

# Biblioteki, biblioteki...

---

Dzisiejsze zadanie polega na przygotowaniu zestawu bibliotek niezbędnych do rozpoczęcia eksperymentów z grafiką i interfejsem użytkownika.

Będziemy korzystać z dwóch bibliotek:

`fltk` – *Fast Light Toolkit* (z domeny publicznej)

`graph_lib` – nakładka na `fltk` przygotowana przez  
Bjarne Stroustrupa

Zadanie nie jest wbrew pozorom błahe – szczególnie jeśli chodzi o budowanie `fltk` (która nie jest aż tak light, jak można by się było spodziewać).

Bibliotekę `graph_lib` (malutką) przygotujemy ręcznie w Codeblocks definiując stosowny projekt.

## Biblioteki – czynności wstępne

---

Pobranie z repozytorium kodu źródłowego FLTK  
(w wersji 1.3.8):

<https://www.fltk.org/pub/fltk/1.3.8/fltk-1.3.8-source.tar.gz>

Jeśli nie macie Państwo programu WinRAR (lub innego obsługującego archiwa tar.gz możecie pobrać źródła fltk z Moodle).

Pobranie z repozytorium programu CMake 3.22.3 w wersji nieinstalacyjnej:

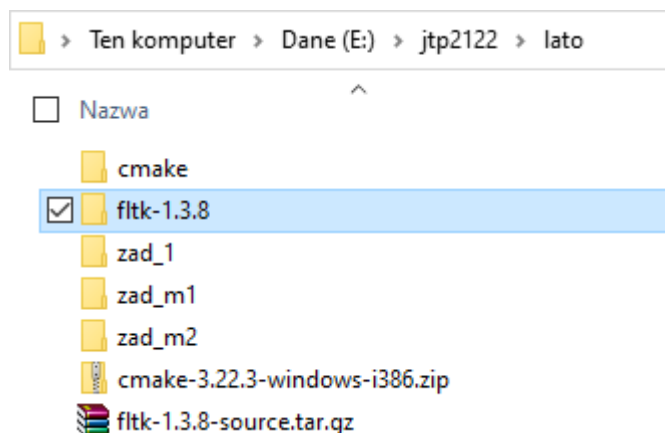
<https://github.com/Kitware/CMake/releases/download/v3.22.3/cmake-3.22.3-windows-i386.zip>

Pliki źródłowe biblioteki graph\_lib są dostarczone w pakiecie zadania trzeciego (małego) w folderze graph\_lib (cały pakiet jest tradycyjnie dostępny na Moodle).

# Przygotowanie struktury folderów

---

1. Ustalenie, w którym folderze mamy MinGW (tzn. zestaw narzędzi programistycznych, które pobraliśmy z Codeblocks 20.03).
2. Zidentyfikowanie foldera, do którego mamy pełne prawa dostępu (folder z projektami jest bardzo dobry – całości kodu źródłowego, jak i CMake będziemy potrzebować tylko w czasie budowania biblioteki)
3. Rozpakowanie do tego foldera biblioteki fltk i programu CMake (u siebie zmieniłem nazwę foldera programu CMake na krótszą)



# Środowisko systemowe do budowania biblioteki

Po uruchomieniu wiersza poleceń (cmd) trzeba przejść do foldera docelowego i wykonać szereg sprawdzeń (i ew. uzupełnić konfigurację):

```
Microsoft Windows [Version 10.0.19043.1586]  
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.
```

```
C:\Users\PWWEITIII>e:  
E:\>cd jtp2122\lato\cmake
```

Ponieważ mój folder jest na dysku e:  
zmieniam najpierw dysk, a następnie  
aktualny katalog

```
E:\jtp_21L\lato\cmake>set MINGW_HOME=e:\jtp2122\cb-20.03\MinGW  
E:\jtp_21L\lato\cmake>path=%path%;e:\jtp2122\cb-20.03\MinGW\bin
```

```
E:\jtp_21L\lato\cmake>cd bin  
E:\jtp_21L\lato\cmake\bin>cmake-gui
```

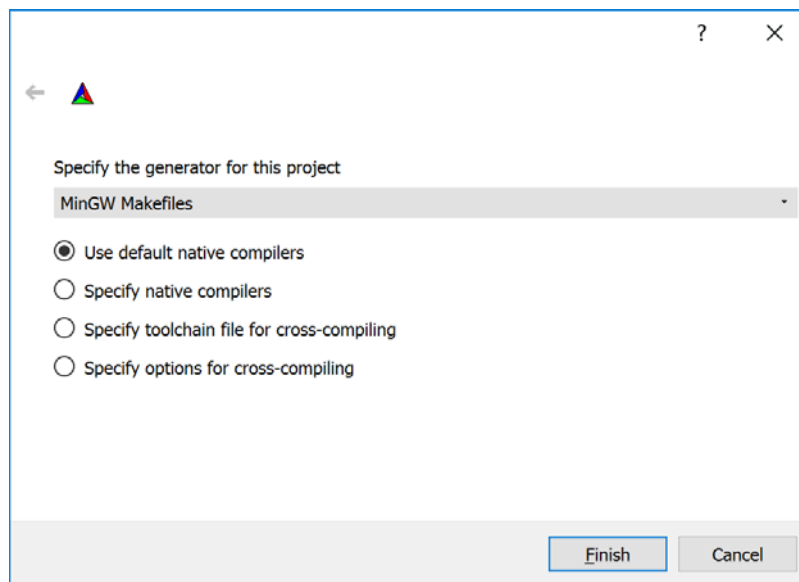
Kolejne polecenia ustawiają zmienną  
systemową MINGW\_HOME oraz dodają  
do ścieżki wyszukiwania folder z  
programami MinGW i wreszcie  
uruchomienie CMake

# CMake – konfiguracja

---

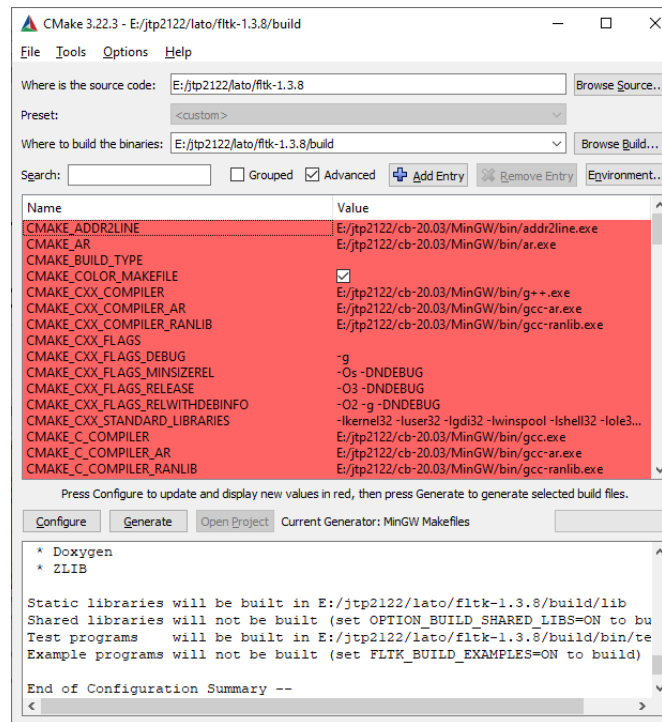
Po uruchomieniu CMake należy ustawić lokalizację folderów: źródłowego (fltk-1.3.8) i wynikowego (fltk-1.3.8/build) – ten ostatni folder trzeba utworzyć.

Po naciśnięciu Configure należy sprecyzować generator, który będzie używany do przygotowania konfiguracji: używamy MinGW Makefiles.



# CMake – generacja

Nie będziemy analizować komunikatów, które pojawiają się w trakcie konfiguracji. Zostawiamy też domyślne ustawienia opcji.



Po naciśnięciu Generate zostanie rozpoczęta procedura tworzenia pliku Makefile dla programu make (narzędzia ułatwiającego budowanie złożonych projektów).

# Budowanie biblioteki fltk

Po zakończeniu tworzenia pliku Makefile możemy zamknąć CMake. Resztę pracy załatwią narzędzia MinGW.

Plik dla programu make (dokładniej: mingw32-make) został utworzony w folderze docelowym wskazanym w CMake. Przechodzę do tego foldera i uruchamiam mingw32-make:

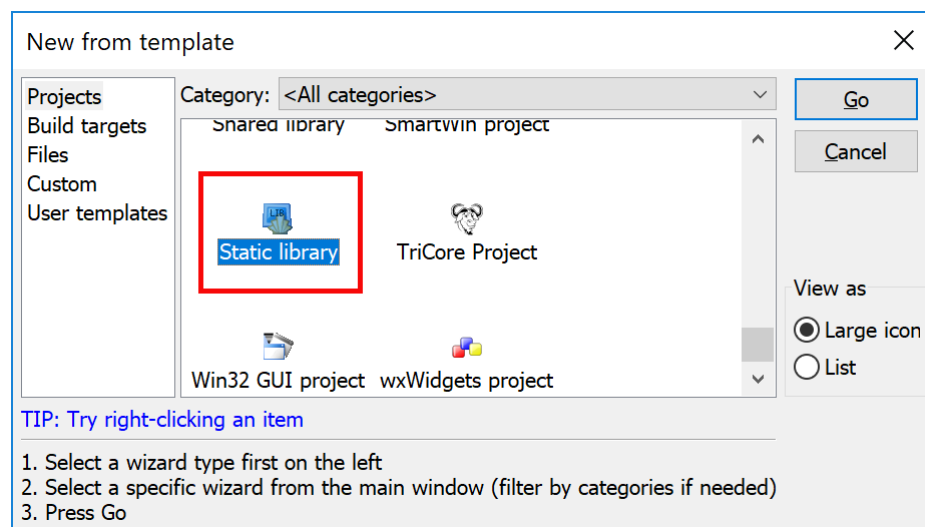
```
E:\jtp2122\lato\cmake\bin>cd ..\..\nE:\jtp2122\lato>cd fltk-1.3.8\build\nE:\jtp2122\lato\fltk-1.3.8\build>mingw32-make
```

Jeśli wszystko poszło dobrze, to mamy kilka minut oddechu, a kompilator, linker i archiwizator – zajęcie.

Nazwa	Data modyfikacji	Typ
bin	17.03.2022 20:39	Folder plików
CMakeFiles	17.03.2022 20:42	Folder plików
data	17.03.2022 20:32	Folder plików
etc	17.03.2022 20:32	Folder plików
FL	17.03.2022 20:32	Folder plików
fluid	17.03.2022 20:35	Folder plików
jpeg	17.03.2022 20:35	Folder plików
lib	17.03.2022 20:39	Folder plików
png	17.03.2022 20:35	Folder plików

# Projekt biblioteki statycznej (1)

Pora na zbudowanie biblioteki `graph_lib` Stroustrupa. Nowy projekt stworzymy standardowo, ale typ projektu standardowy już nie jest:

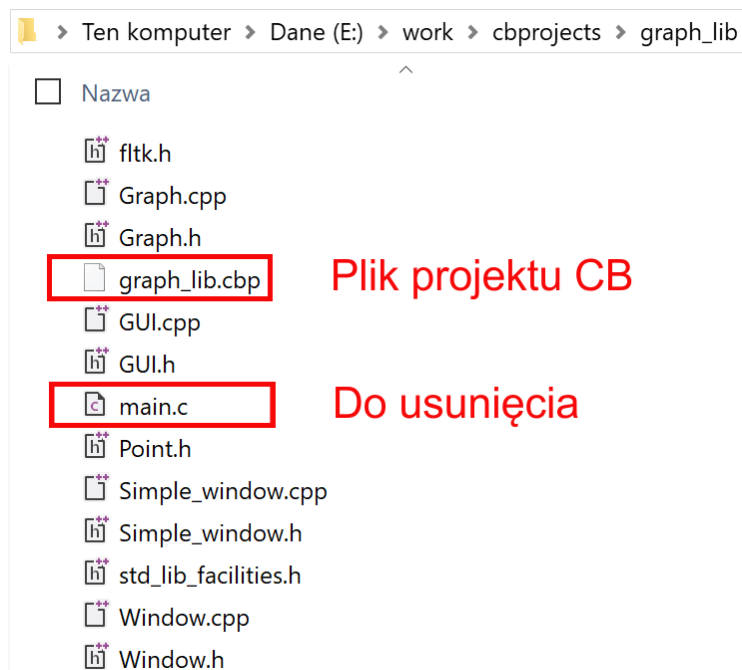


Nazywam projekt `graph_lib` i standardowo nie zmieniam nic w ustawieniach konfiguracji. W projekcie jest domyślnie utworzony plik `main.c` – usuwam go z projektu, bo wszystkie pliki źródłowe biblioteki pochodzą z pakietu ćwiczenia trzeciego (nieuchronnie z foldera `graph_lib`).

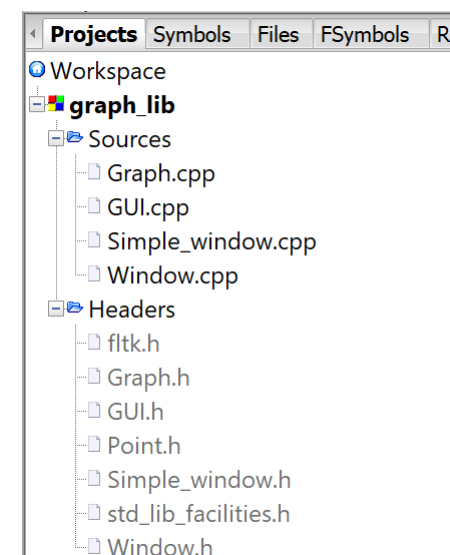


## Projekt biblioteki statycznej (2)

Po dodaniu plików źródłowych folder projektu wygląda następująco:



Po pozbyciu się niepotrzebnego `main.c`, trzeba dodać pliki źródłowe do projektu (Project > Add files...). Przestrzeń robocza z projektem `graph_lib` jest widoczna po prawej stronie.

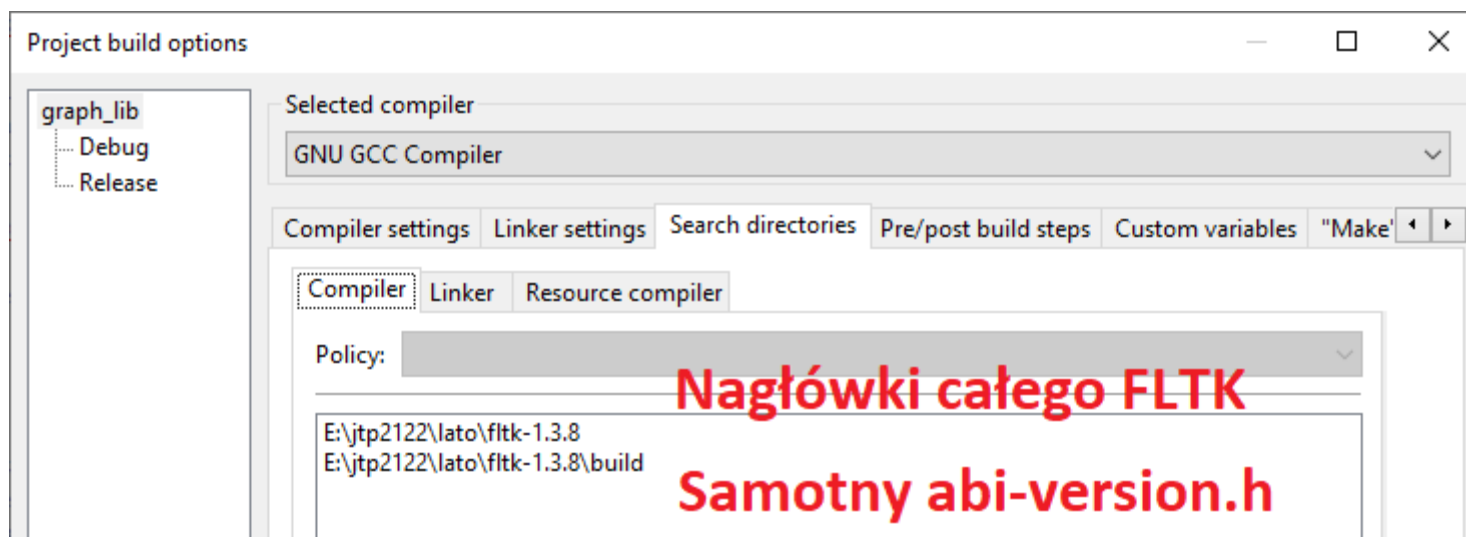


## Projekt biblioteki statycznej (3)

Próba budowania biblioteki kończy się malowniczym padem już na pierwszym pliku:

**..\fltk.h|4|fatal error: FL/Fl.H: No such file or directory|**

Problem w tym, że kompilator nie wie, gdzie szukać plików nagłówkowych. W opcjach budowania (Project > Build options...) w zakładce Search directories dokładamy:



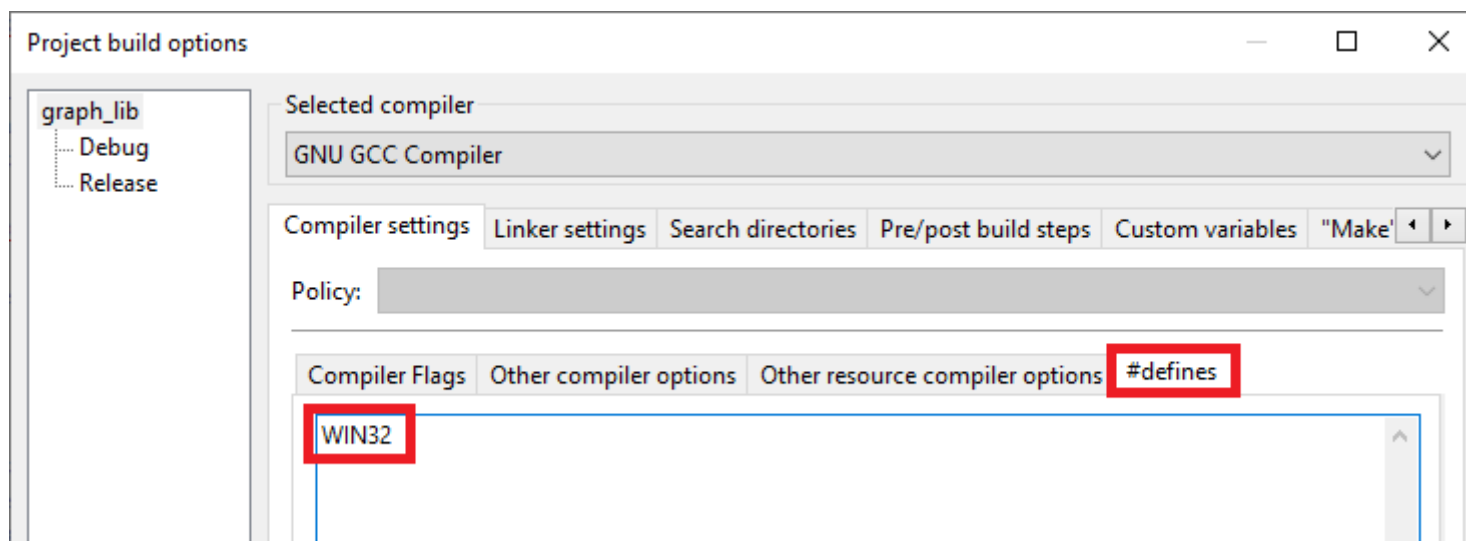
(Flaga C++11 w ustawieniach kompilatora powinna być ustawiona.)

## Projekt biblioteki statycznej (4)

Dalej bez sukcesu i choć błąd wygląda znajomo:

**...\x.H|37|fatal error: X11/Xlib.h: No such file or directory|**

jego rozwiązanie jest zupełnie inne. Musimy poinformować kompilator, że docelowy system to Windows:



W zakładce #defines należy zdefiniować stałą WIN32. Po tym kroku biblioteka buduje się poprawnie, choć z masą ostrzeżeń. Ostrzeżeń nie można oczywiście zostawić bez reakcji (opis poniżej będzie jest przykładem walki z ostrzeżeniami w „obcym” kodzie).

## Projekt biblioteki statycznej (5)

---

Ostatnie z ostrzeżeń manifestuje się następująco:

```
...\Graph.cpp||In function 'Graph_lib::Suffix::Encoding Graph_lib::get_encoding(const string&)':|
```

```
...\Graph.cpp|304|warning: unused variable 'x' [-Wunused-variable]|
```

Statyczna zmienna x została tu użyta po to, żeby funkcja `init_suffix_map` została wykonana tylko raz, i tylko wtedy kiedy jest faktycznie potrzebna.

Ponieważ momentem stwierdzenia, że zmienna jest nieużywana jest skończenie zasięgu funkcji, stosowne dyrektywy „tłumiące” na chwilę ostrzeżenie trzeba dodać przed funkcją:

```
#pragma GCC diagnostic push
```

```
#pragma GCC diagnostic ignored "-Wunused-variable"
```

a po zakończeniu bloku funkcji wrócić do ustawień domyślnych:

```
#pragma GCC diagnostic pop
```

Podobny manewr w `fltk.h` jest już wykonany. Biblioteka powinna kompilować się bez ostrzeżeń 😊

# Wreszcie aplikacja

---

Przy budowaniu bibliotek mieliśmy nieco ułatwioną sytuację, bo biblioteki nie są konsolidowane (tzn. odwołania do zewnętrznych funkcji, nie są w czasie powstawania biblioteki wiązane z odpowiednimi funkcjami).

Ten krok musi być wykonany w czasie tworzenia aplikacji, która w odróżnieniu od biblioteki może być wykonana. Głównym zajęciem przy tworzeniu projektu aplikacji będzie wskazanie bibliotek, w których znajdują się te niepowiązane funkcje. Warto mieć świadomość, że prócz biblioteki `graph_lib` (z której korzystamy bezpośrednio), trzeba będzie wskazać biblioteki, z których korzysta `graph_lib` (`fltk`) i biblioteki, z których korzysta `fltk` (itd...)

Oczywiście niedole związane z lokalizacją plików nagłówkowych, wskazywaniem docelowego systemu operacyjnego i w aplikacji trzeba będzie przeżyć.

Zatem to, że aplikacja składa się z jednego, małego pliku jest niewielkim pocieszeniem.

# Projekt aplikacji (1)

---

Tworzenie projektu aplikacji zaczyna się standardowo (nazwijmy ją: `tst_graph`). Plik `main.cpp` można wyrzucić z projektu i usunąć z dysku, bo w pakiecie zadania jest plik `tst_graph.cpp`, który trzeba skopiować do folderu projektu i dodać do projektu.

Pierwsza próba budowania kończy się szybko:

```
...\tst_graph.cpp|1|fatal error: graph.h: No such file or directory|
```

Przyczyną jest oczywiście problem ze znalezieniem plików nagłówkowych biblioteki `graph_lib`. W zakładce `Search directories` trzeba dodać folder, w którym znajdują się te pliki i od razu foldery z plikami nagłówkowymi `fltk`:

```
E:\jtp2122\lato\graph_lib  
E:\jtp2122\lato\fltk-1.3.8  
E:\jtp2122\lato\fltk-1.3.8\build
```

Od razu włączam opcję `C++11` i definiuję `WIN32`.

## Projekt aplikacji (2)

---

Kompilacja przeszła, ale konsolidator dał 27 błędów „**undefined reference to function**” – to skutek konsolidacji, czyli wiązania (a przynajmniej próby) wywołań funkcji z ich kodem.

W zakładce Linker settings podajemy długą listę bibliotek:

libgraph_lib.a	biblioteka Stroustrupa
libfltk.a	
libfltk_images.a	dwie biblioteki z FLTK
libgdi32.a	
libole32.a	
libuuid.a	
libcomctl32.a	biblioteki systemowe

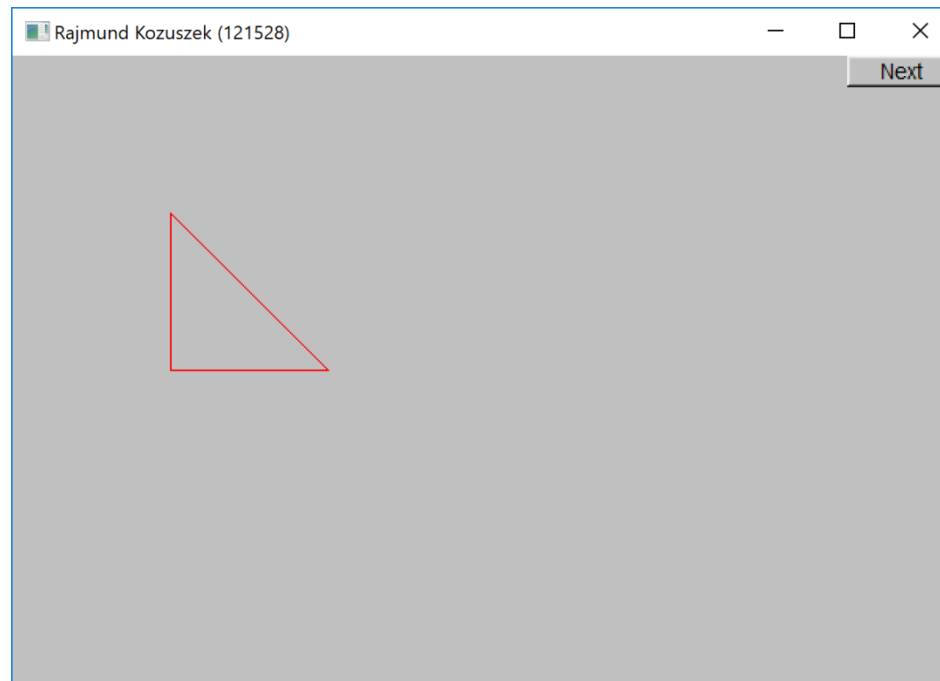
Dodatkowo, w zakładce Search directories > Linker trzeba jeszcze podać dwa foldery, w których konsolidator ma szukać niesystemowych bibliotek:

```
E:\jtp2122\lato\graph_lib\bin\Debug  
E:\jtp2122\lato\fltk-1.3.8\build\lib
```

# Widok okna aplikacji

---

Po tych wszystkich zabiegach mamy wreszcie działającą, choć niezbyt imponującą, aplikację:



Ponieważ „trafienie” w foldery nagłówek i bibliotek bywa sprawą kłopotliwą, na zakończenie przeniesiemy i pliki nagłówkowe, i biblioteki do folderów używanych domyślnie przez MinGW.



# Reorganizacja środowiska

---

Ponieważ w dalszej części semestru będziemy jeszcze parę ładnych razy robić projekty z wykorzystaniem `fltk` i `graph_lib` warto uprościć konfigurację kolejnych projektów. Wykorzystamy do tego szablon projektu.

Polecenie `File > Save project as template` robi dokładnie to, co nam potrzebne. Ja nazywam swój szablon `graph_lib_project`. Tworząc kolejne projekty używające biblioteki `graph_lib` będę je bazował na szablonie użytkownika `graph_lib_project`. Będę co prawda musiał utworzyć ręcznie folder dla projektu, ale w porównaniu do lokalizowania folderów i wpisywania bibliotek, to niewielki koszt.

## Zadanie małe trzecie

---

Trzecie z małych zadań polega na przygotowaniu środowiska do eksperymentów z biblioteką `graph_lib` i uruchomienie programu testowego z jedną, drobną modyfikacją.

W pasku tytułu okna powinno znaleźć się imię i nazwisko autora (bez polskich znaków) a następnie w nawiasie numer albumu.

Jako dowód wykonania zadania ładujecie Państwo w zadaniu Moodle *Biblioteki, biblioteki* obraz okna programu (i tylko tego okna) w formacie png. **Nazwa pliku, to numer Państwa albumu.**

Rozwiązania należy złożyć do:

**27 marca 2022 23:59**