- Potom vygenerujeme *Makefile* příkazem *qmake soubor.pro* kde *soubor.pro* je jméno souboru s projektem (vygenerovaný v předchozím kroku)
- Kompilaci spouštíme příkazem *make* (bez parametrů)
- Při každé změně zdrojového kódu vždy spustíme kompilaci zavoláním příkazu *make* (tj. nemusíme již opakovat předcházející kroky s voláním *qmake*)

Program v Qt

- Pro použití knihovny Qt je třeba vložit příslušné hlavičkové soubory, pro každou třídu existuje samostatný hlavičkový soubor jehož název odpovídá názvu třídy
- Každý program používající Qt knihovnu musí vytvořit objekt třídy `QApplication` a zavolat její metodu `exec()`
- Konstruktore třídy `QApplication` se předají parametry z příkazového řádku
- Metoda `exec()` obsahuje cyklus který načítá a zpracovává vstup z klávesnice a myši dokud není program ukončen

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

int main(int argc, char *argv[])
{
    // Vytvorime objekt tridy QApplication a do konstruktore predame
    // parametry prikazoveho radku
    QApplication app(argc, argv);
    // Zavolame metodu exec() ktera obsahuje cyklus udalosti, ktery
    // zpracovava vstup z klavesnice a mysi dokud neni program ukoncen
    return app.exec();
}
```


Vytvoření okna v Qt

- Pro vytvoření okna v Qt se používá třída `QWidget`
- Po vytvoření objektu třídy `QWidget` se pro zobrazení okna musí zavolat metoda `show()`
- Titulek v záhlaví okna můžeme nastavit metodou `setWindowTitle()`
- Objekty oken alokujeme dynamicky s použitím operátoru `new`

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

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    // Dynamicky alokujeme objekt okna tj. objekt tridy QWidget
    QWidget* testWidget = new QWidget;
    // Nastavime titulek ktery se zobrazi v zahlavi okna
    testWidget->setWindowTitle("Muj prvni program vytvoreny v Qt!");
    // Zavolame metodu show() ktera okno zobrazi
    testWidget->show();
    return app.exec();
}
```

Grafický výstup v Qt

- Pro kreslení do okna vytvoříme třídu odvozenou z `QWidget` a tu použijeme pro vytvoření objektu okna
- Kreslení v okně se provádí definováním virtuální metody `paintEvent()`, která překrývá metodu `paintEvent()` definovanou v `QWidget` (tato metoda je **protected**)
- Metoda `paintEvent()` je zavolána vždy když je potřeba okno překreslit

```
class GraphicWidget : public QWidget
{
public:
    GraphicWidget() {};
protected:
    // V metode paintEvent() budou uvedeny prikazy pro kresleni
    virtual void paintEvent(QPaintEvent* event);
};

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    GraphicWidget* graphicWidget = new GraphicWidget;
    graphicWidget->setWindowTitle("Program s vlastnim widgetem!");
    graphicWidget->show();
    return app.exec();
}
```

Třída QPainter

- Pro kreslení vytvoříme v metodě `paintEvent()` objekt typu `QPainter`, do konstruktoru předáme ukazatel na objekt volající metody, který získáme pomocí klíčového slova **this**
- Třída `QPainter` obsahuje metody pro kreslení, např.:
 - `void drawLine (int x1, int y1, int x2, int y2)`
 - `void drawRect (int x, int y, int width, int height)`
 - `void drawEllipse (int x, int y, int width, int height)`
- Počátek souřadnic je v levém horním rohu, souřadnice y nabývá kladných hodnot směrem dolů, souřadnice x směrem doprava

```
    // Na zacatku musime vlozit hlavickovy soubor QPainter
void GraphicWidget::paintEvent(QPaintEvent* event)
{
    QPainter painter(this);
    // Kresli caru z bodu se souradnicemi 30, 10 do bodu 270, 290
    painter.drawLine(30, 10, 270, 290);
    // Kresli obdelnik se souradnicemi leveho horniho rohu 170 a 20
    // a sirkou 110 a vyskou 80
    painter.drawRect(170, 20, 110, 80);
    // Kresli elipsu se souradnicemi leveho hornimu rohu 20 a 190 a
    // sirkou 120 a vyskou 90
    painter.drawEllipse(20, 190, 120, 90);
}
```

Třídy pro nastavení parametrů kreslení

- Pro nastavení barev pro kreslení, tloušťky čáry, výplně, velikosti a typu fontu a pod. používáme pomocné třídy, které uchovávají příslušné hodnoty:
 - `QColor` – pomocná třída pro specifikaci barvy je využívána následujícími třídami
 - `QPen` – pro nastavení tloušťky, barvy a stylu čáry
 - `QBrush` – pro nastavení typu a barvy výplně
 - `QFont` – pro nastavení typu, velikosti a barvy písma
- Hodnoty často specifikujeme tak, že je předáme do konstruktoru; tyto třídy obsahují několik přetížených konstruktorů umožňující předat jen ty parametry které chceme nastavit
- V objektu typu `QPainter` voláme metody `setPen()`, `setBrush()`, `setFont()`, kterým předáme proměnné těchto typů jako parametry
- V těchto případech často používáme nepojmenované proměnné

Třída QColor

- Třída QColor obsahuje celočíselné hodnoty 0 – 255 složek barvy (red, blue, green) a průhlednost (tzv. alfa kanál)
- Hodnoty složek barvy nejčastěji předáváme přímo do konstruktoru `QColor (int r, int g, int b, int a = 255)`
- V knihovně Qt je předdefinovaných několik barev: `Qt::white`, `Qt::black`, `Qt::red`, `Qt::green`, `Qt::blue`, `Qt::cyan`, `Qt::magenta`, `Qt::yellow`, `Qt::gray` a další (viz. <http://doc.qt.io/qt-5/qt.html#GlobalColor-enum>)
- Objekty QColor se často využívají pro nastavení barvy v objektech QPen a QBrush

```
QColor color1(128, 210, 255);

QPen pen1(color1);
QPen pen2(Qt::red);
QPen pen3(QColor(90, 128, 128));

QBrush brush1(color1);
QBrush brush2(Qt::green);
QBrush brush3(QColor(90, 128, 128));
```

Třída QPen

- Třída QPen obsahuje parametry pro kreslení čar a obrysů
- Třída obsahuje informace o barvě, tloušťce čáry, stylu čáry (rovná, čárkovaná a pod.) a některé další parametry
- Hodnoty se zpravidla předávají přímo do konstruktorů:
`QPen (const QColor & color)`
`QPen (const QBrush & brush, qreal width,
Qt::PenStyle style = Qt::SolidLine,
Qt::PenCapStyle cap = Qt::SquareCap,
Qt::PenJoinStyle join = Qt::BevelJoin)`
- Více na: <http://doc.qt.io/qt-5/qpen.html>

```
// Vytvori pero cervene barvy, ostatni hodnoty budou implicitni
QPen pen1(Qt::red);

// Vytvori pero modre barvy s tloustkou cary 10 pixelu
QPen pen2(QBrush(Qt::blue), 10);

// Misto parametru QBrush muzeme predat primo hodnotu barvy
QPen pen3(Qt::green, 10);

// Vytvori cervene pero s tloustkou 5 kreslici carkovanou caru
QPen pen4(Qt::red, 5, Qt::DashLine);
```

Třída QBrush

- Třída QBrush obsahuje parametry pro kreslení výplně objektů
- Třída obsahuje informace o barvě výplně a stylu výplně
- Hodnoty se zpravidla předávají přímo do konstruktoru:
QBrush (const QColor & color,
Qt::BrushStyle style = Qt::SolidPattern)
- Více na: <http://doc.qt.io/qt-5/qbrush.html>

```
// Vytvorime stetec pro vyplneni souvislou cernou barvou
QBrush brush1(Qt::red);

// Vytvorime stetec pro vyplneni barvou s prislusnymi hodnotami RGB
QBrush brush2(QColor(128, 100, 190));

// Vytvorime zeleny stetec s vyplnovym vzorem tvorenym
// diagonalnimi carami
QBrush brush3(Qt::green, Qt::BDiagPattern);
```

Třída QFont

- Třída QFont slouží pro specifikaci fontu
- Specifikovat lze název fontu, velikost fontu, tučnost písma a použití kurzívy
- Hodnoty se zpravidla předávají do konstruktoru:

```
QFont ( const QString & family, int pointSize = -1,  
        int weight = -1, bool italic = false )
```

- Je-li hodnota pointSize záporná, použije se velikost fontu 12
- Parametr weight nabývá hodnot 0 – 99, lze použít předdefinované konstanty QFont::Normal, QFont::Bold a další
- Pro vypsání textu do okna se používá metoda:

```
void drawText ( int x, int y, const QString & text )
```

```
QPainter painter(this);  
    // Vytvoríme font Arial s implicitní velikostí 12 bodů  
QFont font1("Arial");  
    // Vytvoríme font Times New Roman s velikostí 18 bodů  
QFont font2("Times New Roman", 18);  
    // Vytvoríme font Helvetica s velikostí 18 bodů, tučná kurzíva  
QFont font3("Helvetica", 18, QFont::Bold, true);  
painter.setFont(font3);  
painter.drawText(20, 330, "Toto je text");
```


Použití tříd QColor, QPen, QBrush a QFont

- Třídy QColor, QPen, QBrush a QFont používáme ve spojení s metodami třídy QPainter pro nastavení parametrů kreslení:

```
void setPen ( const QPen & pen )  
void setBrush ( const QBrush & brush )  
void setFont ( const QFont & font )
```

- Třídy můžeme použít pro vytvoření objektové proměnné nebo je používáme přímo tj. jako nepojmenované proměnné

```
QPainter painter(this);  
QPen pen1(Qt::blue, 4);  
QBrush brush1(Qt::red);  
  
painter.setPen(pen1);  
painter.setBrush(brush1);  
painter.drawRect(170, 20, 110, 80);  
  
painter.setPen(QPen(QColor(255, 0, 0), 2));  
painter.setBrush(QBrush(Qt::green));  
painter.drawEllipse(20, 190, 120, 90);  
  
// Misto QBrush muzeme pouzivat barvu QColor, protoze QBrush ma  
// konstruktor který jej umi konvertovat na QColor  
painter.setBrush(Qt::blue);  
painter.drawRect(20, 20, 110, 80);
```

Zpracování událostí od myši

- Události od myši jsou zpracovávány v cyklu událostí v metodě `QApplication::exec()`, která při výskytu události (kliknutí myši, pohyb myši) zavolá příslušné metody okna (tj. metody třídy `QWidget`) v němž se nachází kurzor myši
- Pro různé události myši se volají odlišné metody:

```
virtual void mousePressEvent ( QMouseEvent * event )
virtual void mouseReleaseEvent ( QMouseEvent * event )
virtual void mouseDoubleClickEvent (
    QMouseEvent * event )
virtual void mouseMoveEvent ( QMouseEvent * event )
```
- Pro zpracování události od myši definujeme příslušnou virtuální metodu (např. `mousePressEvent()`), která překryje příslušnou metodu definovanou v `QWidget` (je **protected**)
- Informace o pozici kurzoru a stisknutém tlačítku získáme z parametru `event`, který je objektem třídy `QMouseEvent`
- Třída `QMouseEvent` obsahuje metody `x()` a `y()` poskytující souřadnice kurzoru myši a `button()` pro získání informace o stisknutém tlačítku (nabývá hodnot `Qt::LeftButton`, `Qt::MidButton`, `Qt::RightButton`)

Zpracování událostí od myši - příklad

```
// Vytvoríme třídu GraphicWidget, kterou použijeme pro vytvoření okna
class GraphicWidget : public QWidget
{
public:
    GraphicWidget();
protected:
    virtual void paintEvent(QPaintEvent *event);
    virtual void mousePressEvent(QMouseEvent *event);
    // Zde mohou být další členy třídy
};

// Metoda mousePressEvent() bude volána pokřazde, když stiskneme
// tlačítko myši
void GraphicWidget::mousePressEvent(QMouseEvent *event)
{
    cout << "Stisknuto tlačítko myši!" << endl;
    cout << "    Souradnice myši: ";
    cout << event->x() << ", " << event->y() << endl;
    cout << "    Je stisknuto následující tlačítko myši: ";
    if (event->button() == Qt::LeftButton) cout << "levo";
    if (event->button() == Qt::MidButton) cout << "prostřední";
    if (event->button() == Qt::RightButton) cout << "právo";
    cout << endl;
}
```

Událost od myši a překreslení okna

- Na událost od myši program často potřebuje reagovat tak, že změní obsah vykreslovaný do okna
- Pokud chceme vynutit překreslení okna, zavoláme metodu `update()` třídy `QWidget`:
`void update ()`
- Po zavolání metody `update()` provede program překreslení, tedy zavolá metodu `paintEvent()` pro dané okno.

Událost od myši a překreslení okna - příklad

```
// Na začátku musíme vložit hlavičkový soubor QMouseEvent
class GraphicWidget : public QWidget
{
public:
    GraphicWidget();
protected:
    virtual void mousePressEvent(QMouseEvent *event);
    virtual void paintEvent(QPaintEvent *event);
private:
    // Nasledující proměnné slouží pro uložení souřadnic myši
    // které se jim přiřadí vždy při události stisknutí myši
    // a pak se použijí pro vykreslení čtverečku na dané pozici.
    // Hodnoty je třeba inicializovat v konstruktoru.
    int posX, posY;
};

void GraphicWidget::mousePressEvent(QMouseEvent *event)
{
    posX = event->x(); posY = event->y();
    update(); // Vyvoláme požadavek na překreslení okna
}

void GraphicWidget::paintEvent(QPaintEvent *event)
{
    QPainter painter(this);
    painter.drawRect(posX - 5, posY - 5, 10, 10);
}
```

Použití třídy `Application` s Qt

- Pokud v programu používáme třídu `Application`, je vhodné ji odvodit z `QApplication`
- Konstruktor třídy `Application` musí přijímat argumenty z příkazového řádku a předat je konstruktoru `QApplication`
- Destruktor deklaruujeme jako virtuální (destruktor by měl být vždy virtuální pokud třída nebo její základní třída obsahuje virtuální metody, což je i případ třídy `QApplication`)

```
#include <iostream>
#include <QApplication>
#include <QWidget>
using namespace std;
// Na zacatku bude definovana trida GraphicWidget

class Application : public QApplication
{
public:
    Application(int &argc, char *argv[]);
    virtual ~Application();
    int run();
private:
    GraphicWidget* graphicWidget;
};
// Pokracovani na nasledujici strance
```

Použití třídy Application s Qt - pokračování

```
// Pokracovani z predchozi stranky
Application::Application(int &argc, char *argv[]) : QApplication(argc, argv)
{
    graphicWidget = 0;
}

Application::~Application()
{
    if (graphicWidget != 0)
        delete graphicWidget; // Vymazeme dynamicky alokovane objekty
    graphicWidget = 0;
}

int Application::run()
{
    graphicWidget = new GraphicWidget;
    graphicWidget->setWindowTitle("Muj program vytvoreny v Qt!");
    graphicWidget->show();
    return QApplication::exec();
}

int main(int argc, char *argv[])
{
    Application app(argc, argv);
    return app.run();
}
```

Pomocné třídy v Qt

- Knihovna Qt hojně používá následující pomocné třídy
 - `QPoint` pro bod se souřadnicemi x a y
 - `QRect` pro obdélníkový region
 - `QString` jako kvalitnější alternativa ke třídě `string`
 - `QFile` pro práci se soubory (kvalitnější alternativa k `iostream`)
- Ke třídám `QPoint` a `QRect` existují dvě podobné alternativy `QPointF` a `QRectF` které pracují s hodnotami typu `float` namísto `int`
- Dokumentace k jednotlivým třídám:
<http://doc.qt.io/qt-5/classes.html>

Dodržujte následující pravidla

- Pracujete-li v počítačové učebně, nezapomeňte aktivovat modul s knihovnou Qt: *module add qt*
- Zdrojové soubory umístěte do samostatného adresáře. Pak vygenerujte soubor projektu a potom *Makefile*.
- Poté co vygenerujete soubor projektu **.pro* (příkazem *qmake -project*) nezapomeňte na začátek tohoto souboru přidat řádek *QT += widgets*. Teprve potom vygenerujte soubor *Makefile* příkazem *qmake *.pro*
- Při použití kterékoli třídy z Qt knihovny nezapomeňte vložit příslušný hlavičkový soubor (má stejný název jako třída).
- Používejte dokumentaci k jednotlivým třídám:
<http://doc.qt.io/qt-5/classes.html>
- Seznam barev předdefinovaných v Qt knihovně lze najít zde:
<http://doc.qt.io/qt-5/qt.html#GlobalColor-enum>

Cvičení

1. Vytvořte program využívající knihovnu Qt, který nakreslí do okna elipsu, obdélník a čáru jak je uvedeno na obrázku. Obdélník bude vyplněný barvou s RGB hodnotami 255, 50, 120 a elipsa modrou barvou (`Qt::blue`). Tloušťka čar bude 7. Do spodní části okna se vypíše libovolný text (fontem Helvetica, velikost 18, tučné písmo, barva `Qt::cyan`). **3 body**
2. Do programu přidejte obsluhu stisknutí levého tlačítka myši tak, že po stisknutí se vypíše na terminál informace o pozici kurzoru myši a stisknutém tlačítku. Dále se po stisknutí tlačítka nakreslí v okně čtvereček na místě kde bylo tlačítko stisknuto. **2 body**

