1. Wprowadzenie

Na poprzednich zajęciach laboratoryjnych studenci użyli środowiska *Code::Blocks* m.in. do budowy prostej aplikacji typu kalkulator, wykorzystując do tego celu wieloplatformową bibliotekę *wxWidgets* oraz system *Windows*. W ramach dzisiejszych zajęć studenci pogłębią znajomość biblioteki *wxWidgets* o zdarzenia i sizery oraz zostaną zapoznani z kompilacją projektu utworzonego w systemie *Linux* w dystrybucji *Debian* lub *Ubuntu* przy pomocy odpowiednio skonstruowanego pliku *Makefile*.

2. Zdarzenia (*events*)

W praktyce, jedną z integralnych i najważniejszych właściwości aplikacji graficznych jest interakcja z użytkownikiem. Każda aplikacja graficzna – niezależnie od architektury sprzętowej czy też systemu operacyjnego, na którym jest uruchomiona – odbiera szereg wejściowych sygnałów od użytkownika. Sygnały te są generowane za pomocą urządzeń wejściowych tj. myszy, klawiatury, joysticka itp.

Podstawowym elementem każdej biblioteki służącej do tworzenia graficznego interfejsu użytkownika jest odpowiednie interpretowanie i obsługa sygnałów wejściowych nazywanych potocznie zdarzeniami *(ang events).* Biblioteka *wxWidgets* posiada zestaw predefiniowanych makr preprocesora oraz funkcji umożliwiających w przystępny sposób obsługę zdarzeń. Zdarzenia w bibliotece *wxWidgets* podzielne są na trzy kategorie, w zależności od urządzenia generującego dane zdarzenie:

- zdarzenia generowane przez mysz (*ang. mouse events*),
- zdarzenia generowane przez klawiaturę (*ang. keyboard events*),
- zdarzenia generowane przez joystick (*ang. joystick events*).

W instrukcji zostanie omówiona jedynie obsługa zdarzeń generowanych z wykorzystaniem myszki oraz klawiatury, ze względu na bardzo szerokie i powszechne wykorzystanie tych urządzeń.

2.1 Obsługa zdarzeń generowanych z wykorzystaniem myszki

W przypadku biblioteki *wxWidgets* sygnały/zdarzenia generowane z wykorzystywaniem myszki komputerowej możemy podzielić na dwie kategorie:

- sygnały/zdarzenia bezpośrednie,
- sygnały/zdarzenia pośrednie.

Zdarzenia bezpośrednie generowane są podczas użytkowania myszki komputerowej poprzez określone pojedyncze kliknięcie (lub zwolnienie przycisku), podwójne kliknięcie klawisza tj. lewy klawisz myszy, prawy klawisz myszy, przekręcanie *scroll* (góra/dół) itp. Sygnały bezpośrednie otrzymywane z myszki komputerowej generują specjalne zdarzenia klasy *wxMouseEvents*. Biblioteka *wxWidgets* dostarcza specjalne makra preprocesora za pomocą których do każdego rodzaju zdarzenia wygenerowanego przez myszkę komputerową można podpiąć odpowiednią (własną) funkcję obsługi danego zdarzenia. Poniżej została przedstawiona lista makr preprocesora służących do podpięcia funkcji obsługi zdarzeń bezpośrednich.

1. *EVT_LEFT_DOWN(func)*
2. *EVT_LEFT_UP(func)*
3. *EVT_LEFT_DCLICK(func)*
4. *EVT_MIDDLE_DOWN(func)*
5. *EVT_MIDDLE_UP(func)*
6. *EVT_MIDDLE_DCLICK(func)*
7. *EVT_RIGHT_DOWN(func)*
8. *EVT_RIGHT_UP(func)*
9. *EVT_RIGHT_DCLICK(func)*
10. *EVT_MOTION(func)*
11. *EVT_ENTER_WINDOW(func)*
12. *EVT_LEAVE_WINDOW(func)*
13. *EVT_MOUSEWHEEL(func)*
14. *EVT_MOUSE_EVENTS(func)*

Zdarzenia pośrednie również pochodzą od myszki komputerowej, generowane są jednak na określonym komponencie (tj. z wykorzystaniem określonego komponentu) np. *wxButton*. W takim przypadku dany komponent generuje zdarzenie pośrednie klasy *wxCommandEvent* w zależności od akcji wykonanej na komponencie, które powiązane jest z określonym zdarzeniem bezpośrednim. Do obsługi zdarzeń pośrednich wykorzystywane są

inne makra preprocesora. Poniżej została przedstawiona lista (częściowa) makr preprocesora służących do podpięcia funkcji obsługi zdarzeń pośrednich dla określonego komponentu.

1. *EVT_BUTTON(id, func)*
2. *EVT_CHOICE(id, func)*
3. *EVT_TEXT(id, func)*
4. *EVT_COMBOBOX(id, func)*
5. *EVT_CHECKBOX(id, func)*
6. *EVT_LISTBOX(id, func)*
7. *EVT_LISTBOX_DCLICK(id, func)*
8. *EVT_CHECKLISTBOX (id, func)*
9. *EVT_RADIOBOX(id, func)*
10. *EVT_RADIOBUTTON(id, func)*
11. *EVT_SPIN(id, func)*
12. *EVT_SPIN_UP(id, func)*
13. *EVT_SPIN_DOWN(id, func)*
14. *EVT_SPINCTRL(id, func)*
15. *EVT_SLIDER(id, func)*
16. *EVT_TEXT(id, func)*
17. *EVT_TEXT_ENTER(id, func)*
18. *EVT_TEXT_MAXLEN(id, func)*
19. *EVT_TOGGLEBUTTON(id, func)*

Odpowiednia obsługa zdarzeń stanowi kolejny istotny element programowania interfejsu graficznego. W bibliotece *wxWidgets* wyróżniamy dwa sposoby obsługi zdarzeń. Pierwszy sposób realizowany jest z wykorzystaniem tzw. tablicy zdarzeń (*ang. table events*), z kolei drugi sposób odbywa się w oparciu o metodę *Connect()*. Warto podkreślić, że zaleca się używać metody *Bind()* zamiast metody *Connect()*.

Różnica pomiędzy tymi dwoma sposobami polega na połączeniu zdarzeń z funkcjami obsługi. W przypadku tablicy zdarzeń połączenia te są statyczne (podczas kompilacji przypisywana jest funkcja obsługi do zdarzenia) i nie mogą zostać zmienione w czasie działania aplikacji, natomiast w przypadku użycia metody *Connect()* lub *Bind()* połączenia pomiędzy zdarzeniami i funkcjami obsługi są dynamiczne (można przypinać/odpinać określone funkcje obsługi od/do zdarzeń), mogą się one zmieniać w trakcie działania aplikacji w zależności od potrzeb, spełnienia określonego warunku itp.

Poniżej zamieszczono przykład użycia tablic zdarzeń (z oficjalnej strony *wxWidgets*):

```
/* button.h */
#include <wx/wx.h>

class MyButton : public wxFrame {
public:
    MyButton(const wxString& title);
    void OnQuit(wxCommandEvent& event);
private:
    DECLARE_EVENT_TABLE()
};
```

```
/* button.cpp */
#include "button.h"

MyButton::MyButton(const wxString& title) :wxFrame(NULL,
wxID_ANY, title, wxDefaultPosition, wxSize(270, 150)) {

  wxPanel *panel = new wxPanel(this, wxID_ANY);
   wxButton *button = new wxButton(panel, wxID_EXIT,
wxT("Quit"), wxPoint(20, 20));
  Centre();
}

void MyButton::OnQuit(wxCommandEvent& WXUNUSED(event)) {
    Close(true);
}

BEGIN_EVENT_TABLE(MyButton, wxFrame)
    EVT_BUTTON(wxID_EXIT,  MyButton::OnQuit)
END_EVENT_TABLE()
```

```
/* main.h */
#include <wx/wx.h>

class MyApp : public wxApp {
  public:
    virtual bool OnInit();
};
```

```
/* main.cpp */
#include "main.h"
#include "button.h"

IMPLEMENT_APP(MyApp)
bool MyApp::OnInit() {
    MyButton *button = new MyButton(wxT("Button"));
    button->Show(true);
    return true;
}
```

Przykład użycia metody *Connect()* (z oficjalnej strony *wxWidgets*):

```
/* move.h */
#include <wx/wx.h>

class Move : public wxFrame {
public:
  Move(const wxString& title);
  void OnMove(wxMoveEvent & event);
  wxStaticText *st1;
  wxStaticText *st2;
};
```

```
/* move.cpp */
#include "move.h"

Move::Move(const wxString& title)
          : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition,
wxSize(250, 130)) {
  wxPanel *panel = new wxPanel(this, -1);
  st1 = new wxStaticText(panel, -1, wxT(""), wxPoint(10, 10));
  st2 = new wxStaticText(panel, -1, wxT(""), wxPoint(10, 30));
  Connect(wxEVT_MOVE, wxMoveEventHandler(Move::OnMove));
  Centre();
}

void Move::OnMove(wxMoveEvent& event) {
  wxPoint size = event.GetPosition();
  st1->SetLabel(wxString::Format(wxT("x: %d"), size.x ));
  st2->SetLabel(wxString::Format(wxT("y: %d"), size.y ));
}
```

```
/* main.h */
#include <wx/wx.h>

class MyApp : public wxApp {
  public:
    virtual bool OnInit();
};
```

```
/* main.cpp */
#include "main.h"
#include "move.h"

IMPLEMENT_APP(MyApp)
bool MyApp::OnInit() {
    Move *move = new Move(wxT("Move event"));
    move->Show(true);
    return true;
}
```

2.2 Obsługa zdarzeń generowanych z wykorzystaniem klawiatury

Podobnie jak w przypadku myszy komputerowej, sygnały pochodzące z klawiatury opisywane są przez specjalną klasę zdarzenia *wxKeyEvent*. Zdarzenia generowane z wykorzystaniem klawiatury w bibliotece *wxWidgets* można podzielić na dwie kategorie:

- zdarzenia nietłumaczone (*ang. untranslated events*),
- zdarzenia tłumaczone (*ang. translated events*).

Różnica pomiędzy tymi zdarzeniami polega na tym, że zdarzenia nietłumaczone generowane są przez wszystkie klawisze funkcyjne tj. Enter, Delete, Spacja, Shift, Alt, Ctrl, Home, End, F1, F2, F3 itp. Można je podzielić dodatkowo na zdarzenia generowane przez wciśnięcie klawisza (*ang. key down*) lub puszczenie klawisza (*ang. key up*). Natomiast zdarzenia tłumaczone, nazywane też znakowymi (*ang. characters events*), stanowią zdarzenia generowane przez klawisze liter, cyfr itp. Biblioteka *wxWidgets* dostarcza specjalne makra preprocesora za pomocą których do każdego rodzaju zdarzenia wygenerowanego przez klawiaturę można podpiąć odpowiednią (własną) funkcję obsługi danego zdarzenia. Poniżej została przedstawiona lista makr preprocesora służących do podpięcia funkcji obsługi zdarzeń.

1. *EVT_KEY_DOWN(func)*
2. *EVT_KEY_UP(func)*
3. *EVT_CHAR(func)*

W klasie *wxKeyEvent* dostępne są specjalne metody *GetKeyCode()* oraz *GetUnicodeKeyCode()*, za pomocą których można sprawdzić który klawisz został wciśnięty. Dla wszystkich zdarzeń nietłumaczonych w bibliotece *wxWidgets* istnieją predefiniowane stałe dla określonych klawiszy zaczynające się od *WXK_* np. *WXK_BACK, WXK_RIGHT*, *WXK_TAB*, *WXK_DOWN*, *WXK_RETURN*, *WXK_SELECT*, *WXK_ESCAPE*, *WXK_PRINT*, *WXK_SPACE*, *WXK_EXECUTE*, *WXK_DELETE* (całą listę dostępnych stałych można znaleźć w dokumentacji biblioteki).

Obsługa zdarzeń pochodzących z klawiatury odbywa się analogicznie jak w przypadku myszki komputerowej – tj. z wykorzystaniem tablicy zdarzeń lub metody *Connect()*/*Bind()*.

Przykład użycia:

```
BEGIN_EVENT_TABLE(MyHandler, wxFrame)
      EVT_CHAR(MyHandler::OnChar)
END_EVENT_TABLE()

void MyHandler::OnChar(wxKeyEvent& event) {
      switch (event.GetKeyCode()) {
          case WXK_LEFT:
          case WXK_RIGHT:
              ... move cursor ...
              break;
          case WXK_F1:
              ... give help ...
              break;
      }
}
```

W bibliotece *wxWidgets* możliwa jest również obsługa skrótów klawiszowych np. Ctrl+O, Alt+K, Shift+Ctrl+V itd. Skróty klawiszowe zazwyczaj powiązane są z paskami narzędziowymi (*ang. toolbars*) lub menu. Skróty klawiszowe opisywane są za pomocą klasy *wxAcceleratorEntry*, na podstawie której tworzy się tablicę obiektów i umieszcza się ją w obiekcie *wxAcceleratorTable*, który jest ostatecznie przypisywany do okna/panelu.

```
wxAcceleratorEntry entries[4];

entries[0].Set(wxACCEL_CTRL, (int) 'N', wxID_NEW);
entries[1].Set(wxACCEL_CTRL, (int) 'X', wxID_EXIT);
entries[2].Set(wxACCEL_SHIFT, (int) 'A', wxID_ABOUT);
entries[3].Set(wxACCEL_NORMAL, WXK_DELETE, wxID_CUT);

wxAcceleratorTable accel(4, entries);
frame->SetAcceleratorTable(accel);
```

3. Budowa pliku *Makefile*

W przypadku budowy aplikacji w środowisku *Code::Blocks* projekt najczęściej będzie się składał z pięciu plików. Jeśli podczas tworzenia nowego projektu zaproponowaliśmy nazwę „test", wówczas cztery pliki otrzymają następujące nazwy: *testMain.cpp*, *testApp.cpp* (pliki źródłowe), *testMain.h*, *testApp.h* (pliki nagłówkowe). Dodatkowy piąty plik niezależnie od zaproponowanej nazwy to *resource.rc*.

Przed budową pliku *Makefile* należy zadbać o to, żeby w systemie zainstalowany został pakiet *make* (narzędzie do zarządzania procesem kompilacji). Można to uczynić w systemie *Debian* lub *Ubuntu* wykonując polecenie:

apt-get install make

oczywiście po uprzednim uaktualnieniu systemu poleceniem:

apt-get update && upgrade

Polecenia wykonujemy w terminalu mając uprawnienia administratora (*root*), ewentualnie poprzedzamy je słowem *sudo* (program umożliwiający wykonanie poleceń zarezerwowanych tylko dla konta administratora), czyli:

sudo apt-get update && upgrade

sudo apt-get install make

Alternatywnie można do instalacji pakietu *make* wykorzystać jedną z wielu nakładek graficznych. Bardzo popularnym menedżerem pakietów w systemie *Debian* jest program *Synaptic*. Po zainstalowania narzędzia *make* konieczne jest zainstalowanie następujących pakietów zawierających klasy *wxWidgets*:

libwxbase3.0-0
libwxbase3.0-dev
libwxgtk3.0-0
libwxgtk3.0-dev
wx-common
wx3.0-headers

Wyżej wymienione pakiety instaluje się analogicznie do sposobu, w jaki zainstalowano pakiet *make*. W celu sprawdzenia poprawnego zainstalowania pakietów należy wydać polecenie:

wx-config --version

w odpowiedzi powinniśmy uzyskać:

3.0.2

Jeśli aplikacja jest umieszczona w jednym pliku o nazwie *main.cpp*, wówczas można ją skompilować i uruchomić prawie jak zwykły program napisany w języku C++. Oto przykład tworzenia pliku wykonywalnego o nazwie *main*:

g++ -o main main.cpp `wx-config - -cxxflags - -libs`

przy czym kompilacja (*`wx-config - -cxxflags`*) i linkowanie (*`wx-config - -libs`*) zostało połączone (*`wx-config - -cxxflags - -libs`*), natomiast uruchamiamy aplikację poleceniem:

./main

Tworzenie pliku wykonywalnego można podzielić na dwa etapy, najpierw zaczynając od kompilacji:

g++ -c main.cpp `wx-config - -cxxflags`

w efekcie, której otrzymamy plik wynikowy z rozszerzeniem .o

main.o

który z kolei poddamy procesowi linkowania:

g++ -o main main.o `wx-config - -libs`

co umożliwi otrzymanie pliku wykonywalnego

main

który następnie będzie można uruchomić poleceniem:

./main

Cały proces można zautomatyzować zapisując kolejne etapy w formie zrozumiałej dla narzędzia *make*. Poniżej zamieszczono przykład zawartości pliku *Makefile*, który pozwala ograniczyć się użytkownikowi aplikacji do wydania polecenia *make* pod warunkiem, że plik *Makefile* oraz plik (ewentualnie pliki źródłowe) źródłowy znajdują się w tym samym katalogu:

```
**********************************************************
main: main.o
        g++ -o main main.o `wx-config - -libs`

main.o: main.cpp
        g++ -c main.cpp `wx-config - -cxxflags`
**********************************************************
```

Ważne jest, aby w linii drugiej i (uwzględniając trzecią linię jako pustą) piątej pliku *Makefile* poprzedzić treść wiersza tabulatorem. Na zakończenie tej sekcji zaprezentowana zostanie zawartość pliku *Makefile* projektu składającego się z czterech plików (*testMain.cpp*, *testApp.cpp*, *testMain.h*, *testApp.h*).

Oto ona:

```
**********************************************************
testMain: testMain.o testApp.o
        g++ -o testMain testMain.o testApp.o `wx-config - -libs`

testMain.o: testMain.cpp
        g++ -c testMain.cpp `wx-config - -cxxflags`

testApp.o: testApp.cpp
        g++ -c testApp.cpp `wx-config - -cxxflags`
**********************************************************
```

Warto pamiętać, że obowiązkowe tabulatory poprzedzające treść wiersza znajdują się w linii nr 2, 5, 7 itd. w przypadku, gdy aplikacja rozbita jest na więcej plików pomocniczych.

4. Wybrane elementy palety Layout

Aby wykorzystać w pełni możliwości biblioteki *wxWidgets* w tworzeniu aplikacji wieloplatformowych wskazane jest wykorzystywanie tzw. sizerów (*ang. sizers*). Sizery umożliwiają w sposób automatyczny dobranie wielkości okna aplikacji do wszystkich elementów znajdujących się w jego obrębie.

Najprostszy sizer o nazwie *wxBoxSizer* grupuje elementy okna takie jak: przyciski (*wxButton*), statyczne pola tekstowe (*wxStaticText*), pola wprowadzania i edycji tekstu (*wxTextCtrl*) itp. w sposób wertykalny lub horyzontalny w zależności od woli programisty.

Bardzo podobny w działaniu do *wxBoxSizer* jest *wxStaticBoxSizer*, który tym różni się od swojego poprzednika, że dodaje dodatkową przestrzeń pomiędzy elementami go wypełniającymi.

Nieco więcej możliwości oferuje programiście *wxGridSizer*, który umożliwia dostosowanie liczby kolumn i wierszy z elementami okna poprzez zmianę odpowiednio właściwości „Col" i „Rows". Ten sizer dzieli okno na komórki o jednakowych rozmiarach i jeśli element komórki nie wypełni w całości przestrzeni dla niego zarezerwowanej występują w komórce wolne przestrzenie.

Jeśli nie odpowiadają nam wolne przestrzenie i podział okna na komórki jednakowej wielkości to najprawdopodobniej zadowoli nas sizer o nazwie *wxFlexGridSizer*, który dostosowuje wielkość komórki do gabarytów elementu występującego w wierszu.

5. Zadania do wykonania

1. Zaimplementować kalkulator, który będzie wykonywał dodawanie, odejmowanie, mnożenie oraz dzielenie dwóch liczb zespolonych. Implementacja powinna wykorzystywać przynajmniej jeden sizer.
2. Zaznajomić się z wykorzystaniem okna *wxFrame* zawierającego informacje dotyczące myszki tj. położenie kursora, stanu poszczególnych klawiszy, *scroll*, położenia kursora w przestrzeni okna itp. Warto przetestować poszczególne makra *EVT_LEFT_DOWN(func)*, *EVT_LEFT_UP(func)*, *EVT_LEFT_DCLICK(func)* itd., a na potrzeby zadania wykorzystać tablicę zdarzeń.
3. Napisać przykładową aplikację, w której zostaną umieszczone 4 dowolne komponenty np. *wxButton, wxComboBox*, *wxListBox*, *wxRadioButton.* Przetestować obsługę zdarzeń z wykorzystaniem tych komponentów, makra *EVT_BUTTON(id, func)*, *EVT_COMBOBOX(id, func)* itp., a na potrzeby zadania wykorzystać funkcję *Connect()* lub *Bind()*.
4. Zapoznać się z tworzeniem pliku *Makefile* dla plików wynikowych aplikacji z poprzedniej instrukcji laboratoryjnej, czyli kalkulatora.

Treść zadania dla przypomnienia zawarta jest poniżej:

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).