Laboratorium 0: „Środowisko Code::Blocks”

dr inż. Arkadiusz Chrobot

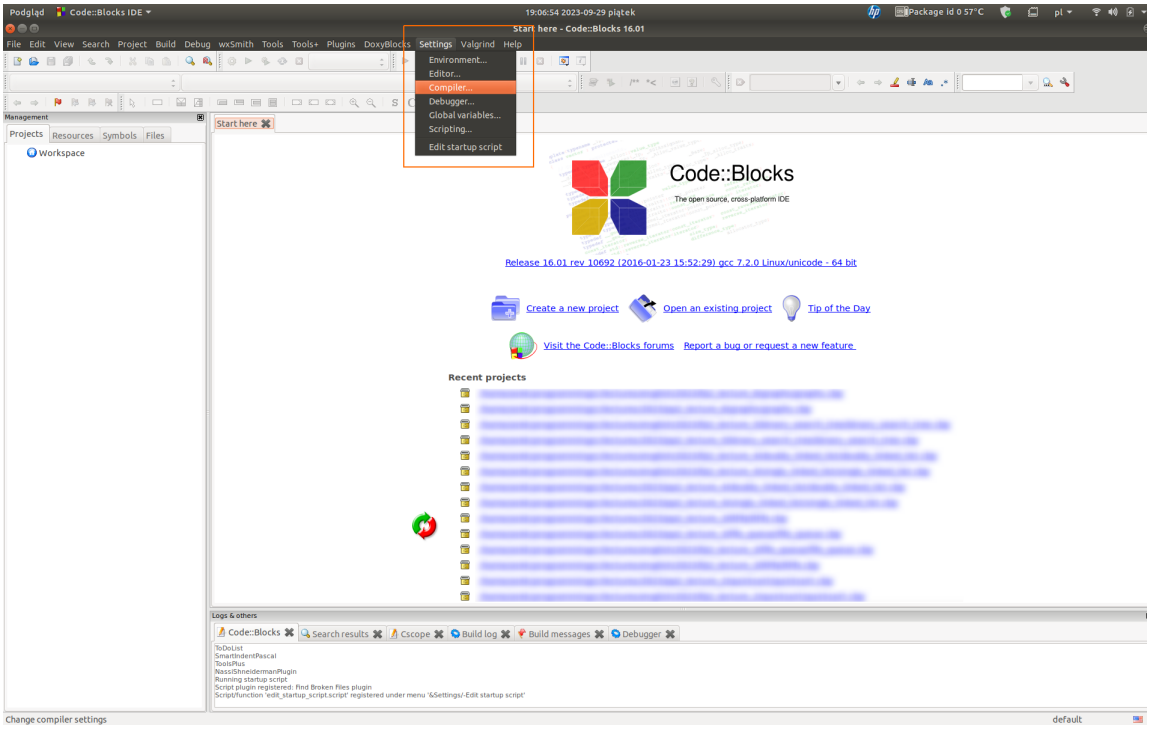
1 października 2023

1. Wprowadzenie

W tej instrukcji został zawarty krótki opis zintegrowanego środowiska do tworzenia programów w języku C, o nazwie Code::Blocks. Program ten wydawany jest na wolnej licencji GPL. Można go pobrać ze strony projektu https://www.codeblocks.org. Środowisko dostępne jest dla wszystkich popularnych systemów operacyjnych i współpracuje z wieloma kompilatorami języka C. Na zajęciach laboratoryjnych będzie używany kompilator `gcc`.

2. Konfiguracja

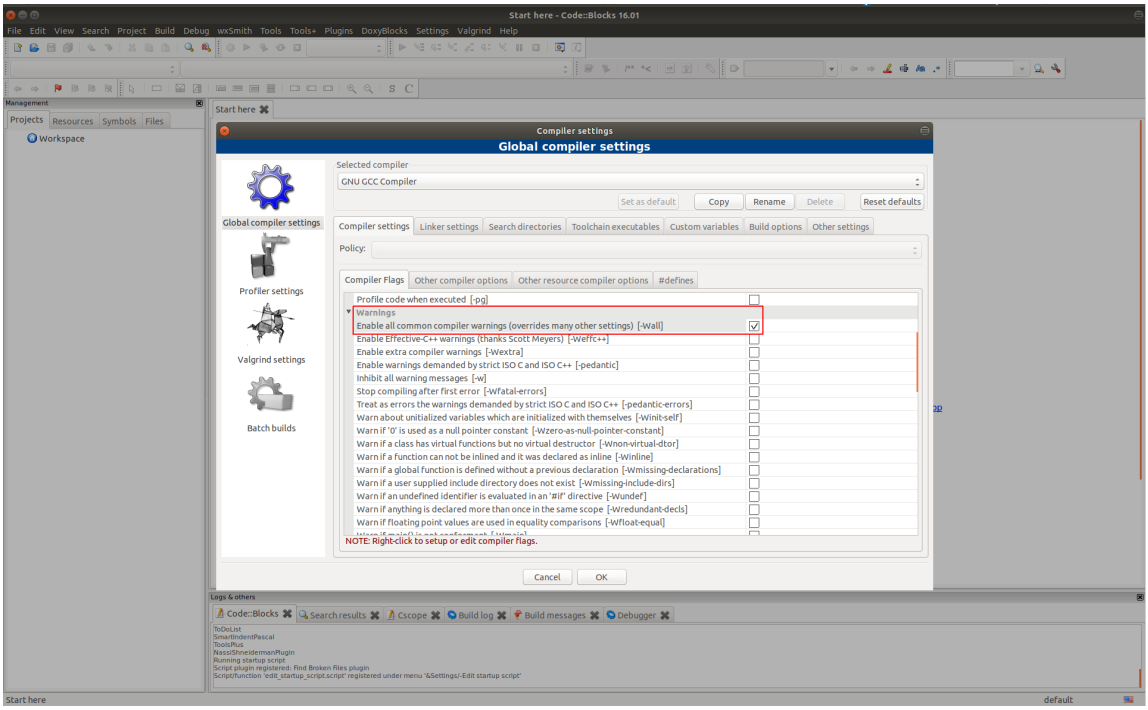
Po uruchomieniu środowiska należy zamknąć okno z poradą dnia, a następnie z górnego menu wybrać opcje `Settings` → `Compiler…` (Rysunek 1).



Rysunek 1: Otwarcie okna ustawień

W sekcji `Warnings` należy zaznaczyć pole „Enable all common compiler warnings (overrides many other settings) [-Wall]” i kliknąć przycisk OK (Rysunek 2). Wybrana opcja będzie zaznaczona po ponownym uruchomieniu środowiska. Nakazuje ona kompilatorowi dokładniejsze sprawdzanie kompilowanego programu pod względem zgodności ze składnią. Jeśli kompilator wykryje jakieś rozbieżności z gramatyką języka, to wygeneruje ostrzeżenie (ang. *warning*). Jest ono informacją od kompilatora, że napotkał niekrytyczną usterkę w kodzie źródłowym, ale mimo to jest w stanie utworzyć na jego podstawie postać wykonywalną programu. Najczęściej ta usterka związana jest z wieloznacznością zapisu fragmentu programu i skutkuje tym, że kompilator musi przyjąć pewne założenia, które nie muszą być zgodne z zamiarami programisty. W takiej sytuacji program wynikowy może działać inaczej niż zakładał jego twórca. Błąd w kodzie źródłowym (ang. *error*), a dokładniej *błąd składni*, oznacza całkowitą niezgodność z gramatyką języka, powoduje natychmiastowe przerwanie kompilacji i plik z programem wykonywalnym (kod wynikowy) nie powstaje.

Aby program działał poprawnie należy poprawić wszystkie jego fragmenty powodujące generowanie przez kompilator ostrzeżeń, a także usunąć błędy składniowe i *logiczne*. Zazwyczaj wyeliminowanie dwóch pierwszych rodzajów defektów nie nastręcza większych trudności. Prawdziwym problemem są błędy logiczne, ponieważ nie są one wykrywane przez kompilator, a powodują nieprawidłowe działanie



Rysunek 2: Okno ustawień z zaznaczoną opcją `-Wall`

programu. Różnicę między błędami składniowymi i logicznymi, nazywanymi również *semantycznymi*, wyjaśnię przy pomocy przykładu. W matematyce zapis $2\ i\ 2 = 4$ uznany zostałby za zawierający błąd składniowy, bo znaczenie litery „i" nie jest określone. Z kolei wyrażenie $2+2 = 5$ jest zapisane poprawnie (wszystkie symbole są znane), ale jest błędne z punktu widzenia arytmetyki. Niestety błędy semantyczne w programach nie są takie łatwe do zlokalizowania.

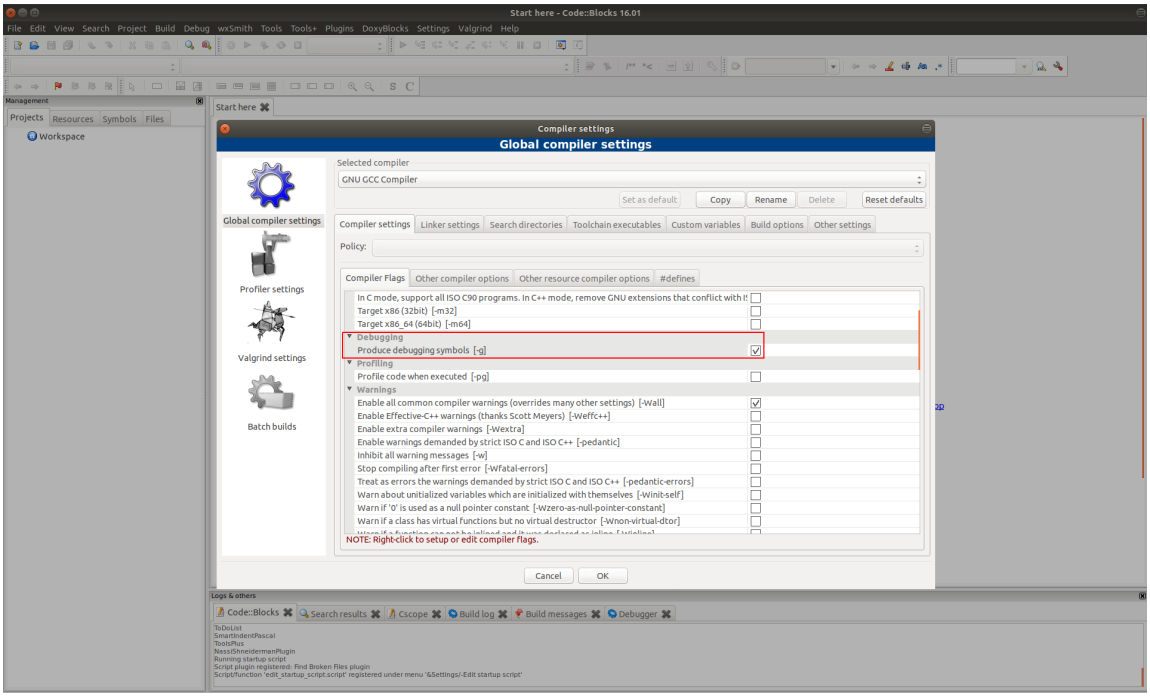
Środowisko Code::Blocks ułatwia programiście znajdywanie błędów logicznych w programie dzięki współpracy z programem nazywanym debuggerem. Aby w pełni wykorzystać jego możliwości należy kompilatorowi nakazać umieszczenie dodatkowych informacji w kodzie wynikowym kompilowanego programu. Można to zrobić w oknie ustawień zaznaczając opcję „Produce debugging symbols [-g]" (Rysunek 3) i klikając przycisk OK.

Dla Code::Blocks opracowano kilka pomocnych wtyczek (ang. *plugins*). Jedną z nich jest „AStyle", która pozwala poprawić formatowanie kodu pisanego programu. Umożliwia ona wybór kilku różnych formatów. W przykładach z wykładu stosowany jest styl K&R. Styl formatowania w Code::Blocks można określić wybierając z menu głównego `Settings` → `Editor`, co spowoduje otwarcie okna „Configure editor". Należy przewinąć listę znajdującą się w jego lewej części, odnaleźć pozycję „Source formatter" i ją kliknąć. Następnie w zakładce „Style" zaznaczyć opcję K&R, a następnie kliknąć przycisk OK. Jeśli kod programu poprawnie się kompiluje, to można poprawić jego formatowanie, zaznaczając go kombinacją klawiszy `Ctrl+A`, a następnie wybierając z menu głównego `Plugins` → `Source code formatter (AStyle)`.

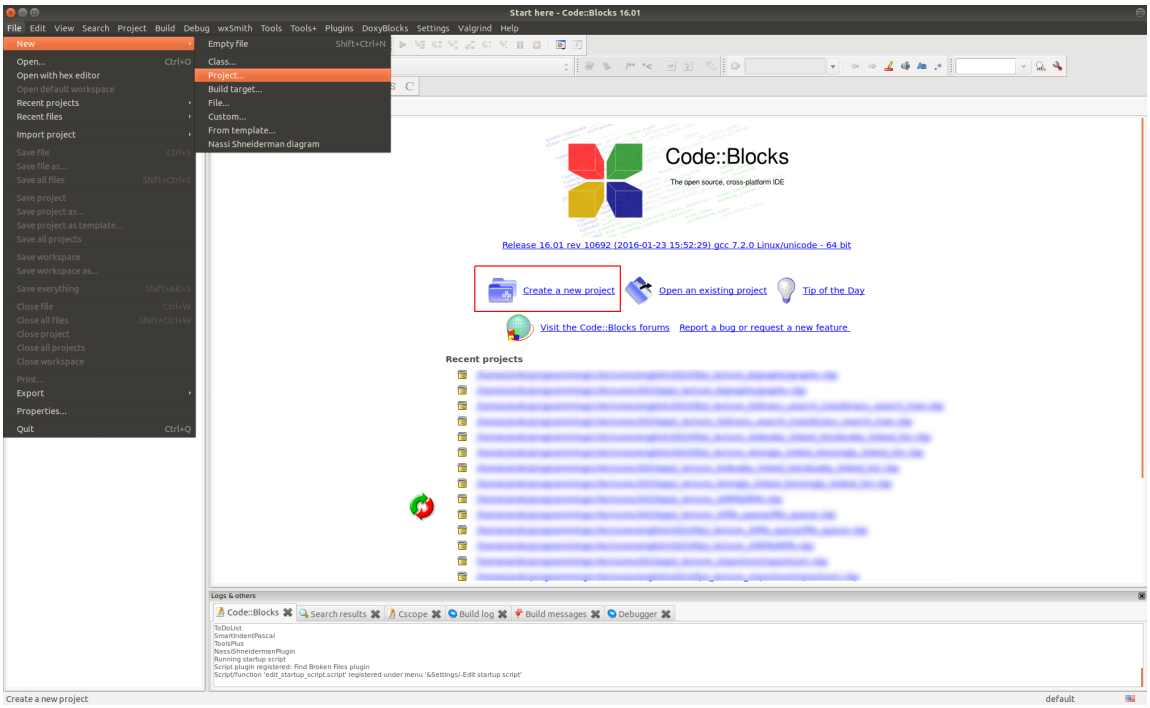
3. Tworzenie nowego projektu

Programy w środowisku Code::Blocks tworzone są w ramach projektów. Nowy projekt można utworzyć na kilka sposobów. Najprostszy to kliknięcie łącza „Create a new project" w głównym oknie aplikacji (Rysunek 4) . Można również w górnym menu wybrać następujący ciąg opcji `File` → `New` → `Project…` (Rysunek 4).

W oknie dialogowym, które się ukaże należy kliknąć ikonę „Console application" (Rysunek 5), następnie wskazać język C (**nie** C++) i kliknąć przycisk „Next" (Rysunek 6). W nowym formularzu należy podać nazwę projektu (Rysunek 7). Ponieważ na jej podstawie tworzone są nazwy plików, to nie zale-

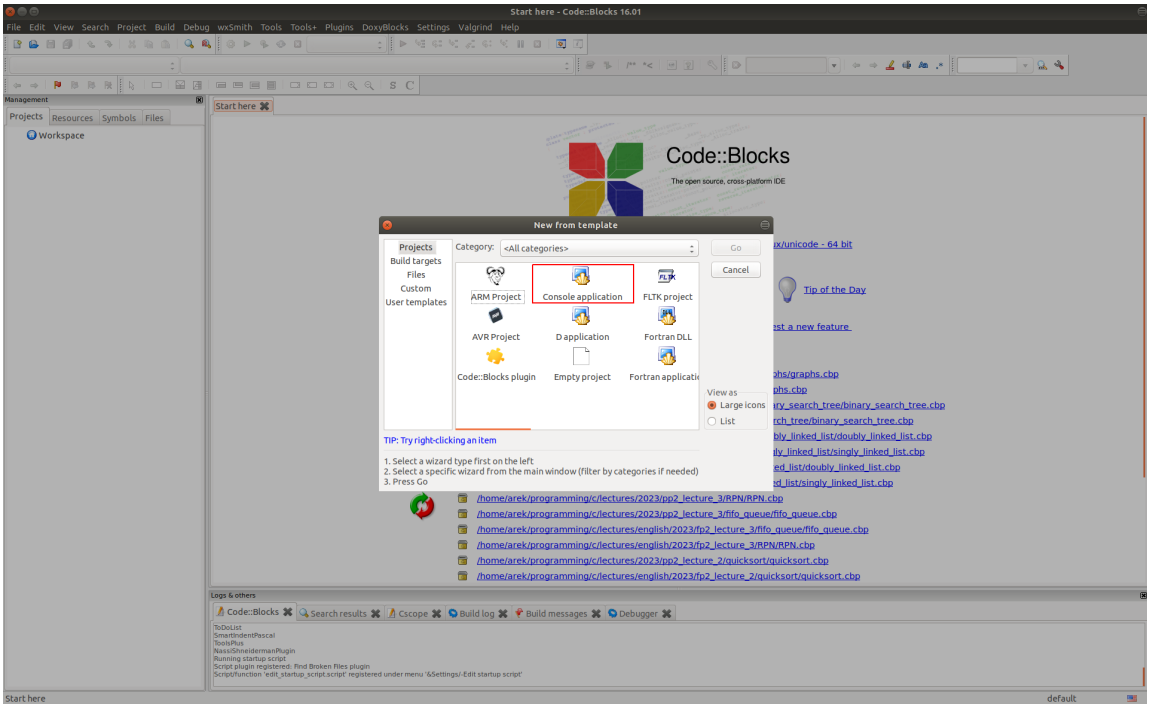


Rysunek 3: Okno ustawień z zaznaczoną opcją `-g`

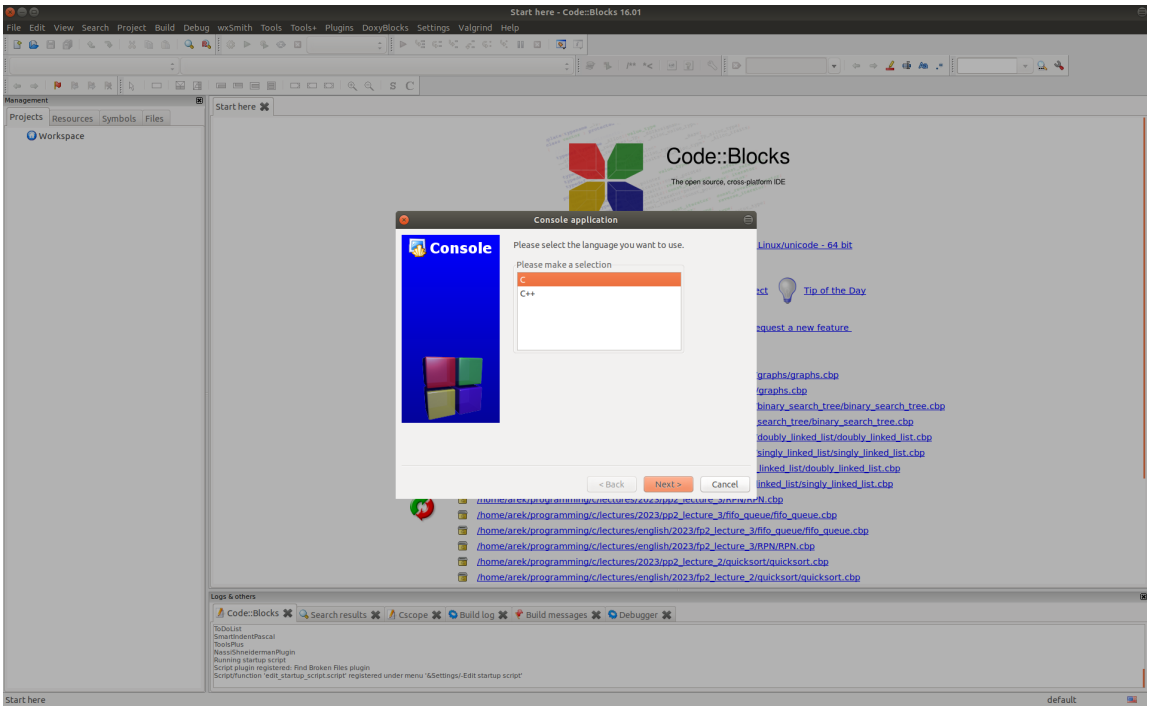


Rysunek 4: Tworzenie nowego projektu.

ca się stosowania w niej spacji. Zamiast nich można użyć znaków podkreślenia (_). Po wprowadzeniu nazwy należy ponownie kliknąć przycisk „Next”, a następnie, w nowym oknie, „Finish”. W oknie głównym, na liście rozwijanej po prawej stronie, stanie się widoczna ikona z nazwą projektu, a poniżej niej ikona z katalogiem o nazwie „Sources”. Po jej kliknięciu pojawi się ikona z plikiem o nazwie `main.c`.

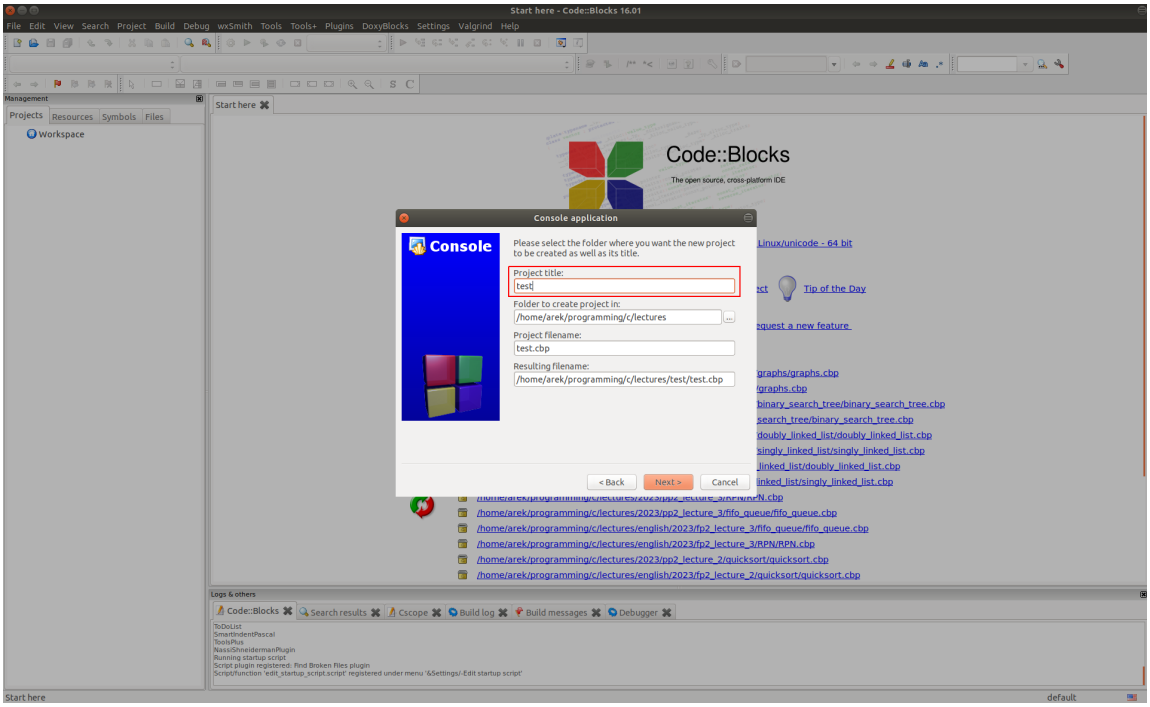


Rysunek 5: Wybór rodzaju programu

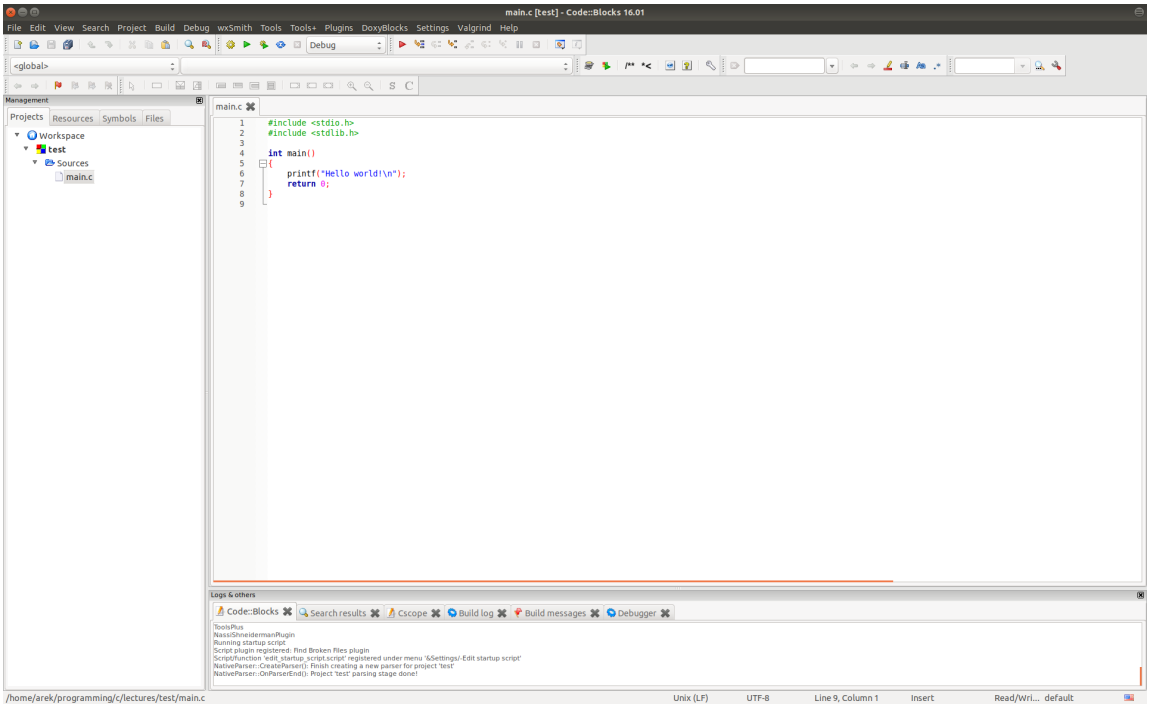


Rysunek 6: Wybór języka

Pliki z rozszerzeniem `.c` są plikami tekstowymi, ale zawierają kod źródłowy programu zapisany w języku C. Nazwa pliku może być dowolna, ale środowisko Code::Blocks zwyczajowo stosuję nazwę `main`. Po jej kliknięciu, w oknie głównym środowiska, czyli oknie edytora, pojawi się zawartość tego pliku - program typu „Hello World!" (Rysunek 8).



Rysunek 7: Definiowanie nazwy projektu (w tym przypadku *test*)



Rysunek 8: Główne okno środowiska z widocznym kodem źródłowym programu

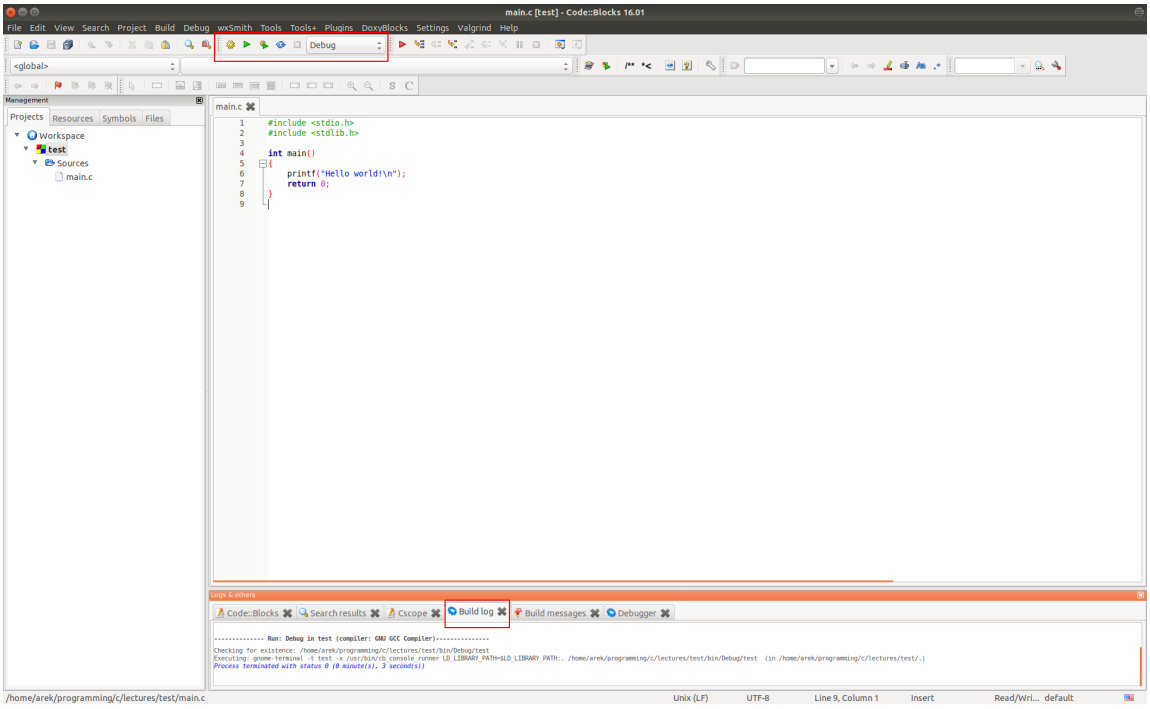
Został on automatycznie dodany z szablonu przez środowisko i ma trochę inną postać niż ta, którą podano na wykładzie. Do wypisywania komunikatu użyto w nim funkcji `printf()` zamiast `puts()`, stąd konieczność dodania na końcu komunikatu znaku nowego wiersza (`\n`), aby po wypisaniu komunikatu kursor przeszedł do kolejnego wiersza na ekranie. Nadmiarowo włączono w tym programie plik nagłów-

kowy o nazwie `stdlib.h`. Jeśli usunie się instrukcję `#include<stdlib.h>`, to program nadal będzie kompilował się bez ostrzeżeń i działał poprawnie. Funkcja `main()` ma pustą listę parametrów, brakuje w niej słowa kluczowego `void`. Zatem proponowany przez Code::Blocks zapis tego programu nie jest najlepszy, ale kompiluje się i uruchamia.

Proszę zwrócić uwagę, że wiersze kodu źródłowego w głównym oknie, które jest jednocześnie oknem edycji kodu źródłowego, są ponumerowane. Te numery nie są częścią kodu źródłowego. Są one dodawane przez środowisko celem ułatwienia pracy programistom.

4. Kompilacja i uruchomienie programu

Środowisko Code::Blocks pozwala program tylko skompilować, uruchomić (jeśli wcześniej został skompilowany) lub jednocześnie uruchomić i skompilować. Istnieje wiele sposobów uruchomienia tych czynności. Zalecam opanowanie skrótów klawiszowych, które je aktywują. Jest to najbardziej ergonomiczny i ekonomiczny sposób pracy ze środowiskiem. Kompilacja kodu źródłowego uruchamiana jest kombinacją klawiszy `Ctrl+F9`. Jeśli program skompiluje się poprawnie, to można go uruchomić naciskając `Ctrl+F10`. Obie te czynności można wykonać jednocześnie za pomocą naciśnięcia klawisza `F9`. Program uruchomi się, jeśli jego kompilacja przebiegnie prawidłowo. W przypadku wybranego rodzaju programów pojawi się okno terminala, umożliwiające interakcję z uruchomionym programem lub zawierające jego wynik. Code::Blocks ma również pasek narzędziowy, zawierający ikony aktywujące opisywane czynności (Rysunek 9, u góry). Pierwsza od lewej ikona w tym pasku (żółty tryb) rozpoczyna kompilację, druga (zielony

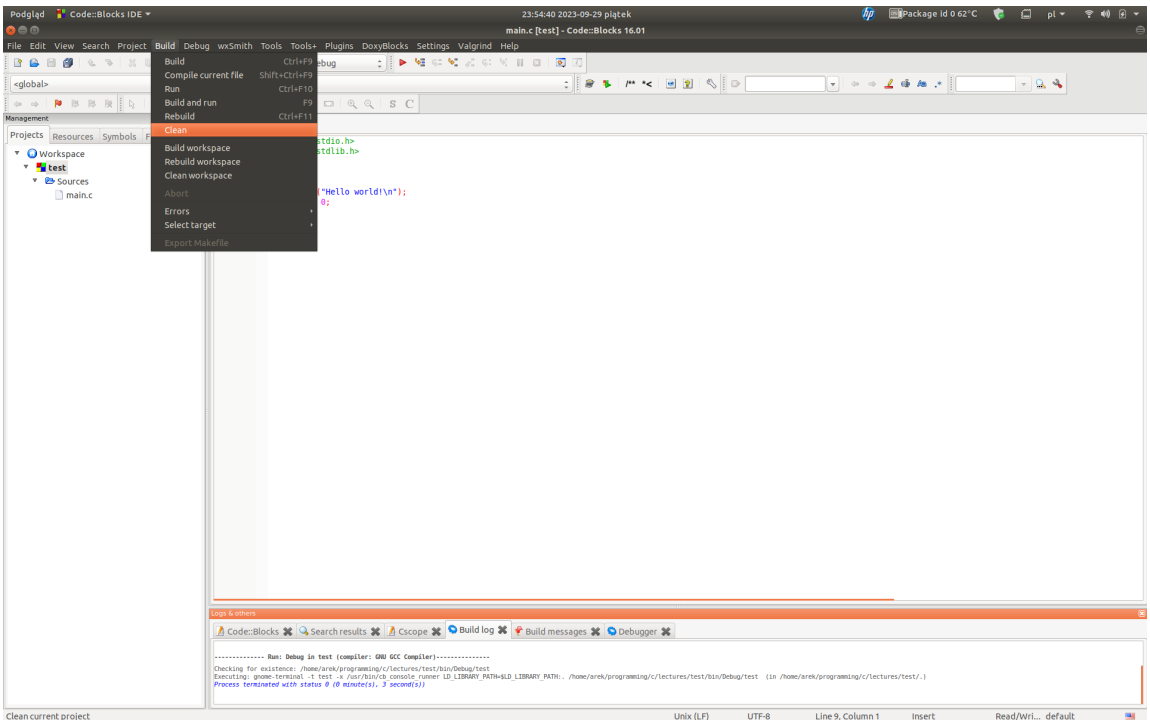


Rysunek 9: Pasek narzędzi kompilacji i uruchamiania oraz zakładka „Build log”

trójkąt) uruchamia program, a trzecia (zielony trójkąt na tle żółtego trybu) wykonuje obie te czynności. Dalej znajduje się ikona, która jest szara, zatem nieaktywna. Można jej użyć, kiedy wykonywany jest skompilowany wcześniej program. Pozawala ona go zakończyć w trybie awaryjnym i ma wtedy wygląd iksa na czerwonym tle. Proszę zwrócić uwagę zakładkę „Build log” (Rysunek 9, na dole). Zawiera ona komunikaty kompilatora, z którymi warto się zapoznać i sprawdzić, czy nie wygenerował on ostrzeżeń lub komunikatów o błędach. Może to wymagać „przewinięcia” jej zawartości.

Więcej opcji związanych z kompilacją i uruchamianiem programu można znaleźć w pozycji „Build” menu górnego środowiska. Wśród nich przydatna może się okazać „Clean”, która usuwa plik powstające

w procesie kompilacji programu, w tym plik wykonywalny (Rysunek 10). Dzięki temu można zwolnić trochę miejsca na nośniku.



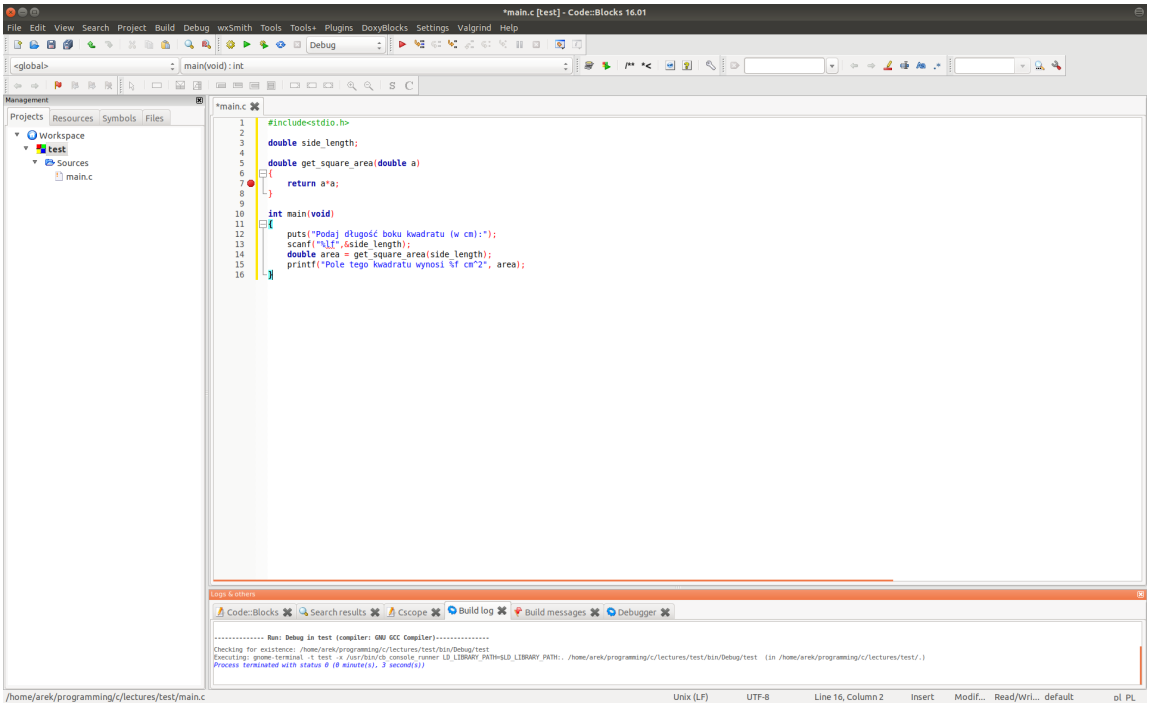
Rysunek 10: Opcja „Clean”

5. Debugowanie programu

Środowisko Code::Blocks umożliwia krokowe (tj. wiersz po wierszu) wykonanie programu i obserwację wartości zmiennych. Taki sposób wykonania programu jest przydatny do ustalenia miejsca, w którym występuje błąd. Ten tryb pracy można uruchomić dwojako, ale najpierw trzeba się upewnić, że projekt jest w perspektywie „Debug” (Rysunek 9, u góry), bo tylko wtedy kompilator doda do programu wynikowego informacje potrzebne dla debuggera, który odpowiedzialny jest za krokowe wykonanie. Pierwszy sposób polega na umieszczeniu kursora w miejscu od którego chcemy śledzić wykonanie programu i naciśnięciu klawisza `F4`. Program wykona się do tego miejsca i zatrzyma. Inny sposób polega na użyciu klawisza `F5` zamiast `F4`. Ustawia on w miejscu umieszczenia kursora pułapkę (ang. *breakpoint*), co oznacza, że program będzie się zatrzymywał zawsze w tym miejscu do czasu zdjęcia pułapki. Do tej ostatniej czynności można ponownie użyć klawisza `F5`. Rysunek 11 pokazuje kod źródłowy prostego programu z funkcją, który oblicza pole kwadratu, o boku, którego długość jest podawana przez użytkownika. Jest w nim założona pułapka w wierszu nr 7.

Jeśli w programie jest więcej zdefiniowanych pułapek to można je usunąć wybierając z menu górnego następujące opcje `Debug` → `Remove all breakpoints`. Aby rozpocząć wykonanie programu z pułapkami należy nacisnąć klawisz `F8`. Po zatrzymaniu wykonania programu możemy wykonywać kolejne instrukcje naciskając sukcesywnie klawisz `F7`. Jeśli chcemy zbadać działanie funkcji zdefiniowanej w programie[1] to musimy użyć następującej opcji z menu górnego: `Debug` → `Step into`[2]. Aby zbadać jak zmieniają się wartości zmiennych należy otworzyć okno śledzenia klikając następujące opcje górnego menu: `Debug` → `Debugging windows` → `Watches`. To okno można osadzić w dolnej części środowiska. Aby śledzić zmianę wartości konkretnej zmiennej należy wpisać jej nazwę w pierwszej kolumnie tabeli w oknie `Watches` i nacisnąć klawisz `Enter`. W ostatniej kolumnie tabeli pojawi się typ tej zmiennej (jeśli ona istnieje i jest

[1] Funkcja to wydzielony fragment programu, nazywany także podprogramem.
[2] Istnieje skrót klawiszowy dla tej opcji: `Shift+F7`, ale nie we wszystkich wersjach środowiska działa on poprawnie.



Rysunek 11: Ustawienie pułapki

widoczna w tej części programu, która bieżąco jest wykonywana), a w środkowej jej obecna wartość, która będzie się zmieniała w trakcie krokowego wykonania programu.

Rysunek 12 pokazuje wykonanie programu obliczającego pole kwadratu w trybie krokowym. Żółty trójkąt w siódmym wierszu, który nałożony jest na czerwoną kropkę symbolizującą pułapkę, oznacza, że w ten fragment kodu zostanie wykonany jako następny po naciśnięciu klawisza `F7`. Na dole rysunku, po prawej stronie zaznaczone jest okno `Watches` w którym widoczne są nazwy trzech zmiennych. Zmienna o nazwie „a" została dodana automatycznie przez środowisko. Jest to parametr funkcji `get_square_area()`. W środkowej kolumnie widoczna jest jego bieżąca wartość. Zmienne `side_length` i `area` zostały dodane przeze mnie ręcznie. Pierwsza jest zmienną globalną[3] i w oknie wyświetlana jest nie tylko jej wartość, ale także typ. Druga jest zmienną lokalna funkcji `main()`. Ponieważ ta funkcja w tej chwili nie jest wykonywana, to w oknie jest komunikat, że zmienna z nią powiązana nie jest dostępna.

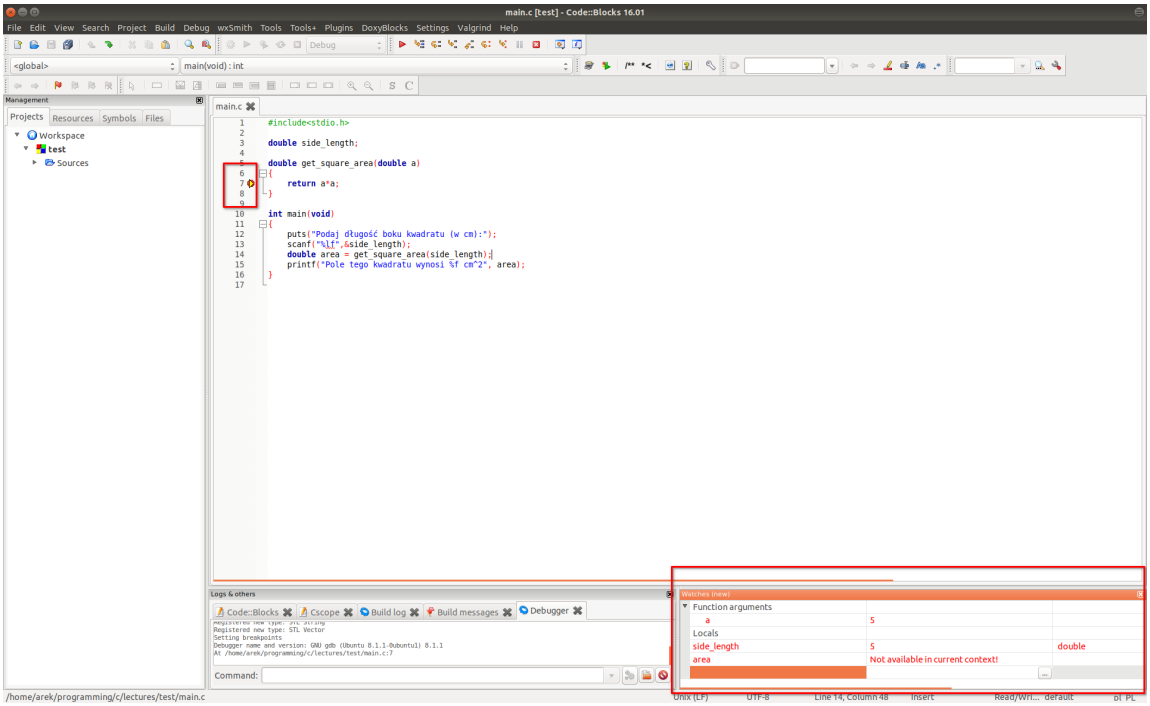
6. Przykładowy program

Listing 1 zawiera kod źródłowy programu, w którym zdefiniowano funkcję o nazwie `calculate_area()`, obliczającą pole koła. Ta funkcja ma jeden parametr wejściowy typu `double`, przez który przekazywana jest długość promienia, a zwraca wartość pola. Typ wartości zwracanej i parametru określa, że obie te liczby będą liczbami rzeczywistymi[4]. Funkcja ta jest wywoływana w głównej funkcji programu o nazwie `main()`. W tej funkcji program pyta użytkownika o długość promienia, a następnie przy pomocy funkcji `scanf()` zapisuje ją w zmiennej `radius`. Jeśli długość promienia jest nieujemna to wartość tej zmiennej jest przekazywana do funkcji `calculate_area()`, która zwraca obliczoną wielkość pola do wywołania funkcji `printf()`. Ta ostania wypisuje na ekranie komunikat, wielkość promienia koła oraz jego pole[5]. Jeśli promień miałby „ujemną długość", to program wypisze komunikat o błędzie. Te

[3] Środowisko umieściło ją w sekcji „Local" okna `Watches`, ale w tym wypadku oznacza to, że ta zmienna jest elementem programu zapisanym w pliku *main.c*.

[4] Dokładniej są to liczby zmiennoprzecinkowe, które stanowią dosyć dobrą reprezentację liczb rzeczywistych w pamięci komputera.

[5] *Uwaga*, w tym miejscu, w listingu widoczna jest zawinięta strzałka. Nie jest ona częścią programu, tylko oznacza, że ten wiersz kodu zawierał zbyt długi tekst, który jest kontynuowany w następnym wierszu.



Rysunek 12: Wykonanie programu w trybie krokowym

dwa przypadku są rozróżniane dzięki zastosowaniu instrukcji warunkowej `if`. **Uwaga!** W kodzie programu celowo wprowadzono dwa błędy. Jeden z nich jest błędem składni i ujawnia się podczas kompilacji, a drugi w trakcie wykonania programu.

Listing 1: Kod źródłowy programu obliczającego pole koła o zadanym promieniu.

```
#include<stdio.h>

#define PI 3.1415

double calculate_area(double r)
{
        return PI*r*r;
}

double radius;

int main(void)
{
        puts("Podaj długość promienia koła");
        scanf("%lf",&radius);
        //Program sprawdza, czy promień ma wartość większą lub równą zero.
        if(radius<=0.0)
                printf("Pole koła o promieniu %f cm wynosi %f
↪   cm^2.\n",radius,calculate_area(radius))
        else
                puts("Podana długość promienia jest nieprawidłowa.");
        return 0;
}
```

7. Zadania

1. Zapoznaj się z treścią tej instrukcji. Naucz się podstawowej obsługi środowiska Code::Blocks.
2. Popraw błąd kompilacji w programie z listingu 1.
3. Znajdź i usuń błąd, który pojawia się w trakcie wykonania programu. Użyj do tego techniki debugowania.
4. Sprawdź jakie inne opcje ułatwiające programowanie oferuje środowisko Code::Blocks. Postaraj się nauczyć niektórych z nich.