

Interfejsy aplikacji w środowisku Windows

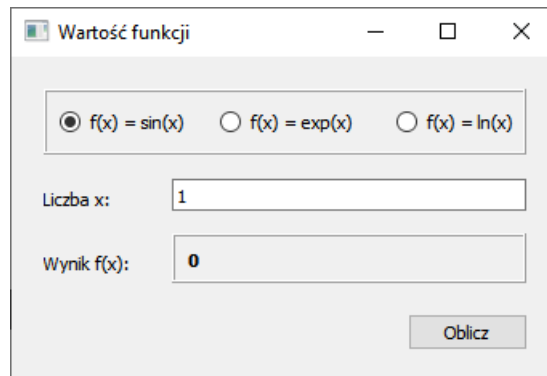
Qt – kontrolki, zadania

Kontrolki - zadania

1. Zadanie

Opracuj aplikację pozwalającą na obliczanie wartości funkcji ($\sin()$, $\exp()$, $\ln()$).

- Do budowy interfejsu użytkownika użyj kontrolerek:
 - etykieta `QLabel`,
 - pole edycyjne `QLineEdit`,
 - przycisk `QPushButton`,
 - przełącznik `QRadioButton`,
 - ramka `QFrame`.



Uzupełnij program o mechanizm sprawdzania poprawności wpisywanych danych (odpowiedni walidator i lokalizacja).

- W przypadku wartości logarytmu naturalnego program musi sprawdzać wartość argumentu (większy od zera), program powinien wyświetlać komunikat jeśli wartość argumentu jest błędna.
- Przełączniki `QRadioButton` powinny być umieszczone w ramce `QFrame`.
- Do sprawdzania aktywnego przełącznika wykorzystaj metodę `isChecked()` klasy `QRadioButton`.

```
if(ui->radioButton->isChecked())
{
    f = sin(x);
}
else
...
```

- W przypadku większej liczby różnie grupowanych przełączników, można (a nawet trzeba) tworzyć z nich grupy, zaznacz przełączniki należące do grupy i z menu podręcznego wybierz opcję **Assign to button group | New button group** (powstanie nowy obiekt o nazwie `buttonGroup` typu `QButtonGroup`).
- Konstruktor klasy okna uzupełnij o kod przypisujący identyfikatory przełączników w grupie.

```
ui->buttonGroup->setId(ui->radioButton, 0);
ui->buttonGroup->setId(ui->radioButton_2, 1);
ui->buttonGroup->setId(ui->radioButton_3, 2);
```

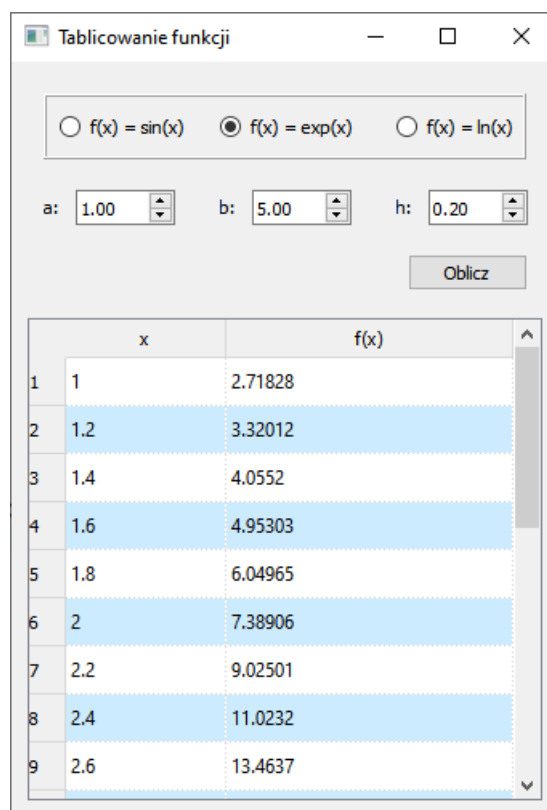
- Teraz można sprawdzać przełączniki za pomocą metody `checkedId()`.

```
switch(ui->buttonGroup->checkedId())
{
    case 0:
    {
        f = sin(x);
        break;
    }
    ...
}
```

2. Zadanie

Opracuj aplikację pozwalającą na tablicowanie wartości funkcji ($\sin()$, $\exp()$, $\ln()$).

- Do budowy interfejsu użytkownika użyj kontroltek:
 - etykieta `QLabel`,
 - pole edycyjne `QDoubleSpinBox`
 - przycisk `QPushButton`,
 - przełącznik `QRadioButton`,
 - ramka `QFrame`,
 - tabela `QTableWidget`.



- W konstruktorze klasy onka zdefiniuj odpowiednią lokalizację domyślną projektu i kontroltek (`QLocale::setDefault()` i `QWidget::setLocale()`).
- Ustal początkową liczbę kolumn i wierszy.

```
ui->tableWidget->setColumnCount(2);
ui->tableWidget->setRowCount(0);
```

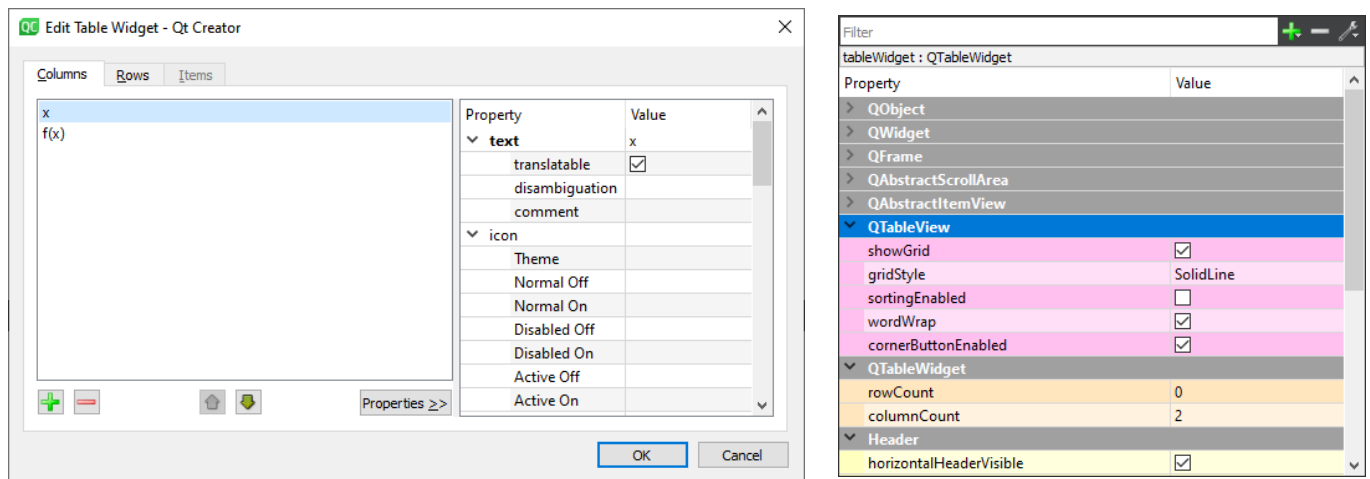
- Zdefiniuj nazwy kolumn tabeli typu **QTableWidget**.

```
QStringList nazwyKolumn;  
nazwyKolumn << "x" << "f(x)";  
ui->tableWidget->setHorizontalHeaderLabels(nazwyKolumn);
```

- Zmień konfigurację tabeli według propozycji (konstruktor klasy okna).

```
ui->tableWidget->setStyleSheet(  
    "QTableWidget{"  
        "background-color: #ffffff;"  
        "alternate-background-color: #cdebff;"  
    }"  
    "QTableCornerButton::section{"  
        "background-color: #f0f0f0;"  
    }"  
    "QHeaderView::section{"  
        "background-color: #f0f0f0;"  
        "border-top:0px;"  
        "border-left:0px;"  
        "border-right:1px solid #d8d8d8;"  
        "border-bottom: 1px solid #d8d8d8;"  
    }"  
);  
ui->tableWidget->setAlternatingRowColors(true);  
  
ui->tableWidget->setSelectionMode(QAbstractItemView::SingleSelection);  
ui->tableWidget->setSelectionBehavior(QAbstractItemView::SelectRows);  
  
ui->tableWidget->setCornerButtonEnabled(false);  
ui->tableWidget->setShowGrid(true);  
ui->tableWidget->setGridStyle(Qt::DotLine);  
  
ui->tableWidget->horizontalHeader()->setDefaultSectionSize(100);  
ui->tableWidget->horizontalHeader()->setStretchLastSection(true);
```

- Sprawdź znaczenie poszczególnych poleceń.
- Wprowadź własne zmiany w ustawienia kontrolki **QTableWidget**.
- Sprawdź możliwości zmian właściwości z poziomu IDE:
 - narzędzie **Edit Table Widget** po podwójnym kliknięciu w kontrolkę,
 - w inspektorze obiektów.



- W sekcji **private** klasy okna dodaj typ wyliczeniowy, ułatwiający odwołania do kolumn.

```
enum kolumny
{
    FUNARG, FUNWART
};
```

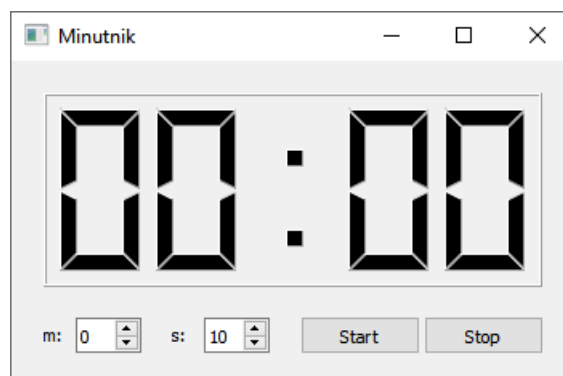
- Wyliczone wartości argumentu i funkcji można dodać do kontrolki za pomocą metody **setItem()**, która wymaga wskaźnika na obiekt typu **QTableWidgetItem** (nie jest to jedyny sposób).
- Wśród konstruktorów obiektu **QTableWidgetItem** jest taki, który wymaga jedynie stałej **const QString**.

```
ui->tableWidget->setItem(i, FUNARG, new QTableWidgetItem(...));
ui->tableWidget->setItem(i, FUNWART, new QTableWidgetItem(...));
```

3. Zadanie

Opracuj aplikację minutnik do odliczania czasu.

- Do budowy interfejsu użytkownika użyj kontroltek:
 - etykieta **QLabel**,
 - pole edycyjne **QSpinBox**
 - przycisk **QPushButton**,
 - wyświetlacz **QLCDNumber**.



- Do projektu dodaj odpowiednie pliki nagłówkowe.

```
#include <QTimer>
```

```
#include <QTime>
```

- Uzupełnij sekcję **private** definicji klasy okna o dodatkowe zmienne.

```
QTimer *minutnik;  
int sekundy;
```

- W konstruktorze klasy okna wykreuj nowy obiekt klasy **QTimer** i ustaw jego interwał czasu na jedną sekundę.

```
minutnik = new QTimer(this);  
minutnik->setInterval(1000);
```

- Slot obsługi przycisku **Start** powinien obliczać liczbę sekund (wewnętrzna reprezentacja czasu), którą należy dekrementować zgodnie ze zdarzeniem generowanym przez timer (zmienna **sekundy**). Wartości **s** i **m** (czas ustawiany przez użytkownika) należy odczytać z kontrolki **QSpinBox**.
- Dodatkowo slot powinien wyświetlać początkowy czas na wyświetlaczu i startować timer.
- Do operacji związanych z czasem można wykorzystać obiekt **QTime**.

```
sekundy = 60 * m + s;  
  
QTime czas(0, 0, 0);  
czas = czas.addSecs(sekundy);  
QString napis = czas.toString("mm:ss");  
  
ui->lcdNumber->display(napis);  
minutnik->start();
```

- Slot obsługi przycisku **Stop** powinien zatrzymywać timer i wpisywać zera na wyświetlaczu.

```
minutnik->stop();  
ui->lcdNumber->display("00:00");
```

- Slot obsługi sygnału emitowanego przez timer zdefiniuj w sekcji **private slots** definicji klasy okna.

```
private slots:  
    void minutnikSlot();
```

- Najedź na nazwę slotu, i z menu podręcznego wybierz **Refactor | Add definition in mainwindow.cpp**. Spowoduje to wygenerowanie odpowiedniej funkcji w kodzie źródłowym klasy okna.
- Uzupełnij metodę o dekrementację wartości zmiennej **sekundy**.
- Metoda powinna wyświetlać czas na wyświetlaczu LCD.
- Metoda musi zatrzymywać działanie timera po skończonym odliczaniu.

```
sekundy--;  
  
QTime czas(0, 0, 0);  
czas = czas.addSecs(sekundy);  
QString napis = czas.toString("mm:ss");  
  
ui->lcdNumber->display(napis);  
  
if(!sekundy)
```

```
{
    minutnik->stop();
}
```

- W konstruktorze klasy okna dodaj połączenie sygnału timera i slotu obsługi zdarzenia.

```
connect(minutnik, SIGNAL(timeout()), this, SLOT(minutnikSlot()));
```

- Skompiluj program i przetestuj jego działanie.

4. Zadanie

Zadanie z poprzedniego punktu uzupełnij o odtwarzanie sygnału dźwiękowego po zakończeniu odliczania (plik **rooster.wav** jest do pobrania ze strony przedmiotu).

- Do projektu dodaj plik z zasobami (*.qrc) i dodaj do niego prefix **dzwieki** oraz plik dźwiękowy.
- Do pliku projektu (*.pro) dodaj liniijkę (będzie można używać klas związanych z obsługą multimediiów).

```
QT += multimedia
```

- Do projektu dodaj plik nagłówkowy:

```
#include <QSound>
```

- Plik dźwiękowy można odtwarzać za pomocą metody **play()**. Uzupełnij metodę obsługi sygnału timera.

```
QSound::play(":dzwieki/rooster.vaw");
```

Zadania dodatkowe

1. Uzupełnij zadanie z punktu nr 1 o dodatkowe funkcje (minimum dwie).
2. Uzupełnij zadanie z punktu nr 2 o możliwość obliczenia potęgi kwadratowej lub pierwiastka kwadratowego wyznaczonych wartości funkcji. Wybór dodatkowej operacji powinien odbywać się za pomocą dodatkowych przełączników **QRadioButton** umieszczonych w osobnej ramce **QFrame**. Wyniki powinny być wyświetlone w dodatkowej kolumnie z odpowiednią nazwą kolumny korespondującą z dodatkową operacją.
3. W przykładzie z punktu nr 3 do zamiany czasu na łańcuch tekstowy był użyty obiekt **QTime** (rozwiązanie czasochłonne i nieoptymalne przy częstych operacjach związanych z wyświetlaniem czasu). To tego celu można użyć prostej zmiany typu całkowitego na łańcuch znaków i makra **QStringLiteral()**.

```
QString text = QStringLiteral("%1:%2").arg(m, 2, 10, QLatin1Char(':')).arg(s, 2, 10,
    QLatin1Char('0'));
```

Zmienne **m** i **s** to odpowiednio liczba minut i sekund. Zaproponuj metodę obliczania ich wartości mając całkowitą liczbę sekund (zmienna **sekundy**).

4. Opracuj aplikację *Stoper*. Aplikacja powinna umożliwiać pomiar czasu z dokładnością co do setnej części sekundy. Rozbuduj aplikację o możliwość mierzenia międzyczasów.