9. 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, je možné 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 hodnot. Parametry, které mají specifikovány implicitní hodnoty předávat nemusíme (v takovém případě se použijí jejich implicitní hodnoty).

```
// Nekterym parametrum funkce priradime
// implicitni 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;
 // Vsechna nasledujici volani jsou platna
  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 definovana trida Vect3d,
// viz. predchozi prednasky

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

int main()
{
  // Do funkce printVector() predame
  // nepojmenovanou promennou, ktera
  // bude vytvorena az v okamziku volani
  // funkce; hodnotu teto nepojmenovane
  // promenne nastavime predanim
  // prislusnych hodnot do konstruktoru
  printVector(Vector3D(1.0, 2.0, 3.0));

  // Take muzeme udelat nasledujici: dve
  // nepojmenovane promenne se predaji do
  // operatoru + a ten vrati hodnotu,
  // ktera se preda do printVector();
  printVector(Vector3D(4.0, 5.0, 4.0) +
              Vector3D(1.0, 2.0, 3.0));
  return 0;
}
```

Kompilace programu používajícího Qt knihovnu

Knihovna Qt používá program *qmake*, který slouží k automatickému generování souboru *Makefile*.

Na počítačích používajících systém modulů (např. v počítačové učebně) je potřeba zpřístupnit Qt knihovnu příkazem: *module add qt*

Každý program umisťujeme 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).

Na začátek souboru projektu přidejte řádek *QT += widgets*.

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()`.

Konstruktoru 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 konstruktoru 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);
};

// Definice metody
// GraphicWidget::paintEvent je ukazana
// v prikladu na druhe strane

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

  GraphicWidget* graphicWidget
                  = new GraphicWidget;
  graphicWidget->setWindowTitle(
     "Muj 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 a QPaintEvent
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.

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.

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 )
```

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

```
// Vytvorime stetec pro vyplneni
// 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)
```

Pokud je 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);

// Vytvorime font Arial s implicitni
// velikosti 12 bodu
QFont font1("Arial");

// Vytvorime font Times New Roman s
// velikosti 18 bodu
QFont font2("Times New Roman", 18);

// Vytvorime font Helvetica s velikosti
// 18 bodu, tucna kurziva
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
// ktery 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 myší) 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 muši a button() pro získání informace o stisknutém tlačítku (nabývá hodnot Qt::LeftButton, Qt::MidButton, Qt::RightButton).

```
// Vytvorime tridu GraphicWidget, kterou
// pouzijeme pro vytvoreni okna
class GraphicWidget : public QWidget
{
  public:
    GraphicWidget();
  protected:
    virtual void paintEvent(
            QPaintEvent *event);
    virtual void mousePressEvent(
            QMouseEvent *event);
    // Zde mohou byt dalsi cleny tridy
};

// Metoda mousePressEvent() bude volana
// pokazde, kdyz stiskneme tlacitko mysi
void GraphicWidget::mousePressEvent(
                QMouseEvent *event)
{
  cout << "Stisknuto tlacitko mysi"<<endl;
  cout << "  Souradnice mysi: ";
  cout << event->x() << ", "
       << event->y() << endl;
  cout << " Je stisknuto tlacitko: ";
  if (event->button() == Qt::LeftButton)
     cout << "leve";
  if (event->button() == Qt::MidButton)
     cout << "prostredni";
  if (event->button() == Qt::RightButton)
     cout << "prave";
  cout << endl;
}
```

Událost od myši a překreslení okna

Na událost od myši program často potřebuje ragovat 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.

```
// Na zacatku musime vlozit hlavickovy
// soubor QMouseEvent

class GraphicWidget : public QWidget
{
  public:
    GraphicWidget();
  protected:
    virtual void mousePressEvent(
          QMouseEvent *event);
    virtual void paintEvent(
          QPaintEvent *event);
  private:
    // Nasledujici promenne slouzi pro
    // ulozeni souradnic mysi ktere se
    // jim priradi vzdy pri udalosti
    // stisknuti mysi a pak se pouziji pro
    // vykresleni ctverecku na dane pozici
    // Hodnoty je treba inicializovat
    //v konstruktoru.
    int posX, posY;
};

void GraphicWidget::mousePressEvent(
               QMouseEvent *event)
{
  posX = event->x();
  posY = event->y();

 // Vyvolame pozadavek na prekresleni okna
  update();
}

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 deklarujeme 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 (viz. predchozi stranky)

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

Application::Application(int &argc,
  char *argv[]) : QApplication(argc, argv)
{
  graphicWidget = 0;
}

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

Dodržujte následující pravidla

- Pracujete-li v počítačové učebně, nezapomeňte aktivovat modul s knihovnou Qt: *module add qt*
- Zdrojové soubor(y) 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

Úloha 1 3 body

Vytvořte program využívající knihovnu Qt, který nakreslí do okna elipsu, obdélník a čáru jek je uvedeno na obrázku. Obdélník bude vyplněný barvou s RGB hodnotani 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`).

Nápověda:

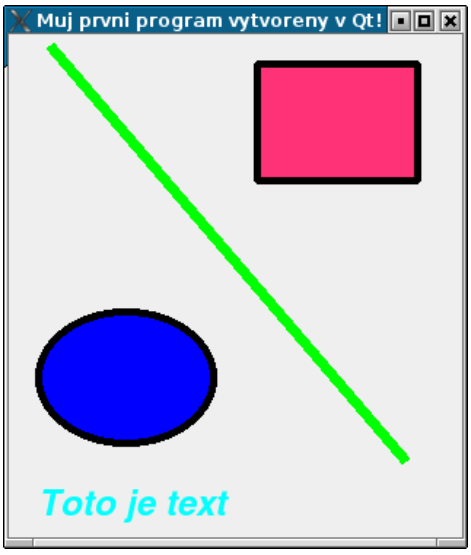
Vytvořte adresář vhodného jména a do něj umístěte soubor se zdrojovým kódem programu. Do souboru zapište kód podle ukázkového příkladu v sekci "Program v Qt" na str. 2. Vygenerujte soubor projektu a *Mekefile* a přeložte (viz. sekce "Kompilace programu používajícího Qt knihovnu").

Do funkce `main()` přidejte kód pro vytvoření okna, jak je ukázáno v příkladu v sekci "Vytvoření okna v Qt". Nezapomeňte vložit hlavičkový soubor *QWidget*. Program přeložte (příkazem *make*) a otestujte.

Vytvořte třídu `GraphicWidget` jak je ukázáno v příkladu v sekci "Grafický výstup v Qt" na str. 2. Vytvořte tělo konstruktoru (které však nebude obsahovat žádný kód). Vytvořte tělo metody `paintEvent()` jak je ukázáno v příkladu v sekci "Třída QPainter" na str. 3. Nezapomeňte vložit hlavičkový soubor *QPainter* a *QPaintEvent*. Funkci `main()` upravte tak, aby se pro okno vytvářel objekt `GraphicWidget` místo `QWidget` (tj. podle příkladu v sekci "Grafický výstup v Qt" na str. 2). Program otestujte (měl by se nakreslit obdélník, elipsa a čára jako na obrázku dole, ale budou nakresleny jen tenkou černou čarou a nebudou vyplněné, okno také může mít jiný rozměr než na obrázku).

V metodě `GraphicWidget::paintEvent()` nastavte před vykreslením každého objektu tloušťku a barvu čáry, případně barvu výplně (vi. příklady v sekcích "Třída QPen" a "Třída QBrush" na str. 3 a "Použití tříd QColor, QPen, QBrush a QFont" na str. 4). Dále přidejte kód pro vypsání textu ve spodní části okna (viz. příklad v sekci "Třída QFont" na str. 4).

Program předělejte tak aby využíval třídu Application jak je uvedeno v příkladu v sekci "Použití třídy Application s Qt" na str. 5.



Úloha 2 2 body

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.

Nápověda:

Vyjděte z předchozí úlohy a do třídy `GraphicWidget` přidejte metodu `mousePressEvent()` jak je ukázáno v příkladu v sekci "Zpracování událostí od myši" na str. 4. Program otestujte (klikejte na různá místa v okně a sledujte výpisy na obrazovku).

Do třídy `GraphicWidget` přidejte členské proměnné `posX` a `posY` a inicializujte jejich hodnoty v konstruktoru (hodnotou 0). Metodu `GraphicWidget::mousePressEvent()` doplňte o kód pro nastavení hodnot `posX` a `posY` jak je ukázáno v příkladu v sekci "Událost od myši a překreslení okna" an str. 5. Metodu `GraphicWidget::paintEvent()` doplňte o kód pro vykreslení čtverečku (viz. příklad v sekci "Událost od myši a překreslení okna" na str. 5), barvu výplně i orámování čtverečku ale nastavte na červenou.