# Instrukcja laboratoryjna nr 9

## Programowanie w języku C 2

### (C++ poziom zaawansowany)

# Wizualne tworzenie aplikacji z wykorzystaniem biblioteki wxWidgets

# część 2

dr inż. Jacek Wilk-Jakubowski

mgr inż. Maciej Lasota

dr inż. Tomasz Kaczmarek

# Wstęp

Na poprzednich zajęciach laboratoryjnych studenci wykorzystali środowisko Code::Blocks do budowy prostej aplikacji typu kalkulator wykorzystując w tym celu wieloplatformową bibliotekę wxWidgets oraz system Windows. W ramach dzisiejszych zajęć zapoznamy się z kompilacją projektu utworzonego na poprzednich zajęciach w systemie Linux w dystrybucji Debian lub Ubuntu przy pomocy odpowiednio skonstruowanego pliku *Makefile* oraz pogłębimy naszą znajomość biblioteki wxWidgets budując kolejną aplikację.

# Budowa pliku *Makefile*

W przypadku budowy aplikacji w środowisku Code::Blocks nasz 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 przez nas nazwy to *resource.rc*.

Przed budową pliku *Makefile* powinnyśmy zadbać o to, żeby w naszym 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 root'a), 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 instalujemy analogicznie do sposobów, którymi instalowaliśmy pakiet *make*. W celu sprawdzenia poprawnego zainstalowania pakietów należy wydać polecenie

*wx-config --version*

w odpowiedzi powinniśmy uzyskać:

*3.0.2*

Jeśli nasza 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 oczywiście poleceniem

*./main*

Tworzenie pliku wykonywalnego możemy rozbić na dwa etapy najpierw zaczniemy od komilacji

*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 nam otrzymanie pliku wykonywalnego

*main*

któr następnie będziemy mogli uruchomić poleceniem

*./main*

Cały proces można zautomatyzować zapisując kolejne etapy w formie zrozumiałej dla narzędzia *make*, oto 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 tego podrozdziału 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`
**********************************************************
```

Przypominam, 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.

# Wybrane elementy palety Layout

Aby wykorzystać w pełni możliwości biblioteki wxWidgets w tworzeniu aplikacji wieloplatformowych wskazane jest wykorzystywanie tzw. sizer'ów (ang. sizers). Sizer'y 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.

# Zadania do wykonania

1. Utworzyć plik Makefile dla plików wynikowych projektu z zadnia nr 2 z poprzedniej instrukcji czyli dla aplikacji kalkulator.
2. Zaimplementować kalkulator, który będzie wykonywał dodawanie, odejmowanie, mnożenie oraz dzielenie dwóch liczb zespolonych. Implementacja powinna wykorzystywać przynajmniej jeden sizer.
3. Utworzyć plik Makefile dla plików wynikowych projektu z zadania nr 2.