Informatyka 2

Modułowa budowa złożonych programów (wersja robocza komentarzem)

Paweł Strączyński
pstraczynski@tu.kielce.pl

Tematem wykładu jest tworzenie złożonych programów w języku C. W wykładzie tym dowiemy się jak korzystać z bibliotek programistycznych oraz jak tworzyć i udostępniać stworzone przez siebie moduły

Modułowa budowa programu

Każdy duży program składa się z określonej liczby wydzielonych części zwanych modułami.

Podział programu na mniejsze części zwane modułami pozwala:
- zachowanie przejrzystości tworzonego programu,
- łatwe wykorzystanie tego samego kodu w różnych programach,
- możliwość niezależnego kompilowania poszczególnych modułów programu.

Modułem nazywamy więc wydzieloną cześć programu np. odpowiedzialną za daną operację.
Moduły z których mogą korzystać różne programy umieszcza się w specjalnie przygotowanych, skompilowanych plikach nazywanych **bibliotekami**.

Biblioteki tworzymy po to aby zachować większą przejrzystość w tworzonym oprogramowaniu - funkcje odpowiedzialne za konkretne funkcjonalności oprogramowania wygodnie jest umieścić w osobnym pliku w postaci biblioteki. Ponadto korzystanie z bibliotek umożliwia wykorzystanie tego samego fragmentu kodu wielokrotnie w wielu programach

Czym jest biblioteka?

Biblioteką nazywamy odrębny plik zawierający zbiór funkcji i zmiennych (klas w przypadku języków obiektowych).

Funkcje te zostały wydzielone do postaci pliku, aby dało się z nich korzystać w wielu programach.

Dzięki takiemu rozwiązaniu tworzone oprogramowanie staje się modularne.

Korzystanie z bibliotek znacząco ułatwia pisanie programu, gdyż pozwala na wykorzystywanie we własnym programie funkcji wcześniej przygotowanych np. przez innych programistów.

Przykładem bibliotek są biblioteki standardowe języka C. Zwierają one szereg funkcji powszechnie stosowanych podczas pisania programów np. funkcje `printf(), scanf()`

Biblioteki składają się z:

- interfejsu - pliku nagłówkowego (`.h`), zawierającego deklaracje funkcji bibliotecznych. - plik ten stanowi informacje dla programisty co dany moduł udostępnia - pliki nagłówkowe często zawierają komentarze opisujące działanie oraz sposób użycia poszczególnych funkcji.
- implementacji - pliku źródłowego (`.c`) - po kompilacji w postaci kilku binarnego - zawierającego implementację poszczególnych funkcji bibliotecznych

Rodzaje bibliotek

Ze względu na moment w który biblioteki zostanie dołączona do programu biblioteki dzielimy na:

- statyczne (*ang. static libraries*) - biblioteka która jest łączona jest z programem w momencie konsolidacji (linkowania).
  - *WINDOWS* **.lib**, **.obj**,
  - *UNIX* **.a .o**,
- dynamicznie linkowane - biblioteka jest dynamicznie łączona z programem trakcie jego uruchomienia lub działania
  - współdzielone (ang. shared libraries), **.so** - dołączana w trakcie ładowania programu,
  - bibliotek ładowane dynamicznie (ang. dynamically loaded libraries). **.dll** - dołączana w trakcie działania programu.

W ramach kursu Informatyka 2 skupimy się głównie na tworzeniu bibliotek statycznych. Warto wiedzieć jednak, że istnieją biblioteki które są łączone z programem w trakcie jego działania lub uruchamiania. Zmiany w zawartości funkcji takiej biblioteki nie wymagają kompilacji programu który korzysta z biblioteki. Oczywiście zmodyfikowana biblioteka musi zawierać te same funkcje pobierające te same argumentu i zwracające to samo co biblioteka pierwotnie wykorzystana do pisania programu

Biblioteki statyczne

Biblioteki statyczne są dołączane do programu w czasie kompilacji. Aby użyć biblioteki statycznej w programie należy:

- załączyć plik nagłówkowy biblioteki,
- podczas kompilacji dokonać połączenia kodu aplikacji z kodem biblioteki.

```c
/* main.c */

#include <stdio.h>
#include "kolo.h"

#define R 20

int main (void) {
    printf("Obwod kola o promieniu %d wynosi %d\n”,,R, obwod(R));

    return(0);
}
```

```c
/* kolo.c */

#include „kolo.h"

int obwod(int r)
{
    return 2*3.14*r;
}

int pole(int r)
{
    return 3.14*r*r;
}
```

```c
/* kolo.h */

/* Oblicza obwód kola o zadanym promieniu */
int obwod(int r);

/* Oblicza pole kola o zadanym promieniu */
int pole(int r);
```

Z biblioteki najcześciej korzystamy w sposób jak w zamieszczonym przykładzie. W pliku programu main.c załączamy plik nagłówkowy który przechowuje deklaracje funkcji. Definicje funkcji znajdują się w pliku źródłowym o tej samej nazwie. Zwykle z biblioteka nie jest dostarczany plik źródłowy .c a tylko nagłówkowy .h oraz plik obiektowy biblioteki.

Biblioteki statyczne

Budowanie programu z własną biblioteka statyczną w systemie *UNIX:*

- Pliki źródłowe biblioteki kompilujemy do postaci pliku obiektowego **.o**
- Używając programu **ar** tworzymy archiwum - bibliotekę statyczną z rozszerzeniem **.a** (*ang. archive*).
- Kompilujemy plik źródłowy programu `main.c` wskazując kompilatorowi aby dołączył bibliotekę.
- Uruchamiamy zbudowany program.

```
$ gcc -c kolo.c -o kolo.o
$ ar rcs libkolo.a kolo.o
$gcc main.c -o program -L. libkolo.a
$ls
kolo.c      kolo.o      main.c
kolo.h      libkolo.a   program
$./program
Obwod kola o promieniu 20 wynosi 125
```

Opcja `-L.` informuje kompilator, aby biblioteki szukał w bieżącym katalogu.

Jeżeli zapiszemy bibliotekę z przedrostkiem lib (np. `libkolo.a`) to możemy używając przełącznika `-l` dokonać kompilacji bez podawania przedrostka i rozszerzenia.

```
$gcc main.c -o program -L. -lkolo
```

Przykład dostępny w postaci wideo

W ramach zajęć laboratoryjnych najwygodniej będzie Państwu korzystać z IDE w którym proces konsolidacji i kompilacji programu z biblioteką jest dosyć łatwy. Dla wszystkich zainteresowanych szczególnie tych którzy korzystają z systemów Linux i macOS przykład jak wygląda proces budowania programu w uniksie. Korzystamy tu z kompilatora gcc z poziomu terminala systemowego.

Biblioteki statyczne

```
$ gcc -c kolo.c -o kolo.o
$ gcc -c kwadrat.c -o kwadrat.o
$ ar rcs libfigury.a kolo.o kwadrat.o
$gcc main.c -o test -L. -lfigury
$ls
kolo.c      kolo.o      kwadrat.h   libfigury.a test
kolo.h      kwadrat.c   kwadrat.o   main.c
$./test
Obwod kola o promieniu 20 wynosi 125
Pole kwadratu o boku 10 wynosi 100
```

/* main.c */

```
#include <stdio.h>
#include "kolo.h"
#include „kwadrat.h"

#define R 20
#define A 10

int main (void) {
    printf("Obwod kola o promieniu %d wynosi %d\n",,R, obwodKolo(R));
    printf("Pole kwadratu o boku %d wynosi %d\n", A, poleKwadrat(A));
    return(0);
}
```

Tym razem przykład z dwiema bibliotekami.

/* kwadrat.c */

```
#include „kwadrat.h"

int obwodKwadrat(int a)
{
    return 4*a;
}

int poleKwadrat(int a)
{
    return a*a;
}
```

/* kolo.c */

```
#include „kolo.h"

int obwodKolo(int r)
{
    return 2*3.14*r;
}

int poleKolo(int r)
{
    return 3.14*r*r;
}
```

/* kwadrat.h */

```
/* Oblicza obwód kwadratu o długości */
int obwodKwadrat(int a);

/* Oblicza pole kwadratu o długości */
int poleKwadrat(int a);
```

/* kolo.h */

```
/* Oblicza obwód kola o zadanym promieniu */
int obwodKolo(int r);

/* Oblicza pole kola o zadanym promieniu */
int poleKolo(int r);
```

Program make

Program **`make`** powstał w celu automatyzacji kompilacji złożonych programów. Jeśli program zawiera wiele plików kompilacja ręczna może stać się bardzo uciążliwa. Program make analizuje pliki `Makefile` i na ich podstawie wykonuje określone czynności.

Plik Makefile składa się z:

- zależności określających kolejność wykonywania czynności,
- reguł które określają jak dany plik należy skompilować.

```
# komentarz

cel: zależności
→|reguły
```

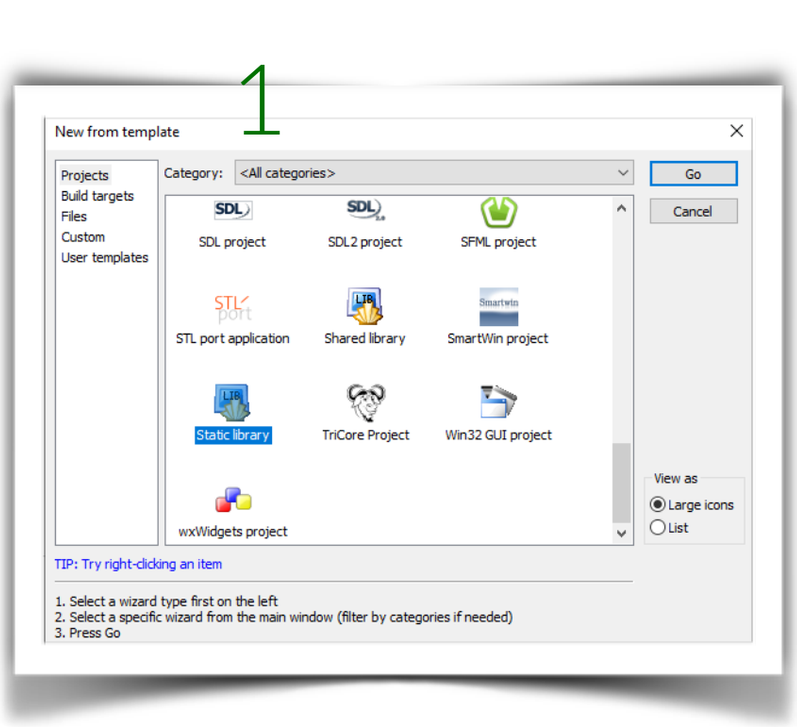
Ze względu na to, że w bardziej złożonych programach budowanie ręczne jest dość mozolne powstał make program który pozwala automatyzować proces kompilacji. Informacje te proszę bardziej potraktować jako ciekawostkę. Nie będę wymagał tworzenia własnych Makefile oraz kompilacji „recznie” z użyciem gcc w ramach ćwiczeń laboratoryjnych. Warto jednak orientować się w temacie

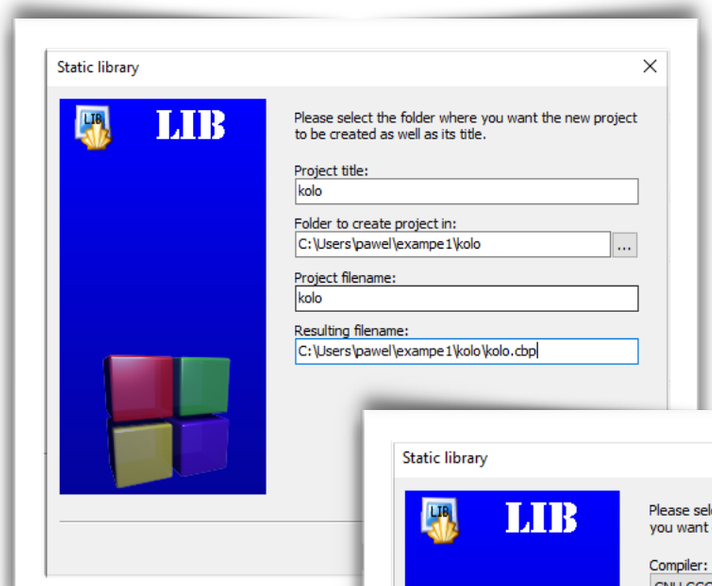
Przykład dostępny w postaci wideo

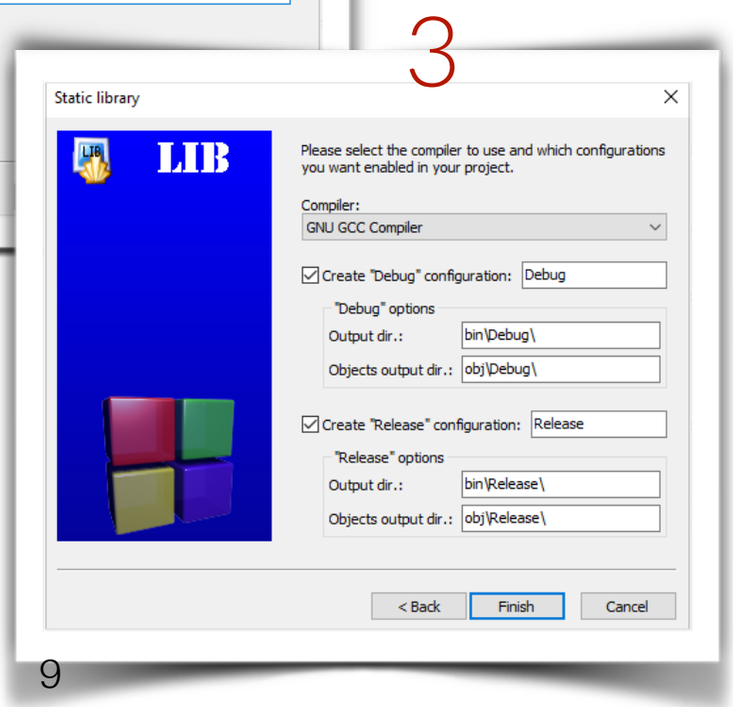
```
# Makefile #

CC = gcc

all: main.c kolo.o kwadrat.o
    $(CC) main.c -o test kolo.o kwadrat.o

kolo.o: kolo.c
    $(CC) -c kolo.c

kwadrat.o: kwadrat.c
    $(CC) -c kwadrat.c
```

Tworzenie biblioteki statycznej w Code::Blocks

Korzystając z IDE takiego jak Code::Blocks tworzenie biblioteki statycznej jest bardzo proste - po odpowiednim skonfigurowaniu projektu konsolidacja odbywa się automatycznie podczas kompilacji programu. Aby utworzyć nową bibliotekę statyczną należy utworzyć nowy projekt typu `Static library`. Możliwe jest to wybierając z menu **`File -> New -> Project`** a następnie `w` oknie **`New from template`** opcje **`Static library`** i zatwierdzamy przyciskiem **`Go`**.



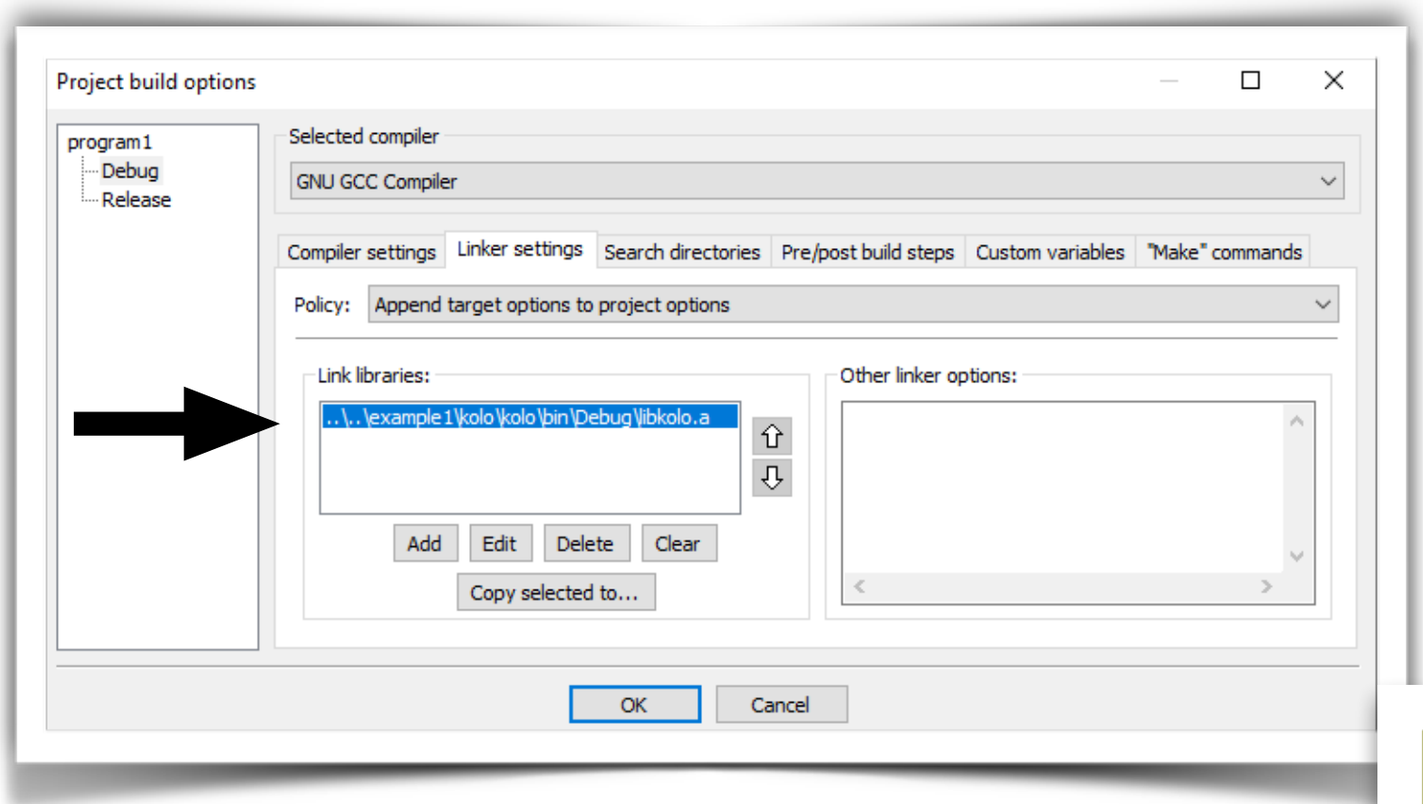




Nie wiem jakiego IDE używaliście Państwo na zajęciach z przedmiotu Informatyka 1 ale ja polecam użyć Code::Blocks'a. Tworzenie projektu biblioteki sprowadza się do przeklinania prostego wizarda.

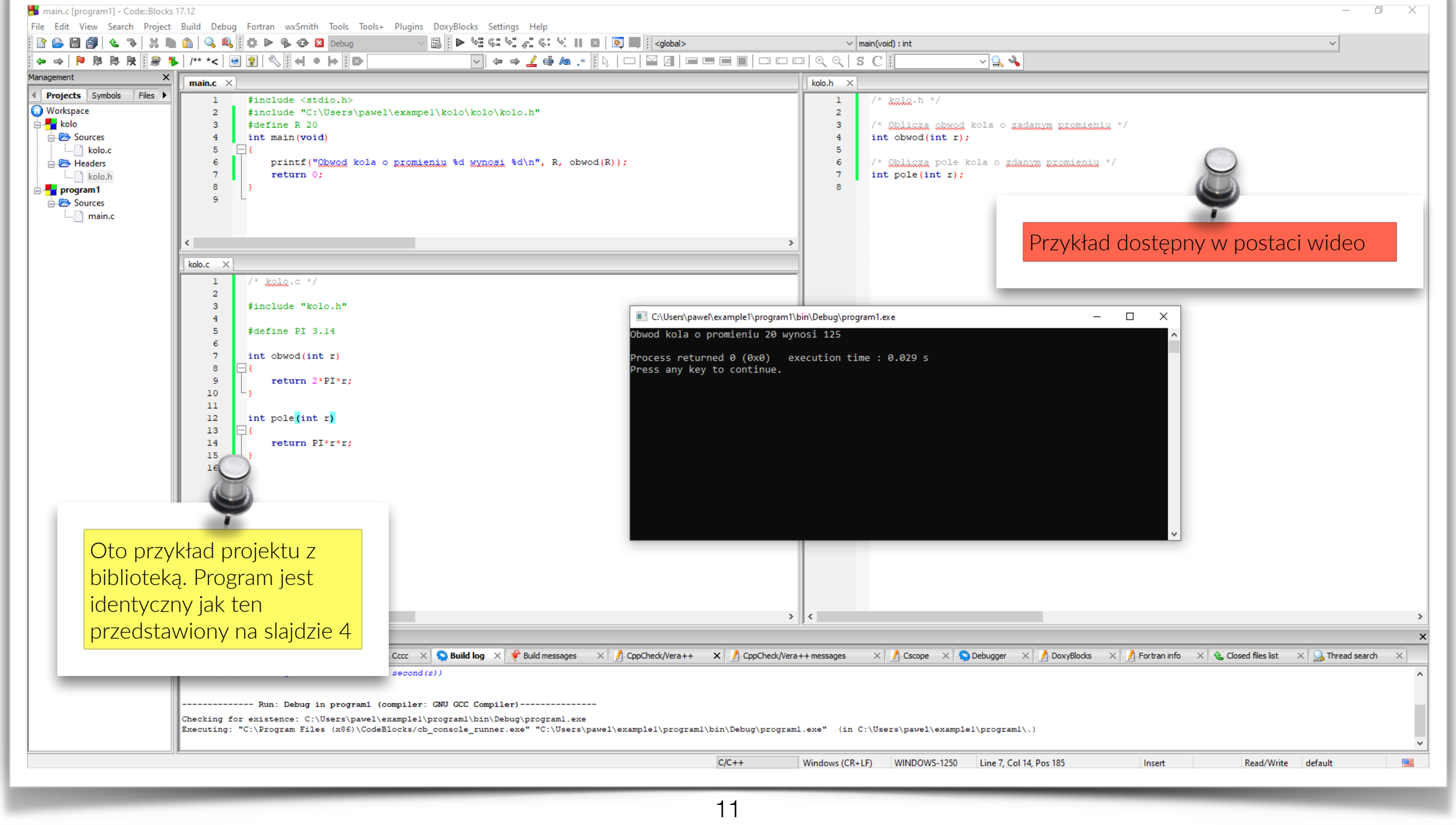
Linkowanie biblioteki statycznej w Code::Blocks

Po utworzeniu nowego projektu aplikacji **`Console application`** należy z menu **`Project`** wybrać **`Project build options.`** Następnie w karcie **`Linker settings`** należy dodać nową bibliotekę.



Jeżeli mamy już utworzoną i skompilowaną bibliotekę to musimy ją podlinkować w projekcie programu

Program korzystający z biblioteki statycznej w Code::Blocks



Biblioteki dynamiczne

Biblioteka współdzielona

- programy używające biblioteki współdzielonej nie zawierają kodu biblioteki a jedynie referencje do biblioteki,
- biblioteka jest ładowana do pamięci razem z uruchomieniem programu,
- zmodyfikowanie biblioteki współdzielonej spowoduje zmianę działania programu bez konieczności kompilacji.

Biblioteka ładowana dynamicznie

- programy używające biblioteki ładowanej dynamicznie nie zawierają kodu biblioteki ale aby używać biblioteki muszą korzystać ze specjalnych metod do ich ładowania.
- biblioteka jest ładowana dynamicznie w trakcie działania programu
- zmodyfikowanie biblioteki ładowanej dynamicznie spowoduje zmianę działania programu bez konieczności kompilacji.