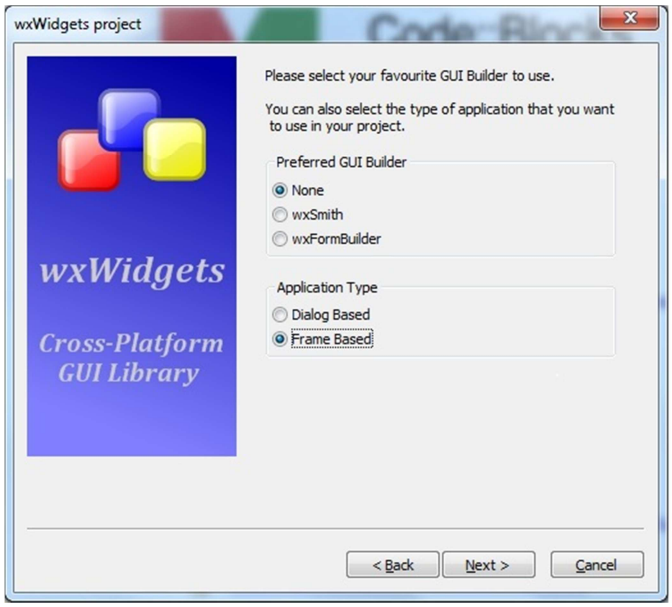
1. Wprowadzenie

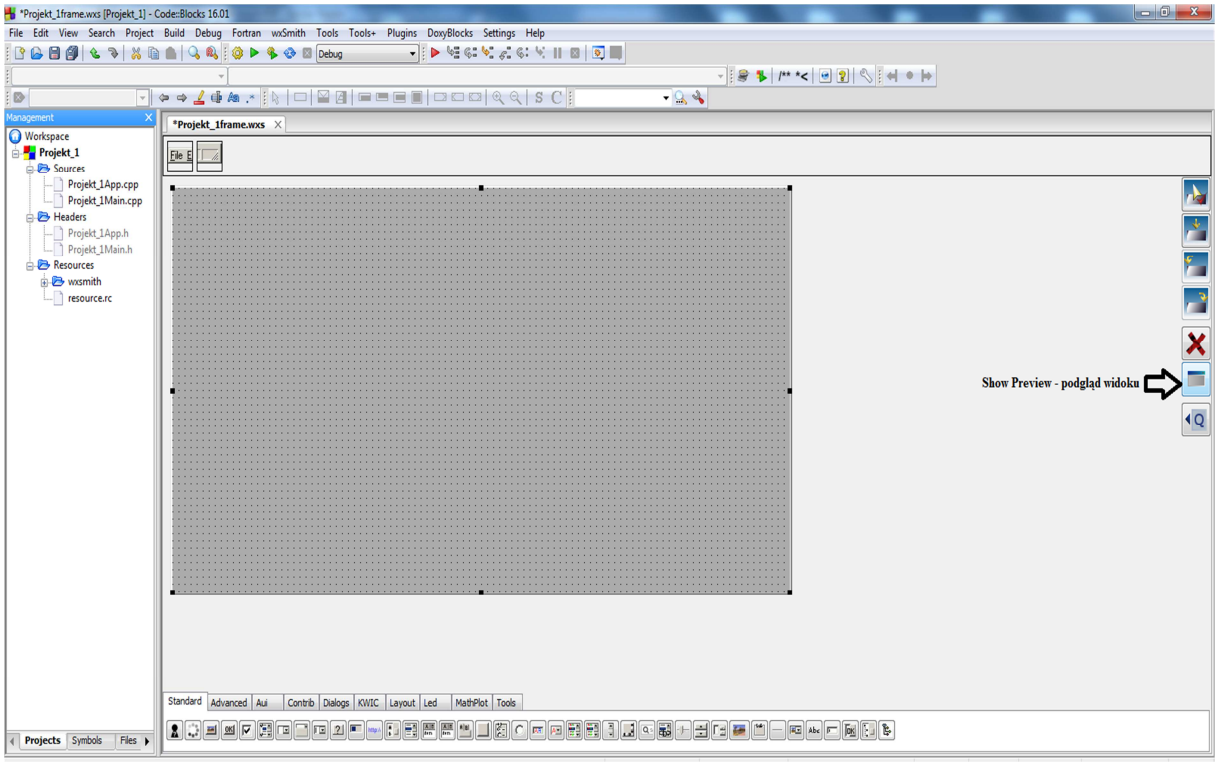
Dotychczas we wszystkich przykładach kod aplikacji umiejscowiony był w jednym pliku źródłowym. Podobnie, jak w wielu innych środowiskach programistycznych, również *Code Blocks* ułatwia programiście tworzenie zaawansowanych aplikacji. W przypadku bardziej rozbudowanych programów użytkownik może wybrać projekty typu: *wxWidgets Dialog Based* oraz *wxWidgets Frame Based* (w projektach tych aplikacja jest wyraźnie podzielona na kilka plików)[1].



Rys. 1. Różne typy projektów *wxWidgets*

Korzyścią w obu przypadkach jest automatyczne generowanie przez środowisko komponentów niezbędnych do budowy aplikacji *wxWidgets* (przyciski, pola tekstowe, listy rozwijalne). Oprócz kontrolek użytkownik ma do dyspozycji również tabele zdarzeń, pola wyboru, przyciski opcji i wiele innych. Dodatkowym ułatwieniem jest możliwość podglądu widoku budowanej aplikacji (*Show Preview*), dostępna z poziomu panelu komponentów aplikacji (użytkownik może na bieżąco obserwować postępy działania aplikacji).

[1] Ponieważ projekty typu: *wxWidgets Dialog Based* oraz *wxWidgets Frame Based* zawierają kilka plików, drzewo projektu będzie zawierało ich nazwy plików.



Rys. 2. Piktogram *Show Preview* do podglądu widoku aktualnie tworzonego projektu

Analizując kody źródłowe wielu aplikacji można zauważyć, że metody klas *wxWidgets* (dotyczy to zwłaszcza konstruktorów) często posiadają podobne parametry. Najważniejsze z nich to:

Id – stanowi identyfikator tworzonego obiektu (dla konstruktora);

parent – stanowi identyfikator obiektu np. okna będącego nadrzędnym w stosunku do tworzonego obiektu;

size – określa wymiary okna w punktach;

pos – określa współrzędne lewego górnego okna obiektu w stosunku do obiektu nadrzędnego;

style – opisuje wygląd oraz zachowanie danego obiektu;

name – nazwa okna wyrażona jako łańcuch znakowy.

Jednym z podstawowych elementów każdej biblioteki służącej do tworzenia graficznego interfejsu użytkownika (*ang. GUI*) są zestawy komponentów (*ang. components*) nazywanych też kontrolkami (*ang. controls*). Biblioteka *wxWidgets* również posiada taki zestaw predefiniowanych kontrolek umożliwiających w przystępny sposób budowanie graficznego interfejsu użytkownika. Komponenty dostępne w bibliotece *wxWidgets* można podzielić na dwa rodzaje:

- niestatyczne komponenty/kontrolki (*ang. non static controls*),
- statyczne komponenty/kontrolki (*ang. static controls*).

Różnica pomiędzy tymi dwoma rodzajami komponentów polega na różnej obsłudze zdarzeń (*ang. events*) oraz docelowym przeznaczeniu. Niestatyczne komponenty pozwalają na interakcję z użytkownikiem poprzez obsługę zdarzeń generowanych przez użytkownika. Natomiast komponenty statyczne służą jedynie do prezentacji określonych danych.

Programowanie interfejsu graficznego użytkownika polega na odpowiednim doborze oraz ułożeniu kontrolek na oknie roboczym, oknie dialogowym lub panelu i odpowiednim ich oprogramowaniu poprzez dodanie obsługi zdarzeń generowanych przez użytkownika z wykorzystaniem urządzeń wejściowych tj. myszy, klawiatury, joysticka itd.

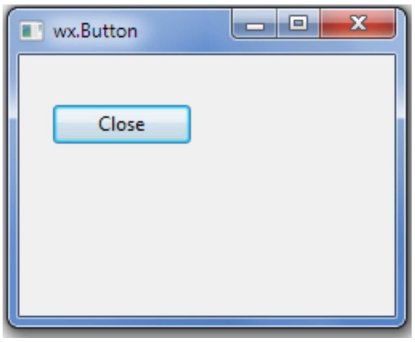
1.1 Niestatyczne komponenty

Niestatyczne komponenty/kontrolki pozwalają na interakcję z użytkownikiem poprzez obsługę zdarzeń generowanych z wykorzystaniem urządzeń wejściowych np. myszy lub klawiatury. Poniżej została przedstawiona lista dostępnych komponentów niestatycznych w bibliotece *wxWidgets*. Ze względu na dużą liczbę tych komponentów omówione zostaną jedynie wybrane kontrolki. Więcej informacji o komponentach/kontrolkach można znaleźć w dokumentacji biblioteki. Listę komponentów zamieszczono poniżej:

wxButton, wxBitmapButton, wxComboBox, wxCheckBox, wxListBox, wxCheckListBox,

wxRadioBox, wxRadioButton, wxScrollBar, wxSpinButton, wxSpinCtrl, wxSlider,

wxTextCtrl, wxToggleButton



wxButton jest to jeden z podstawowych komponentów, najczęściej wykorzystywany w przypadku graficznego interfejsu użytkownika. Jak sama nazwa wskazuje, kontrolka ta opisuje przycisk, którego podstawową funkcjonalnością jest reagowanie na kliknięcie

kursorem myszy lub klawiszem klawiatury na dany komponent. Komponent ten może być umieszczony na innym elemencie, jak np. okno, okno dialogowe lub panel.

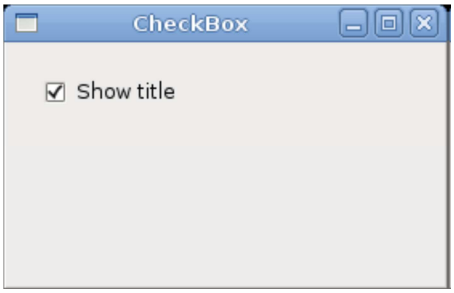
```
#include "wx/button.h"

wxButton* button = new wxButton(panel, wxID_OK, wxT("OK"),

wxPoint(10, 10), wxDefaultSize);
```

Konstruktor klasy *wxButton* przyjmuje kilka parametrów m.in. wskaźnik na okno/panel rodzica, na którym ma być umieszczony przycisk, rodzaj przycisku tzw. *id przycisku*, tekst umieszczony na przycisku (*ang. text label*), położenie przycisku oraz jego rozmiar. Biblioteka *wxWidgets* dostarcza szereg przedefiniowanych rodzajów przycisków, które opisane są za pomocą stałych rozpoczynających się od *wxID_* np. *wxID_ADD*, *wxID_APPLY*, *wxID_BOLD*, *wxID_CANCEL*, *wxID_CLEAR*, *wxID_CLOSE*, *wxID_COPY*, *wxID_CUT*, *wxID_DELETE*, *wxID_FIND*, *wxID_REPLACE* itp. (więcej można znaleźć w dokumentacji biblioteki). Z każdym predefiniowanym rodzajem przycisku powiązany jest również określony tekst umieszczany na przycisku.

Konstruktor klasy *wxButton* pozwala na określenie stylu przycisku. Podobnie jak w przypadku rodzaju przycisku, w bibliotece *wxWidgets* mamy szereg predefiniowanych stałych określających styl przycisku rozpoczynających się od *wxBU_* np. *wxBU_LEFT*, *wxBU_TOP*, *wxBU_RIGHT*, *wxBU_BOTTOM*, *wxBU_EXACTFIT*, *wxNO_BORDER*.



wxCheckBox jest to komponent posiadający dwa stany (włączenia/wyłączenia), które odzwierciedlone są na komponencie za pomocą znacznika *tick* lub *krzyżyka*. Może posiadać również tekst (*ang. text label*) umieszczony z lewej lub prawej strony komponentu.

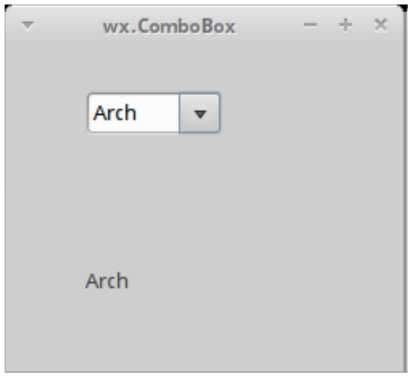
```
#include "wx/checkbox.h"

wxCheckBox* checkbox = new wxCheckBox(panel, ID_CHECKBOX,
wxT("&Check me"), wxDefaultPosition, wxDefaultSize);

checkBox->SetValue(true);
```

Konstruktor klasy *wxCheckBox* przyjmuje kilka parametrów m.in. wskaźnik na okno/panel rodzica, na którym ma być umieszczony komponent, identyfikator komponentu, tekst umieszczony obok komponentu (*ang. text label*), położenie komponentu oraz jego rozmiar.

Ponadto konstruktor klasy *wxCheckBox* pozwala na określenie stylu komponentu. W bibliotece *wxWidgets* mamy szereg predefiniowanych stałych określających styl komponentu rozpoczynających się od *wxCHK_* np *wxCHK_2STATE*, *wxCHK_3STATE*, *wxCHK_ALLOW_3RD_STATE_FOR_USER*, *wxALIGN_RIGHT*.



wxComboBox jest to komponent tzw. *listy wyboru*, który umożliwia w rozwijanej liście zaprezentowanie różnych opcji wyboru. Wszystkie opcje w początkowej fazie działania komponentu są ukryte przed użytkownikiem. Kliknięcie przez użytkownika przyciskiem myszy na ikonę strzałki po prawej stronie komponentu spowoduje rozwinięcie listy wyboru.

```
#include "wx/combobox.h"

wxArrayString strings;
strings.Add(wxT("Apple"));
strings.Add(wxT("Orange"));
strings.Add(wxT("Pear"));
strings.Add(wxT("Grapefruit"));

wxComboBox* combo = new wxComboBox(panel, ID_COMBOBOX,
wxT("Apple"), wxDefaultPosition, wxDefaultSize,
strings, wxCB_DROPDOWN);
```

Konstruktor klasy *wxComboBox* przyjmuje kilka parametrów m.in. wskaźnik na okno/panel rodzica, na którym ma być umieszczony komponent, identyfikator komponentu, początkową opcję wyboru, położenie komponentu oraz jego rozmiar.

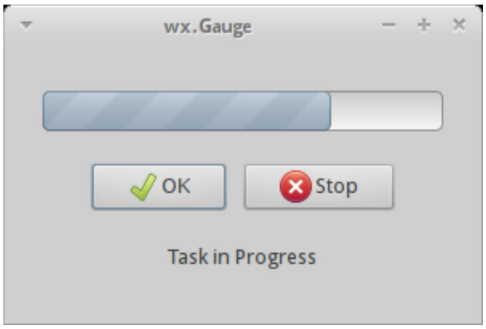
Ponadto konstruktor klasy *wxComboBox* pozwala na określenie stylu komponentu. W bibliotece *wxWidgets* mamy szereg predefiniowanych stałych określających styl komponentu rozpoczynających się od *wxCB_* np *wxCB_SIMPLE*, *wxCB_DROPDOWN*, *wxCB_READONLY*, *wxCB_SORT*.

Analogicznie do przedstawionych powyżej wybranych komponentów tworzone są inne niestatyczne kontrolki dostępne w bibliotece *wxWidget*s, więcej informacji na ten temat można odnaleźć w dokumentacji biblioteki.

1.2 Statyczne komponenty

Statyczne komponenty/kontrolki służą głównie do prezentacji danych wyjściowych/informacyjnych użytkownikowi danego interfejsu graficznego. Podobnie jak niestatyczne komponenty mogą częściowo reagować na wybrane zdarzenia np. najechanie kursorem myszy na dany komponent. Jednakże ich głównym zadaniem i przeznaczeniem jest prezentacja określonych danych. Poniżej została przedstawiona lista dostępnych komponentów statycznych w bibliotece *wxWidgets*:

wxGauge, *wxStaticText*, *wxStaticBitmap*, *wxStaticBox*, *wxStaticLine*



wxGauge jest to komponent statyczny reprezentujący tzw. pasek postępu (*ang. progress bar*). Głównym zadaniem komponentu jest informowanie użytkownika o postępie danego zadania wykonywanego przez aplikację.

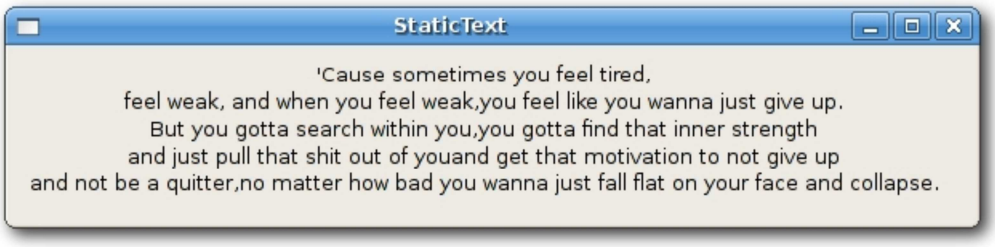
```
#include "wx/gauge.h"

wxGauge* gauge = new wxGauge(panel, ID_GAUGE,
200, wxDefaultPosition, wxDefaultSize, wxGA_HORIZONTAL);

gauge->SetValue(50);
```

Konstruktor klasy *wxGauge* przyjmuje kilka parametrów m.in. wskaźnik na okno/panel rodzica, na którym ma być umieszczony komponent, identyfikator komponentu, zakres paska postępu, położenie komponentu oraz jego rozmiar.

Ponadto konstruktor klasy *wxGauge* pozwala na określenie stylu komponentu. W bibliotece *wxWidgets* mamy szereg predefiniowanych stałych określających styl komponentu rozpoczynających się od *wxGA_* np. *wxGA_HORIZONTAL*, *wxGA_VERTICAL*, *wxGA_SMOOTH*.



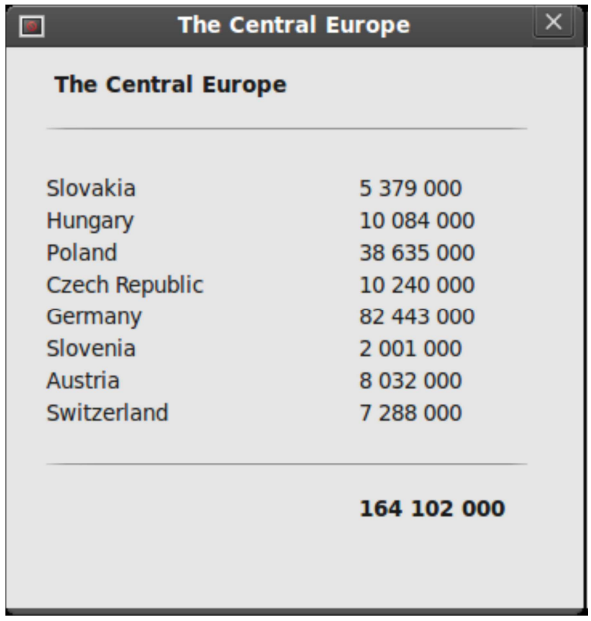
wxStaticText jest to komponent statyczny pozwalający na umieszczanie tekstu statycznego w dowolnym miejscu okna lub panelu.

```
#include "wx/stattext.h"

wxStaticText* staticText = new wxStaticText(panel,
wxID_STATIC, wxT("This is my &static label"),
wxDefaultPosition, wxDefaultSize, wxALIGN_LEFT);
```

Konstruktor klasy *wxStaticText* przyjmuje kilka parametrów m.in. wskaźnik na okno/panel rodzica, na którym ma być umieszczony komponent, identyfikator komponentu, tekst do umieszczenie w komponencie, położenie komponentu oraz jego rozmiar.

Ponadto konstruktor klasy *wxStaticText* pozwala na określenie stylu komponentu oraz tekstu zawartego w tym komponencie np. *wxALIGN_LEFT*, *wxALIGN_RIGHT*, *wxALIGN_CENTRE*, *wxALIGN_CENTER*, *wxST_NO_AUTORESIZE*.



wxStaticLine jest to najprostszy komponent styczny reprezentujący linie oddzielającą.

```
#include "wx/statline.h"

wxStaticLine* staticLine = new wxStaticLine(panel,
wxID_STATIC, wxDefaultPosition, wxSize(150, -1),
wxLI_HORIZONTAL);
```

Konstruktor klasy *wxStaticLine* przyjmuje kilka parametrów m.in. wskaźnik na okno/panel rodzica, na którym ma być umieszczony komponent, identyfikator komponentu, położenie komponentu oraz jego rozmiar.

Konstruktor klasy *wxStaticLine* pozwala również na określenie stylu komponentu (rodzaj linii) np. *wxLI_HORIZONTAL, wxLI_VERTICAL.*

Analogicznie do przedstawionych powyżej wybranych komponentów tworzone są inne statyczne kontrolki dostępne w bibliotece *wxWidget*s, więcej informacji na ten temat można odnaleźć w dokumentacji biblioteki.

2. Parametry stylu aplikacji

Jak wcześniej zauważono, okno aplikacji może stanowić obiekt dwóch klas (*Frame*, *Dialog*). Obie wspomniane klasy dziedziczą po klasie ogólnej opisującej okno *wxWindow*. Na ogół okno *wxDialog* umożliwia wybór opcji aplikacji (podstawowy projekt może zawierać tylko jedno okno dialogowe), natomiast okno *wxFrame* służy do tworzenia aplikacji (zwykle w oparciu o klasę potomną obiektu *wxFrame*).

Od ustawień parametru *style* zależy zaś funkcjonalność okna oraz jego widok. Podstawowe, przykładowe wartości parametru *style* oraz ich znaczenie zaprezentowano w Tabeli 1. W przypadku braku wyboru stylu automatycznie zostanie wybrany styl domyślny.

Tabela 1. Podstawowe wartości parametru style oraz ich znaczenie

Wartość parametru *style*	**Oddziaływanie**
wxICONIZE	dotyczy okna zminimalizowanego
wxCAPTION	dotyczy okna, które zawiera pasek tytułowy
wxMINIMIZE_BOX	dotyczy wyświetlenia w pasku okna przycisku *Minimalizuj*
wxMAXIMIZE_BOX	dotyczy wyświetlenia w pasku okna przycisku *Maksymalizuj*
wxCLOSE_BOX	dotyczy wyświetlenia w pasku okna przycisku *Zamknij*
wxSYSTEM_MENU	dotyczy ikony aplikacji z *menu* okna w lewym górnym rogu
wxRESIZE_BORDER	dotyczy włączenia możliwości modyfikacji wymiarów okna w trakcie działania aplikacji
wxFRAME_NO_TASKBAR	dotyczy wyłączenia możliwości pojawienia się okna na pasku zadań

Do wyboru czcionki służą funkcje *SetFont()* oraz *SetOwnFont()*, które odpowiednio ustawiają czcionkę dla danego okna/kontrolki i wszystkich obiektów potomnych lub wyłącznie dla danego obiektu. Obie funkcje są zdefiniowane w klasie *wxWindow*, co przekłada się na ich dostępność w większości kontrolek. Z punktu widzenia użytkownika niezmiernie ważne są również panele, na których można umieszczać kontrolki (alternatywa do bezpośredniego wykorzystania okna *wxFrame*).

3. Metody stosowane w aplikacjach

Podczas tworzenia aplikacji graficznych do wprowadzania danych można wykorzystać okna tekstowe. Zawartość kontrolki odczytana przez metodę *GetValue()* jest typu *wxString*. Poniżej przedstawiono inne przydatne metody (proszę zapoznać się z ich zastosowaniem w praktyce; tekst jest ciągiem znaków wxString): *GetLabel(); SetValue(text); SetLabel(text); SetLabelText(text)* i wiele innych metod. Jeśli danymi mają być liczby, musimy je przekonwertować na postać liczbową.

Odpowiednie metody konwersji zawarte są w klasie *wxString* (są przedstawione poniżej – zobacz Tabela 2).

Tabela 2. Metody konwersji i ich działanie

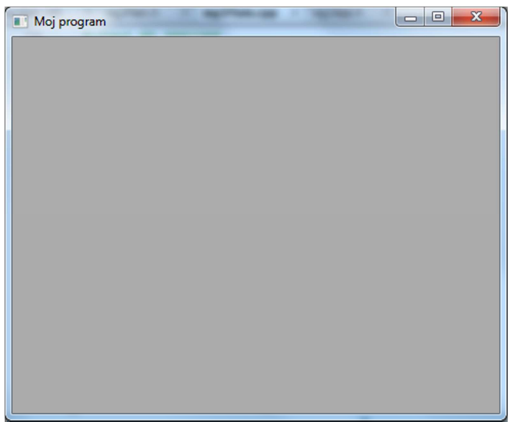
Metody	Działanie
*ToDouble (double * number)*	konwertuje łańcuch znakowy *wxString* na liczbę zmiennoprzecinkową typu double; wynik zapisywany jest w zmiennej liczba (*number*)
*ToLong (long int *number, int base_argument=10)*	konwertuje łańcuch znakowy *wxString* na liczbę całkowitą typu long int; wynik zapisywany jest w zmiennej liczba (*number*); *base_argument* wskazuje system, w którym zapisywana jest liczba (domyślnie jest to system dziesiętny)
*ToULong (long int *number, int base_argument=10)*	konwertuje tak samo jak metoda *ToLong()*, ale wynikiem jest liczba typu unsigned long int (czyli bez znaku).

Wszystkie opisane metody zwracają *true*, jeśli konwersja zakończyła się sukcesem i *false*, jeśli pojawił się błąd (niepowodzenie). Ponieważ wynik konwersji jest zwracany jako parametr funkcji, więc używany jest typ wskaźnikowy.

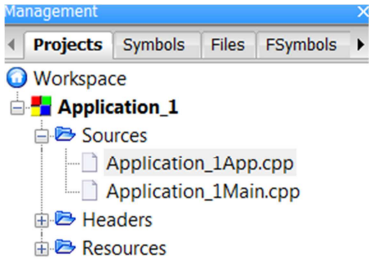
O ile do konwersji *wxStrings* na liczby stosuje się metody wskazane w Tabeli 2, o tyle do konwersji liczb na *wxString* powszechnie stosuje się metodę *Printf()* (np. *Printf(wxT("%f")*, *x*y*). Możliwe jest również użycie operatora << do wprowadzenia np. wyniku operacji matematycznych do wxString, np: *product << x*y* (dla mnożenia 2 liczb), gdzie: *wxString product = wxT("").*

4. Zadania do wykonania

1. Zmodyfikować program z poprzednich zajęć, dodając definicję klasy potomnej *wxFrame* o nazwie *Window* (w jej ciele powinna znaleźć się wyłącznie deklaracja konstruktora). Napisać definicję dla tego konstruktora oraz zmienić metodę *OnInit()* tak, aby zamiast obiektu klasy *wxFrame* tworzony był obiekt klasy *Window*. Po uruchomieniu programu okno powinno wyglądać dokładnie tak samo jak w przykładzie nr 3.



Wskazówka:



```
#include "wx/wxprec.h"
#ifndef WX_PRECOMP
#include "wx/wx.h"
#endif
class Application_1: public wxApp
{   public:
   virtual bool OnInit();};

IMPLEMENT_APP(Application_1);
class Window: public wxFrame
{public:
Window (const wxString& title,const wxPoint& pos,const wxSize&
size, long style = wxDEFAULT_FRAME_STYLE);};
Window::Window(const wxString& title,const wxPoint& pos, const
wxSize& size, long style): wxFrame(NULL,-
1,title,pos,size,style) {}
bool Application_1::OnInit(){

Window* MainWindow = new Window(_T("My application"),
wxPoint(50,50),wxSize(200,200)); MainWindow->Show(TRUE); return
TRUE;
}
```

2. Napisać program kalkulator jako typ projektu *wxWidgets Frame Based.* Program powinien umożliwić przeprowadzenie podstawowych operacji matematycznych na dwóch dowolnych liczbach (*x* i *y*). Jako podstawowe operacje matematyczne rozumie się: dodawanie, odejmowanie, mnożenie oraz dzielenie pierwszej liczby (*x*) przez drugą (*y*).

Wskazówka: zobacz przykładowy fragment kodu przedstawiający interakcję z użytkownikiem poprzez obsługę zdarzenia polegającego na kliknięciu kursorem myszy na komponent *wxButton* (program zwraca wynik podstawowych operacji matematycznych: *total* (suma), *difference* (różnica), *product* (iloczyn) oraz *quotient* (iloraz) dla 2 liczb wprowadzonych przez użytkownika):

```
void KalkFrame::OnButton1Click(wxCommandEvent& event)
{{ double x=0, y=0;
wxString total = wxT("");
wxString difference = wxT("");
wxString product = wxT("");
wxString quotient = wxT("");
if (TextCtrl1->GetValue().ToDouble((&x))
    and TextCtrl2->GetValue().ToDouble((&y)))
{
    total << (x+y);
    difference << (x-y);
    product << (x*y);
    quotient << (x/y);
    TextCtrl3->SetValue(total);
    TextCtrl4->SetValue(difference);
    TextCtrl5->SetValue(quotient);
    TextCtrl6->SetValue(product);
}
}
}
```

3. Zapoznać się z projektowaniem i tworzeniem interfejsu graficznego, na którym zostaną zaprezentowane różne rodzaje komponentów: *wxButton*, *wxBitmapButton*, *wxComboBox*, *wxListBox*, *wxCheckListBox*, *wxRadioBox*, *wxRadioButton* oraz *wxGauge*. Przetestować i zaprezentować różne ustawienia poszczególnych komponentów w oparciu o proste implementacje prezentujące ich działanie.
4. Zapoznać się z projektowaniem i tworzeniem interfejsu graficznego dla formularza pobierającego podstawowe dane osobowe studenta tj. imię, nazwisko, wiek, rok studiów, grupa dziekańska, nr indeksu. Do utworzenia takiego interfejsu warto użyć wielu poznanych komponentów m.in. *wxButton*, *wxTextCtrl*, *wxSpinCtrl*, *wxSpinButton* itp.