

Podstawy programowania

Laboratorium 1

Anna Prusinowska

Wprowadzenie

Język C++

Język programowania – co to takiego

C++ jako język programowania

- Ludzie komunikują się ze sobą m. in. dzięki językowi.
- Każdy język ludzki składa się ze zbioru znaków (np. słów) oraz gramatyki (zasad łączenia znaków w komunikaty).
- Podobnie rzecz ma się w przypadku języków programowania. Są one zbiorami zasad, które określają, kiedy ciąg symboli tworzy program komputerowy oraz jakie obliczenia opisuje.

Język C++

Definicja

- język programowania ogólnego przeznaczenia;
- zaprojektowany przez Bjarne Stroustrupa jako rozszerzenie języka C o obiektowe mechanizmy abstrakcji danych i silną statyczną kontrolę typów;
- umożliwia abstrakcję danych oraz stosowanie kilku paradygmatów programowania: proceduralnego, obiektowego i generycznego, a także funkcyjnego i modularnego;
- charakteryzuje się wysoką wydajnością kodu wynikowego, bezpośrednim dostępem do zasobów sprzętowych i funkcji systemowych, łatwością tworzenia i korzystania z bibliotek (napisanych w C++, C lub innych językach), niezależnością od konkretnej platformy sprzętowej lub systemowej (co gwarantuje wysoką przenośność kodów źródłowych) oraz niewielkim środowiskiem uruchomieniowym.

Algorytm

Definicja
nieformalna

Algorytm a program

Algorytm - [nieformalnie] sposób postępowania (przepis), umożliwiający rozwiązanie określonego zadania, podany w postaci skończonego zestawu czynności do wykonania, ze wskazaniem ich następstwa.

Sposoby zapisu algorytmu: opis słowny, lista kroków, schemat blokowy, drzewo algorytmu, pseudokod, język programowania.

Program - formalnie spisana wersja algorytmu.

Zmienne

Definicja

Zmienne

- Praktycznie każdy algorytm (a więc i program komputerowy) musi pamiętać pewne informacje np. program komputerowy, obsługujący bankomat, musi pamiętać, ile razy wpisano błędny PIN, aby być w stanie zablokować kartę.
- W celu zapamiętania informacji programista używa **zmiennych** (ang. variable), które przechowują liczby, znaki, napisy, lub wręcz obiekty. Przechowywana zawartość może ulec zmianie - stąd nazwa.

Zmienna – konstrukcja programistyczna posiadająca trzy podstawowe atrybuty: symboliczną nazwę, miejsce przechowywania i wartość; pozwalająca w kodzie źródłowym odwoływać się przy pomocy nazwy do wartości lub miejsca przechowywania.

Nazwy zmiennych

W języku C++ nazwą zmiennej może być dowolnie długi ciąg liter, cyfr i znaków podkreślenia (małe i duże litery są rozróżniane). Nazwą nie może być słowo kluczowe z poniższej tabeli:

asm	auto	break	case	catch	char
class	const	continue	default	delete	do
double	else	enum	extern	float	for
friend	goto	if	inline	int	long
new	operator	private	protected	public	register
return	short	signed	sizeof	static	struct
switch	template	this	throw	try	typedef
union	unsigned	virtual	void	volatile	while

Typy zmiennych

- Informacja zapisana w pamięci komputera to ciąg "0" i "1". Zmiennym nadaje się typ danych, aby możliwe było ustalenie, czym te dane są.

Najczęściej używane typy podstawowe:

Typ	Rozmiar	Komentarz
<code>int</code>	4 bajty	Wybór domyślny dla wartości całkowitych.
<code>double</code>	8 bajtów	Wybór domyślny dla wartości zmiennopozycyjnych.
<code>bool</code>	1 bajt	Przedstawia wartości, które mogą być prawdziwe lub fałszywe.
<code>char</code>	1 bajt	Używaj dla znaków ASCII w starszych ciągach typu C lub obiektach <code>std::string</code> , które nigdy nie będą musiały zostać skonwertowane na UNICODE.
<code>wchar_t</code>	2 bajty	Przedstawia wartości znaku „szerokiego”, które mogą być zakodowane w formacie UNICODE (UTF-16 w systemie Windows, inne systemy operacyjne mogą się różnić). Jest to typ znaku używany w ciągach typu <code>std::wstring</code> .
<code>unsigned char</code>	1 bajt	Język C++ nie ma wbudowanego typu bajtowego. Użyj <code>unsigned char</code> , aby reprezentować wartość bajtu.
<code>unsigned int</code>	4 bajty	Wybór domyślny dla flag bitowych.
<code>long long</code>	8 bajtów	Przedstawia bardzo duże wartości całkowitoliczbowe.

Podział typów języka C++

Pierwszy podział:

- a) **typy fundamentalne** np: int, short, long, long long, char, float, double, long double;
- b) **typy pochodne**, które powstają na bazie typów fundamentalnych np: tablice, funkcje, wskaźniki, struktury.

Drugi podział:

- a) **typy wbudowane**, czyli takie, w które język C++ jest wyposażony;
- b) **typy zdefiniowane** przez użytkownika.

Typ *auto*

Od C++11 (od 2011 r.) wprowadzono tzw. automatyczną dedukcję typu, czyli ty *auto*. W określonych sytuacjach można zastąpić konkretny typ słowem kluczowym *auto*, a kompilator określi automatycznie typ zmiennej.

Można go używać jedynie w sytuacji, gdy z deklaracją następuje inicjalizacja zmiennej wartością, tj. wtedy, gdy wpisując wartość określonego typu pozwalamy kompilatorowi „domyślić się”, jaki to typ.

```
auto number = 4 // typ int
```

Instrukcja przypisania

Definicja

Instrukcja przypisania

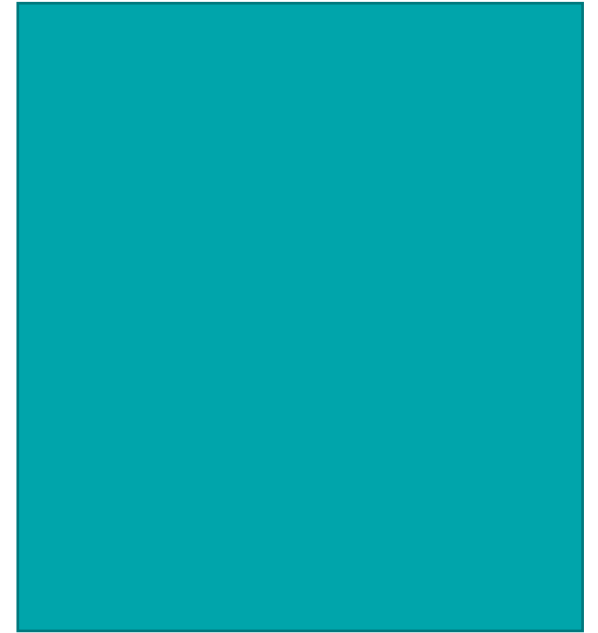
Instrukcja przypisania - rodzaj przypisania, stanowiący odrębną instrukcję w kodzie źródłowym, w której do pewnej lokacji (**l-wartości**) przypisuje się wartość, która będzie w niej przechowywana.

Operator przypisania (=)

```
int x = 10
```

Zmiennej x przypisana jest wartość 10. Operator przypisania działa od prawej do lewej strony czyli zmiennej po lewej stronie przypisana jest wartość lub wyrażenie po prawej stronie.

Budowa programu w języku C++



Element programu	Wyjaśnienie
main	funkcja, od której zaczyna się wykonywanie każdego programu w języku C++ (jest w każdym programie w tym języku)
{}	nawiasy klamrowe, obejmują ciało funkcji, instrukcje wykonywane w ramach niej
cout	instrukcja wyświetlająca tekst na urządzeniu wyjściowym (ekranie)
<<	akcja, która ma zostać podjęta np. cout << wprowadza na ekran tekst
#include <iostream.h>	pozwala na skorzystanie z jednej ze standardowych bibliotek - biblioteki iostream, w które zazwyczaj wyposażone są kompilatory; biblioteka ta zawiera podprogramy odpowiedzialne za np. operacje wejścia/wyjścia (w tym miejscu kompilator wstawia plik nagłówkowy biblioteki iostream)
using namespace std;	polecenie, które nakazuje użycia standardowej przestrzeni nazw std; pojęcie przestrzeni nazw służy do określenia, które zmienne i funkcje można użyć w danym miejscu i zostało wprowadzone po to, aby zapobiegać nakładaniu się nazw zmiennych, funkcji itp.;
;	koniec instrukcji
kompilator	pozwala na przetłumaczenie zapisanego kodu na język maszyny; program poddaje się kompilacji i otrzymuje wersję skompilowaną
linker	proces łączenia skompilowanej wersji programu z bibliotekami (treścią funkcji bibliotecznych)
\n, endl	znak new line - przejście do nowej linii
//, /* */	komentarze; teksty zupełnie ignorowane przez kompilator
cin	console input; instrukcja umożliwiająca wczytanie danych z klawiatury
>>	akcja, która ma zostać podjęta, np. pobranie danych z klawiatury

Przykład 1a

Wypisywanie
na konsolę

Przykład 1a: Wypisywanie w konsoli nazwy przedmiotu

```
#include <iostream>
using namespace std;

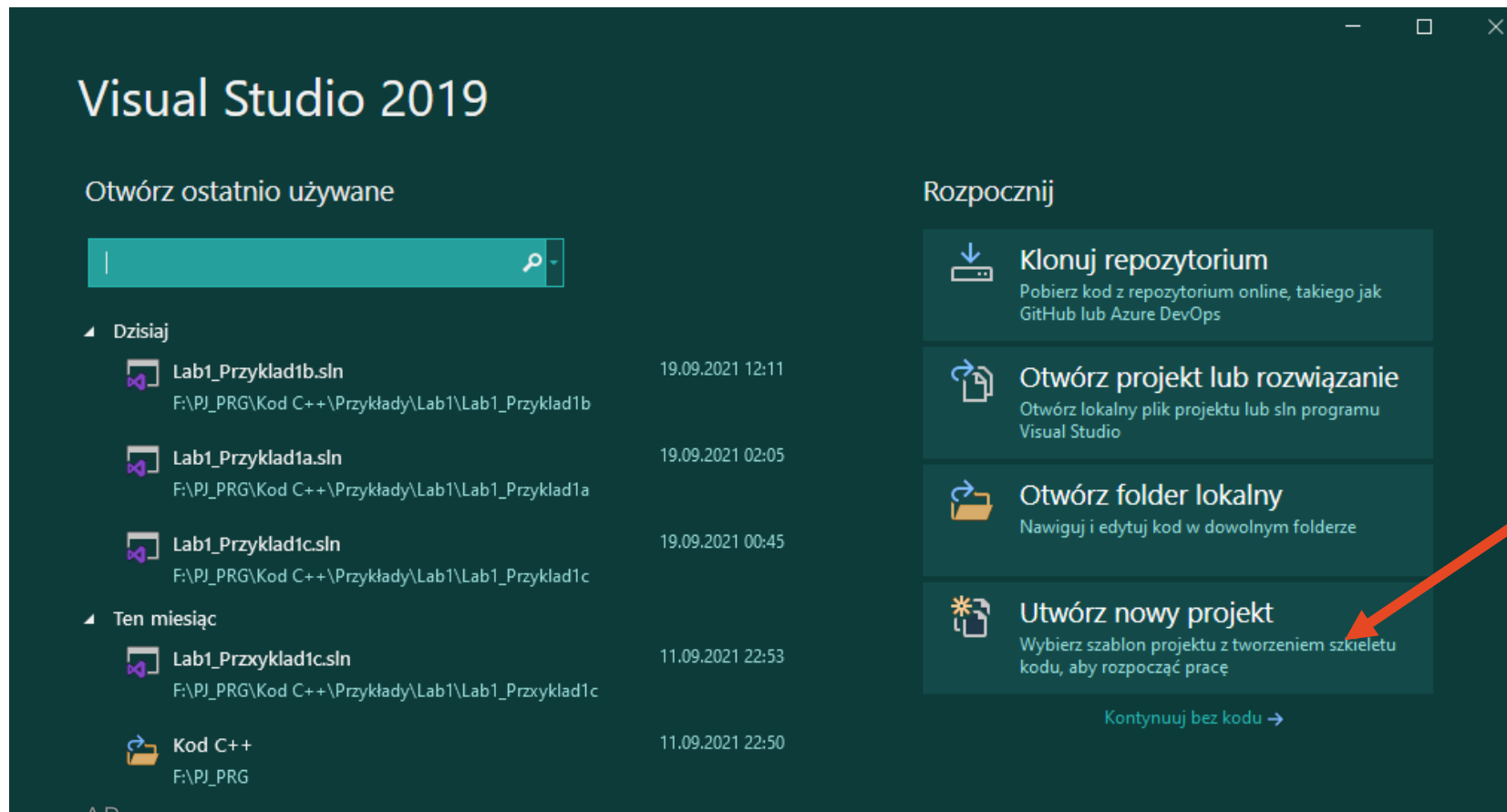
int main()
{
    cout << "Podstawy programowania\n";
    return 0;
}
```

Konsola debugowania programu Microsoft Visual Studio

Podstawy programowania

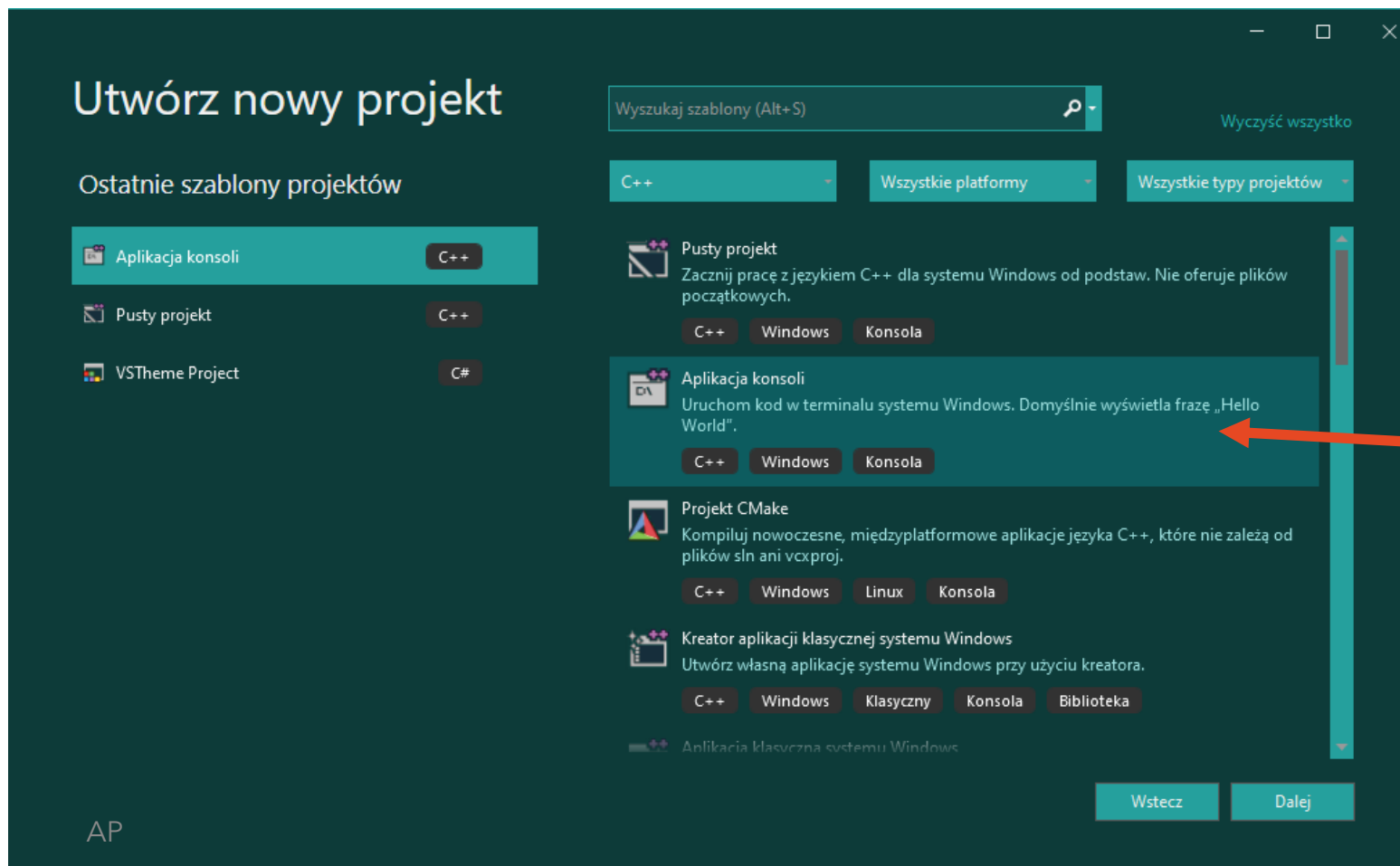
F:\PJ_PRG\Kod C++\Przykłady\Lab1\Lab1_Przykład1a\Debug\Lab1_Przykład1a.exe (proces 25584) zakończono z kodem 0.
Aby automatycznie zamknąć konsolę po zatrzymaniu debugowania, włącz opcję Narzędzia -> Opcje -> Debugowanie -> Automatycznie zamknij konsolę po zatrzymaniu debugowania.
Naciśnij dowolny klawisz, aby zamknąć to okno...

Przykład 1a: Tworzenie nowego projektu



Tworzymy nowy projekt

Przykład 1a: Tworzenie nowego projektu



Wybieramy opcję
„Aplikacja konsoli”

Przykład 1a: Tworzenie nowego projektu

Konfiguruj nowy projekt

Aplikacja konsoli C++ Windows Konsola

Nazwa projektu

NazwaProjektu

Lokalizacja

C:\Users\annap\source\repos

Nazwa rozwiązania ⓘ

NazwaProjektu

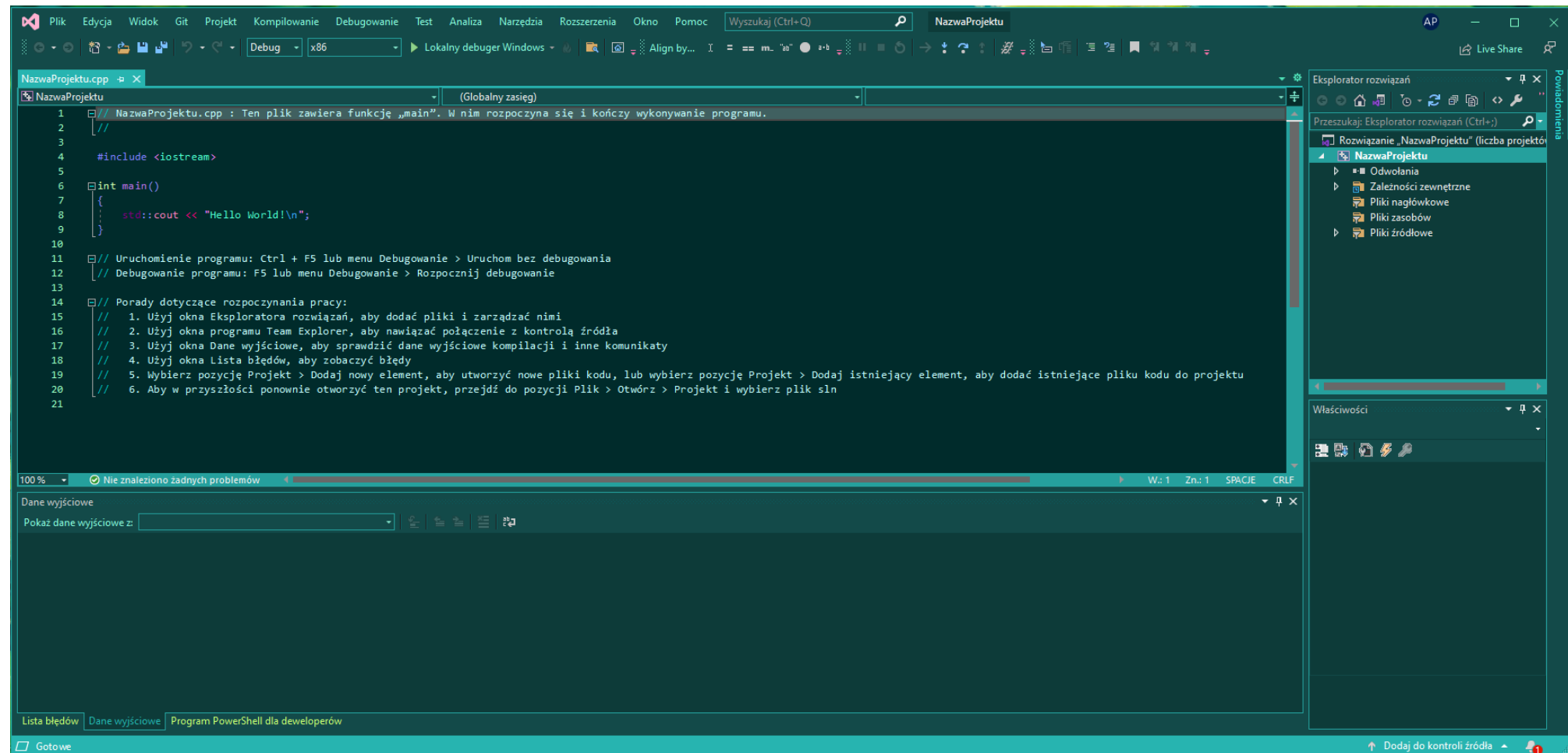
☒ Umieść rozwiązanie i projekt w tym samym katalogu

Wstecz Utwórz

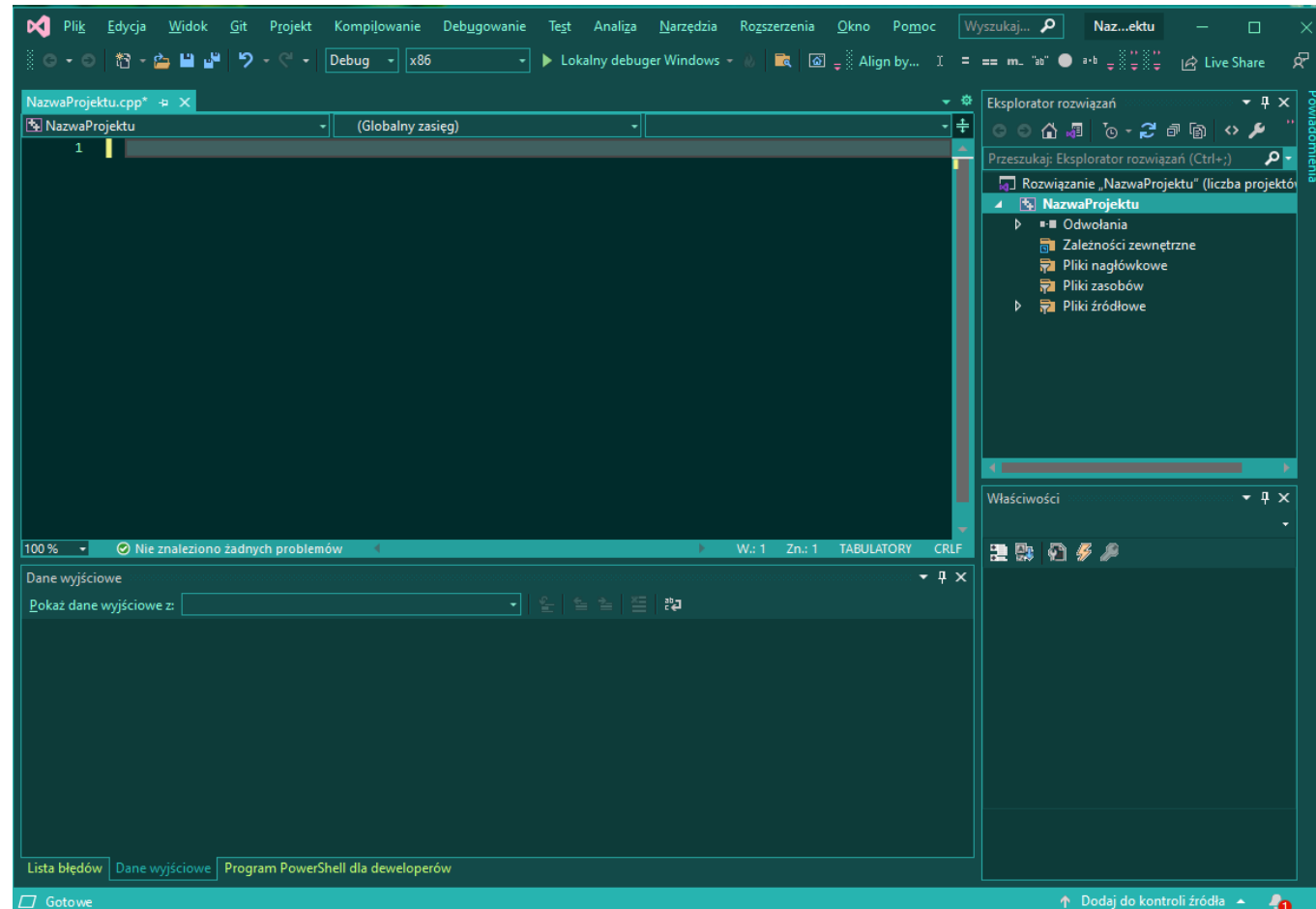
Nadajemy nazwę

Klikamy „Utwórz”

Przykład 1a: Okno nowego projektu



Przykład 1a: Okno nowego projektu - usuwamy automatycznie wygenerowany kod



Przykład 1a: W oknie edytora piszemy kod programu

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Podstawy programowania\n";

    return 0;
}
```



Przykład 1a: W oknie edytora piszemy kod programu

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Podstawy programowania\n";

    return 0;
}
```




pozwała na skorzystanie z jednej ze standardowych bibliotek - biblioteki iostream, w które zazwyczaj wyposażone są kompilatory; biblioteka ta zawiera podprogramy odpowiedzialne za np. operacje wejścia/wyjścia (w tym miejscu kompilator wstawia plik nagłówkowy biblioteki iostream)

Przykład 1a: W oknie edytora piszemy kod programu

```
#include <iostream>
```

```
using namespace std;
```

```
int main()  
{  
    cout << "Podstawy programowania\n";  
    return 0;  
}
```




polecenie, które nakazuje użycia standardowej przestrzeni nazw std; pojęcie przestrzeni nazw służy do określenia, które zmienne i funkcje można użyć w danym miejscu i zostało wprowadzone po to, aby zapobiegać nakładaniu się nazw zmiennych, funkcji itp.;

Przykład 1a: W oknie edytora piszemy kod programu

```
#include <iostream>
```

```
using namespace std;
```

```
int main()   
{  
    cout << "Podstawy programowania\n";  
  
    return 0;  
}
```

funkcja, od której zaczyna się wykonywanie każdego programu w języku C++ (jest w każdym programie w tym języku)

Przykład 1a: W oknie edytora piszemy kod programu

```
#include <iostream>
```

```
using namespace std;
```


```
int main()  
{
```

```
    cout << "Podstawy programowania\n";
```

```
    return 0;
```

```
}
```

instrukcja wyświetlająca tekst na urządzeniu wyjściowym (ekranie)



Przykład 1a: W oknie edytora piszemy kod programu

```
#include <iostream>
```


```
using namespace std;
```

```
int main()  
{
```

```
    cout << "Podstawy programowania\n";
```

```
    return 0;
```

```
}
```



akcja, która ma zostać
podjęta: cout <<
wprowadza na ekran
tekst

Przykład 1a: W oknie edytora piszemy kod programu

```
#include <iostream>
```

```
using namespace std;
```

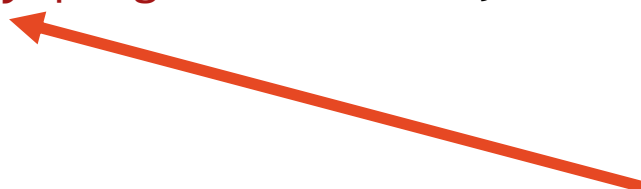
```
int main()
```

```
{
```

```
    cout << "Podstawy programowania\n";
```

```
    return 0;
```

```
}
```



Tekst wypisywany na ekranie - w cudzysłowie " "

Przykład 1a: W oknie edytora piszemy kod programu

```
#include <iostream>
```

```
using namespace std;
```

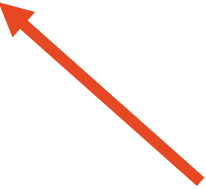
```
int main()
```

```
{
```

```
    cout << "Podstawy programowania\n";
```

```
    return 0;
```

```
}
```



znak new line - przejście do nowej linii

Przykład 1a: W oknie edytora piszemy kod programu

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    cout << "Podstawy programowania\n";
```

```
    return 0;
```

```
}
```

Instrukcja,
przerywająca
wykonanie funkcji,
zwracająca kontrolę
do funkcji
wywołującej

Przykład 1a: W oknie edytora piszemy kod programu

```
#include <iostream>
```

```
using namespace std;
```


```
int main()
```

```
{
```

```
    cout << "Podstawy programowania\n";
```

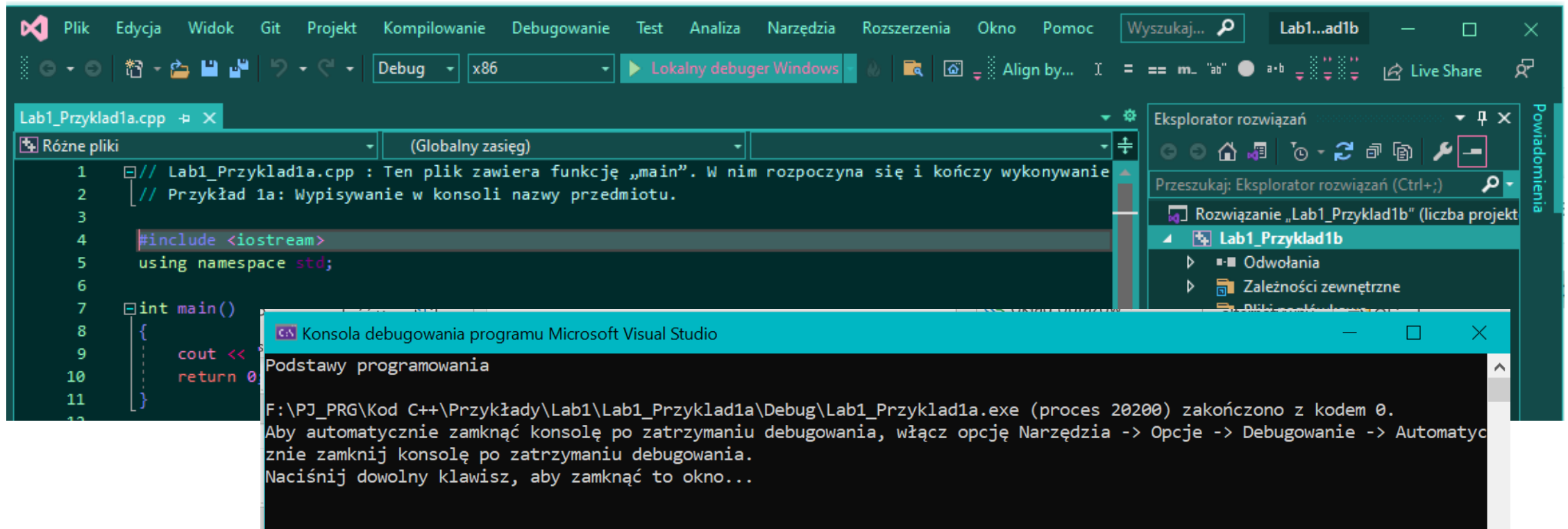
```
    return 0;
```

```
}
```



return 0 oznacza, że program wykonał się pomyślnie i zrobił to, co miał zrobić

Przykład 1a: Uruchamianie aplikacji



Instrukcje sterujące

Instrukcje sterujące - sterują przebiegiem programu; umożliwiają podjęcie decyzji o wykonaniu konkretnych instrukcji w programie. Podejmowane decyzje zależą od wyniku sprawdzenia określonych warunków - ustalenia, czy warunek jest prawdziwy, czy nie.

Podział instrukcji sterujących:

- a) instrukcja warunkowa if/if-else;
- b) instrukcja wyboru switch;
- c) pętle:
 - while;
 - do... while;
 - for;
- d) instrukcja break;
- e) instrukcja continue.

Instrukcje warunkowe

Należą do
instrukcji
sterujących

Instrukcje warunkowe if / if-else

Praktycznie każdy algorytm (a więc i program komputerowy) musi wykonywać się warunkowo.

Przykładowo program komputerowy, obsługujący bankomat, musi:

- zabrać Użytkownikowi pieniądze z konta, tylko POD WARUNKIEM, że pieniądze wypłacono;
- wypisać komunikat "Błędny PIN", tylko POD WARUNKIEM, że podano błędny PIN;
- sprawdzić PIN, tylko JEŚLI podano wymaganą liczbę cyfr.

Instrukcja warunkowa if / if-else

Jeśli (coś) to
w innym przypadku to...

Jeśli będzie padać, zostanę w domu. (Domyślnie: W innej sytuacji wyjdę z domu).

Jeśli będę mieć wysoką gorączkę, to pójdę do lekarza; jeśli stan podgorączkowy – zostanę w domu.

(Domyślnie: W innej sytuacji nie pójdę do lekarza ani nie zostanę w domu – np. pójdę do pracy).

Przykład 1b

Instrukcja
warunkowa

Przykład 1b: Program do sprawdzania, czy liczba całkowita jest dodatnia czy ujemna. Ten program traktuje 0 jako liczbę dodatnią.

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int number;
6
7      cout << "Podaj liczbe calkowita: ";
8      cin >> number;
9      if (number >= 0) {
10         cout << "Wpisales dodatnia liczbe calkowita: " << number << endl;
11     }
12     else {
13         cout << "Wpisales ujemna liczbe calkowita: " << number << endl;
14     }
15     cout << "Dziekuje.";
16     return 0;
17 }
```


Przykład 1b: Przykładowy output

```
C:\> Konsola debugowania programu Microsoft Visual Studio  
Podaj liczbe calkowita: 3  
Wpisales dodatnia liczbe calkowita: 3  
Dziekuje.
```

```
C:\> Konsola debugowania programu Microsoft Visual Studio  
Podaj liczbe calkowita: -5  
Wpisales ujemna liczbe calkowita: -5  
Dziekuje.
```

```
C:\> Konsola debugowania programu Microsoft Visual Studio  
Podaj liczbe calkowita: 0  
Wpisales dodatnia liczbe calkowita: 0  
Dziekuje.
```


Przykład 1c

Instrukcja
warunkowa

Przykład 1c: Napisz program, obliczający pole kwadratu. Pamiętaj, że długość boku każdej figury geometrycznej jest liczbą dodatnią. Program powinien sprawdzić, czy podana przez użytkownika liczba ma wartość większą od zera. Jeśli tak, to zostanie obliczone pole kwadratu, w przeciwnym wypadku program się zakończy.

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      double bok, pole;
7
8      cout << "Program obliczający pole kwadratu w oparciu o dlugosc boku." << endl;
9      cout << "Podaj dlugosc boku: ";
10
11     cin >> bok;
12
13     if (bok > 0) {
14         pole = bok * bok;
15
16         cout << "Pole kwadratu o boku dlugosci " << bok << " wynosi " << pole;
17     }
18     return 0;
19 }
20
21
```

Przykład 1c: Przykładowy output.

 Konsola debugowania programu Microsoft Visual Studio

```
Program obliczajacy pole kwadratu w oparciu o dlugosc boku.  
Podaj dlugosc boku: 10  
Pole kwadratu o boku dlugosci 10 wynosi 100
```



MS Visual Studio

Zintegrowane
środowisko
programistyczne

Wprowadzenie do MS Visual Studio

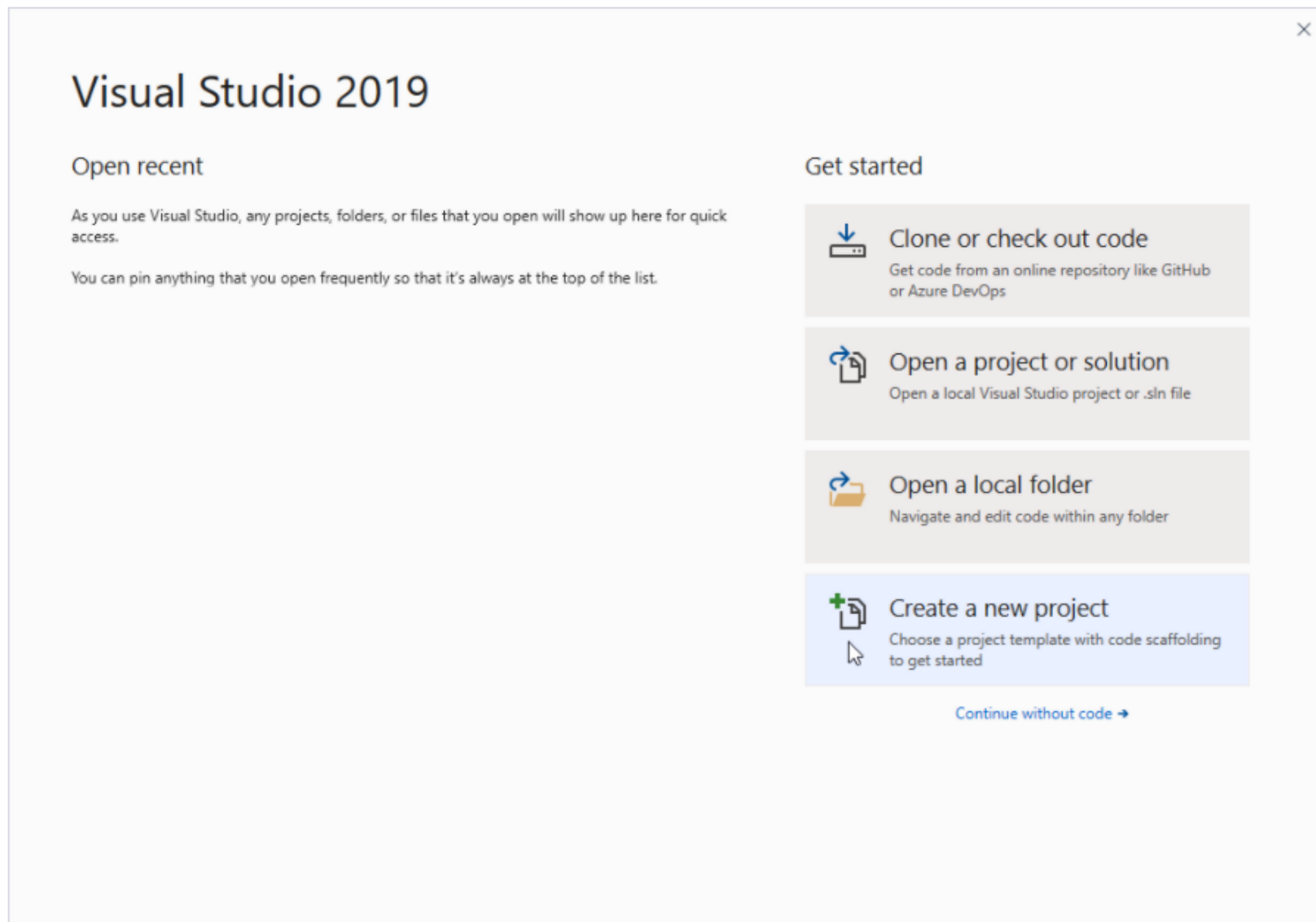
Dokumentacja:

- <https://docs.microsoft.com/pl-pl/cpp/get-started/tutorial-console-cpp?view=msvc-160>

Tworzenie projektu aplikacji

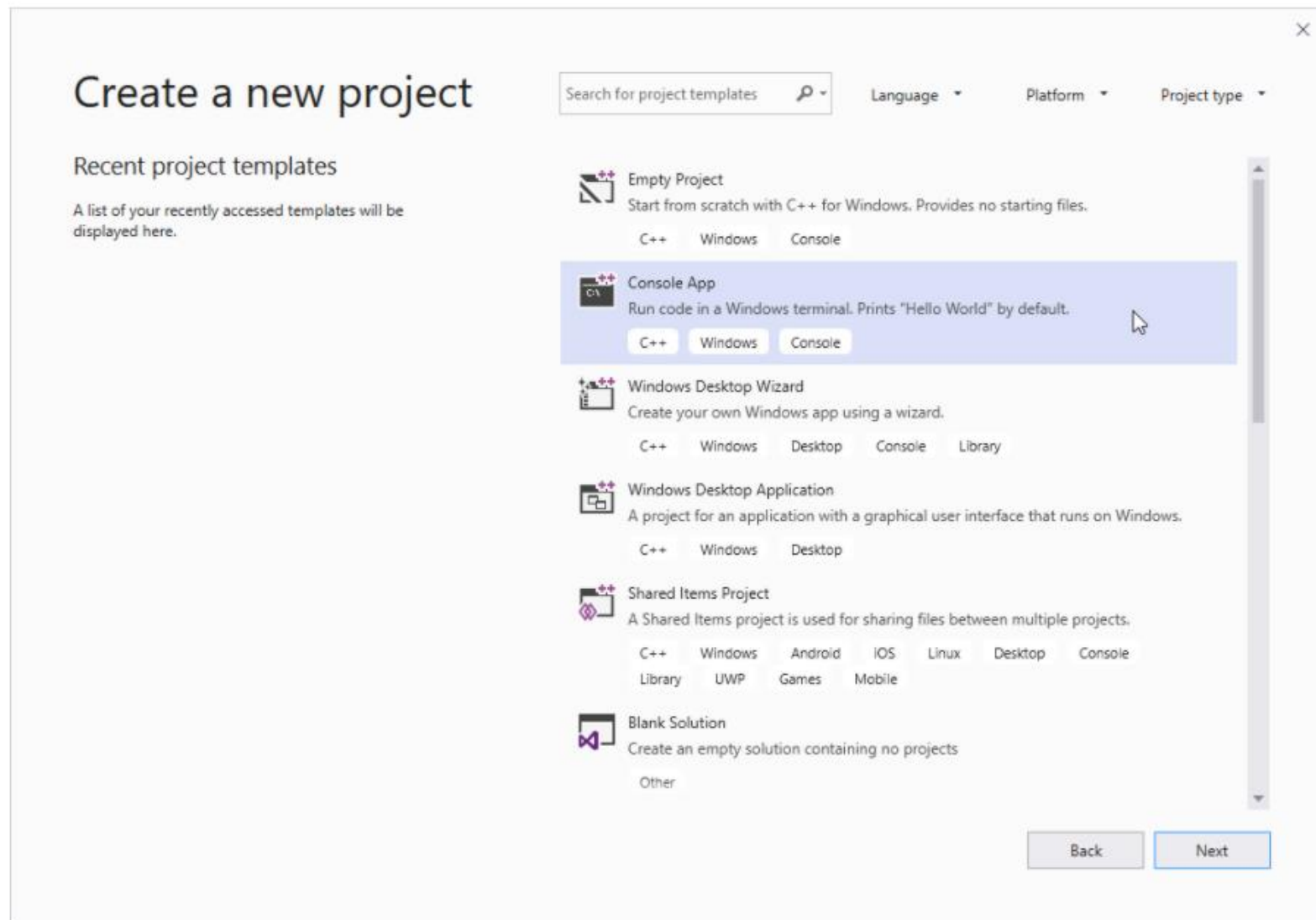
MS Visual
Studio

1. Jeśli właśnie rozpoczęto tworzenie Visual Studio, zostanie wyświetlone okno dialogowe Visual Studio 2019. Wybierz pozycję **Utwórz nowy projekt**, aby rozpocząć pracę.



W przeciwnym razie na pasku menu w Visual Studio wybierz pozycję **Nowy > > plik Project**. Zostanie otwarte okno **Tworzenie nowego projektu**.

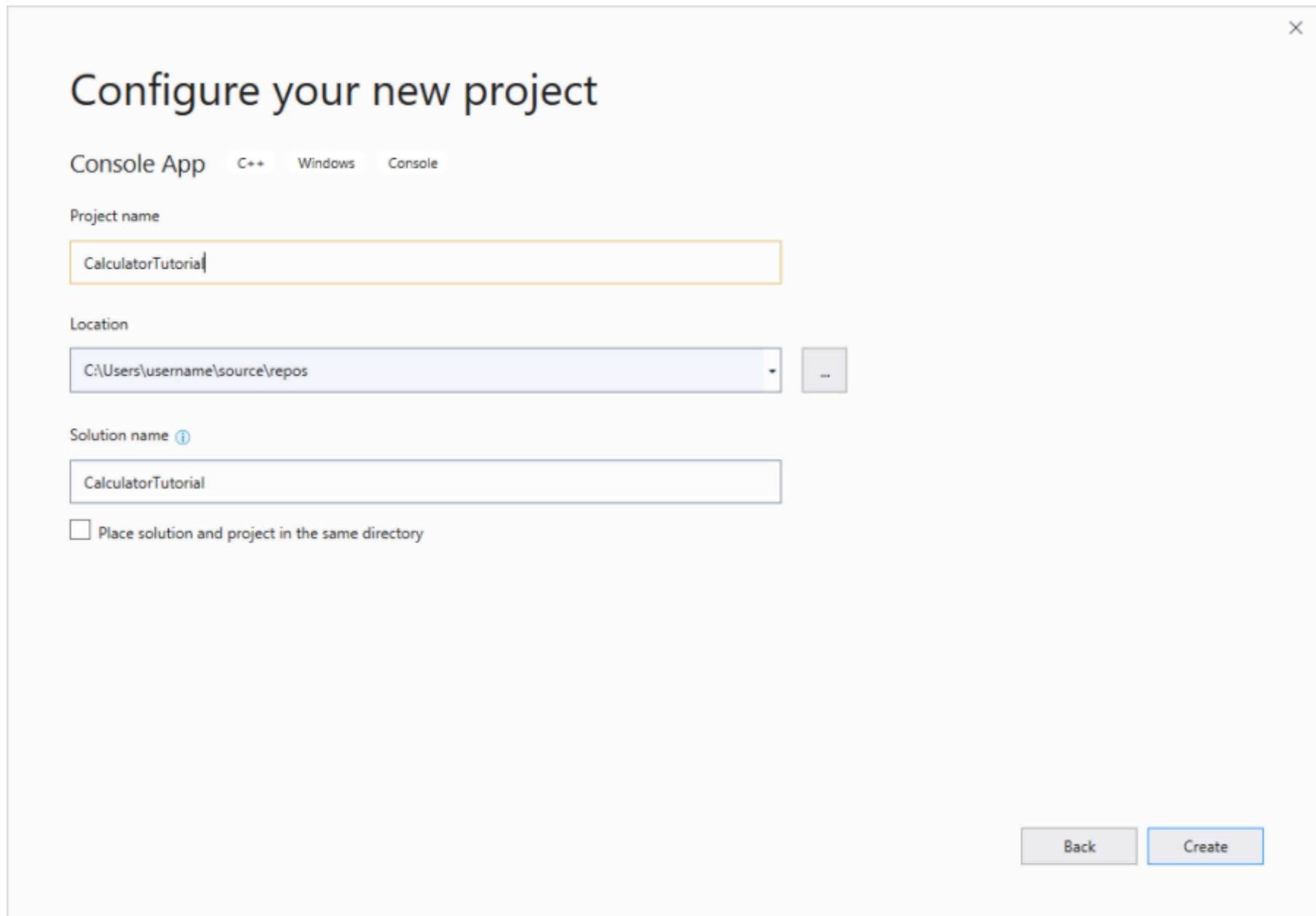
2. Na liście szablonów projektów wybierz pozycję **Aplikacja konsolowa**, a następnie wybierz pozycję **Dalej**.



Ważne

Upewnij się, że wybierasz wersję C++ szablonu **aplikacja konsolowa**. Ma tagi **C++**, **Windows** i **Console**, a w rogu ikony znajduje się symbol "++".

3. W oknie dialogowym *Configure your new project* (Konfigurowanie nowego projektu) wybierz **Project** edytuj nazwę projektu, nazwij swój nowy projekt *CalculatorTutorial*, a następnie wybierz pozycję **Create** (Utwórz).



Configure your new project

Console App C++ Windows Console

Project name

CalculatorTutorial

Location

C:\Users\username\source\repos

Solution name ⓘ

CalculatorTutorial

☐ Place solution and project in the same directory

Back Create

Tworzona jest pusta Windows konsolowa w języku C++. Aplikacje konsolowe używają okna Windows do wyświetlania danych wyjściowych i akceptowania danych wejściowych użytkownika. W Visual Studio zostanie otwarte okno edytora z wygenerowanym kodem:

C++ Kopiuuj

```
// CalculatorTutorial.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include <iostream>

int main()
{
    std::cout << "Hello World!\n";
}

// Run program: Ctrl + F5 or Debug > Start Without Debugging menu
// Debug program: F5 or Debug > Start Debugging menu

// Tips for Getting Started:
// 1. Use the Solution Explorer window to add/manage files
// 2. Use the Team Explorer window to connect to source control
// 3. Use the Output window to see build output and other messages
// 4. Use the Error List window to view errors
// 5. Go to Project > Add New Item to create new code files, or Project > Add Existing Item to add existing code
// 6. In the future, to open this project again, go to File > Open > Project and select the .sln file
```

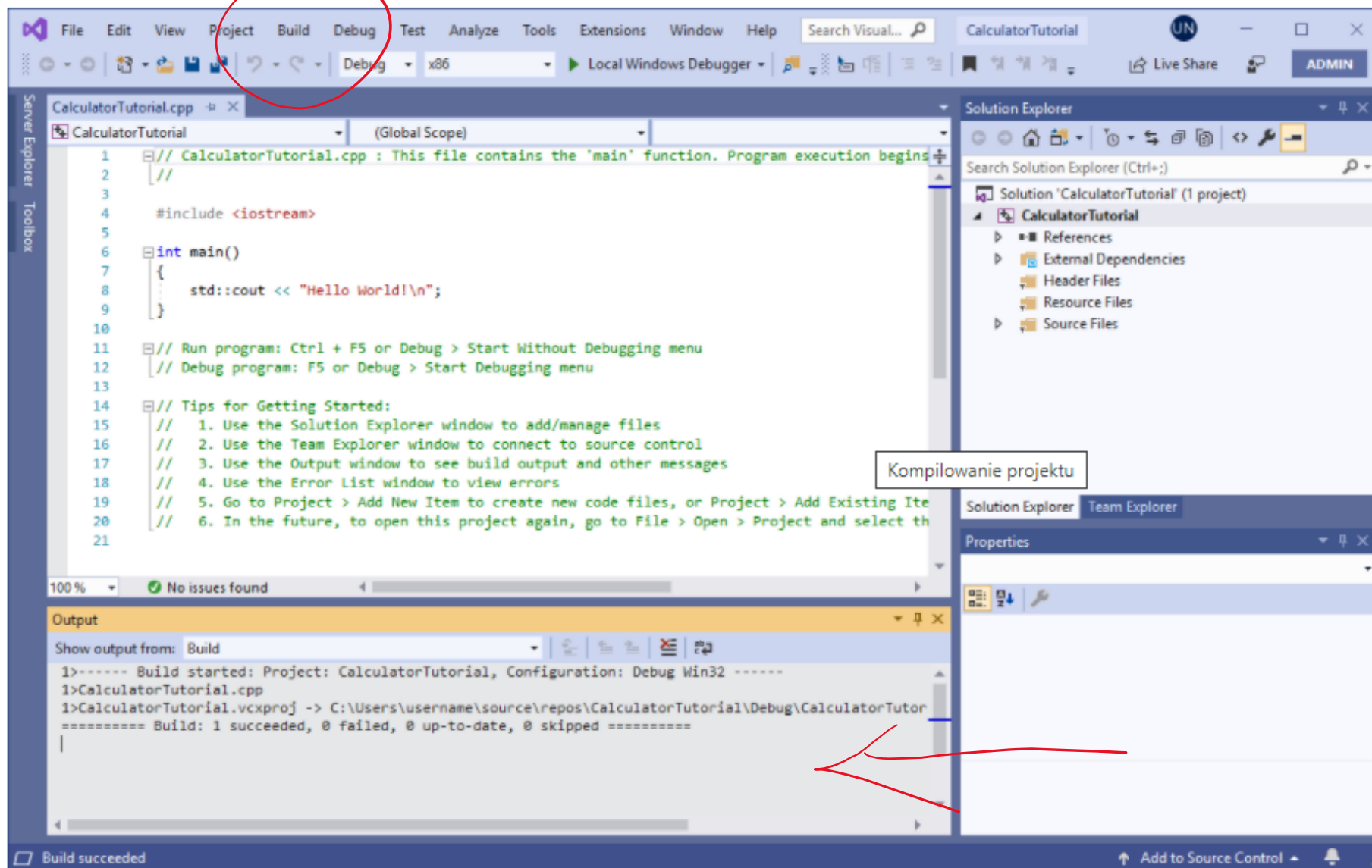
Sprawdzanie, czy nowa
aplikacja jest
kompilowana
i uruchamiana



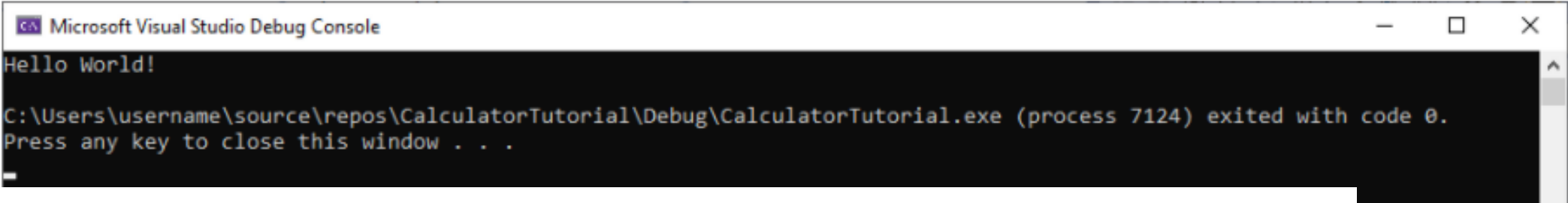
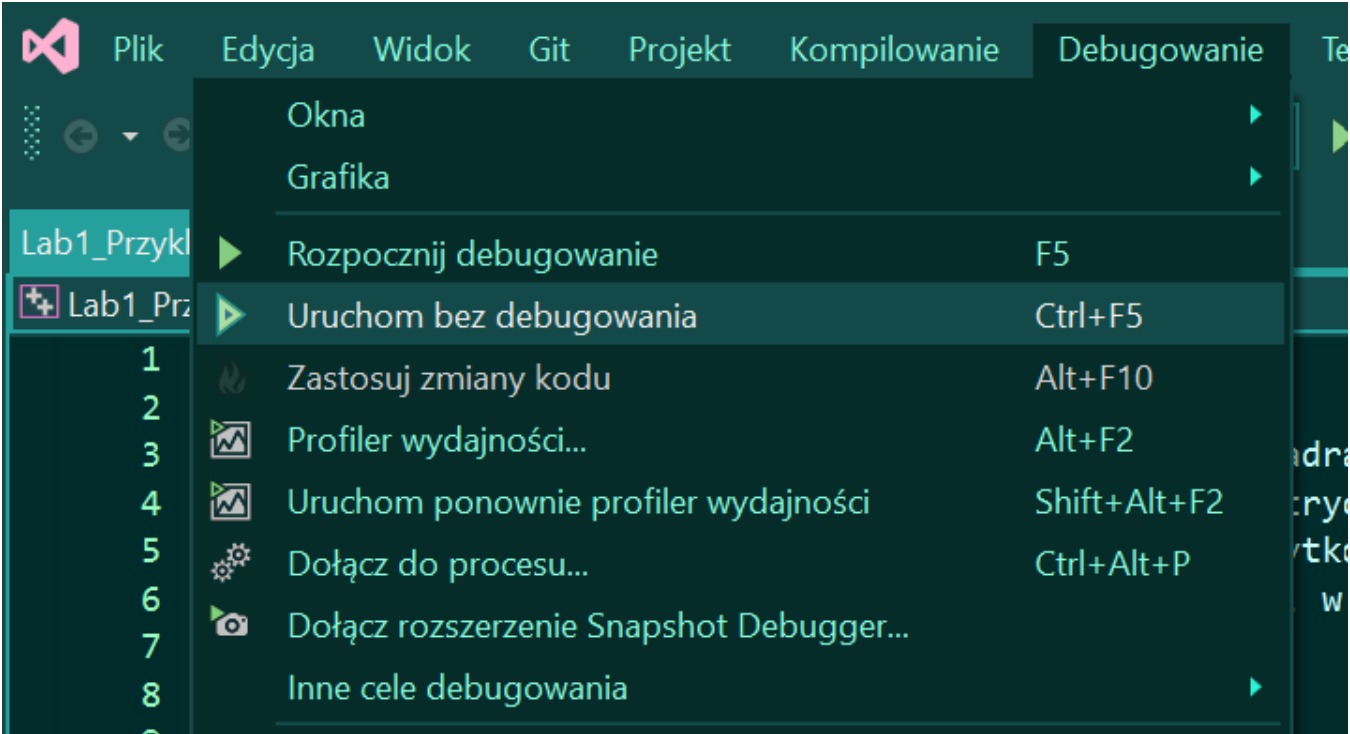
MS Visual
Studio

Szablon nowej aplikacji konsolowej Windows tworzy prostą aplikację "Hello world" języka C++. W tym momencie możesz zobaczyć, jak Visual Studio i uruchamia aplikację, które tworzysz bezpośrednio ze środowiska IDE.

1. Aby skompilować projekt, wybierz pozycję **Build Solution** (Skompilowanie rozwiązania) z menu **Build** (Kompilacja). Okno Dane wyjściowe zawiera wyniki procesu kompilacji.



2. Aby uruchomić kod, na pasku menu wybierz pozycję **Debuguj, Rozpocznij bez debugowania**.



Zostanie otwarte okno konsoli, a następnie aplikacja zostanie uruchomiona. Po uruchomieniu aplikacji konsolowej w Visual Studio uruchamia kod, a następnie drukuje komunikat "Naciśnij dowolny klawisz, aby zamknąć to okno. . .", aby wyświetlić dane wyjściowe. Gratulacje! Utworzono swój pierwszy "Hello, world!" aplikacja konsolowa w Visual Studio!

AP

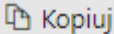
3. Naciśnij klawisz, aby odrzucić okno konsoli i wrócić do Visual Studio.

Edytowanie kodu

MS Visual
Studio

Teraz przekształćmy kod w tym szablonie w aplikację kalkulatora.

1. W pliku *CalculatorTutorial.cpp* edytuj kod, aby dopasować go do tego przykładu:

C++ 

```
// CalculatorTutorial.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include <iostream>

using namespace std;

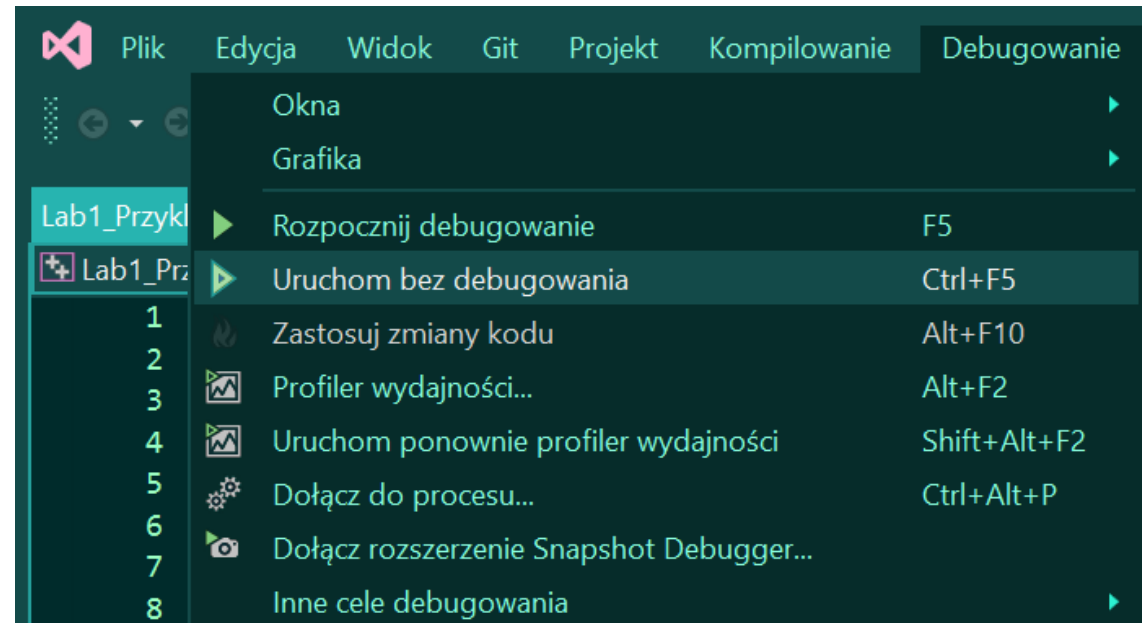
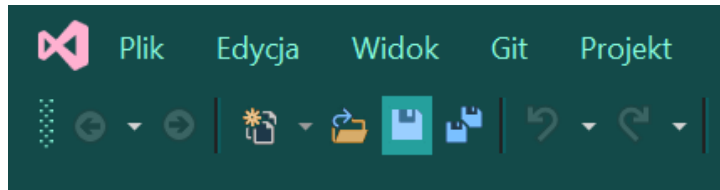
int main()
{
    cout << "Calculator Console Application" << endl << endl;
    cout << "Please enter the operation to perform. Format: a+b | a-b | a*b | a/b"
        << endl;
    return 0;
}

// Run program: Ctrl + F5 or Debug > Start Without Debugging
// Debug program: F5 or Debug > Start Debugging
// Tips for Getting Started:
// 1. Use the Solution Explorer window to add/manage files
// 2. Use the Team Explorer window to connect to source control
// 3. Use the Output window to see build output
// 4. Use the Error List window to view errors
// 5. Go to Project > Add New Item to create new class libraries, projects, or other items
// 6. In the future, to open this project again, use the Solution Explorer window
```

Opis kodu:

- Instrukcje `#include` umożliwiają odwołanie się do kodu znajdującego się w innych plikach. Czasami nazwa pliku może być otoczona nawiasami kątowymi (`<>`); czasami jest ona owijona cudzysłowami (`" "`). Ogólnie rzecz biorąc, nawiasyątowe są używane podczas odwoływania się do standardowej biblioteki C++, a cudzysłowy są używane dla innych plików.
- Wiersz nakazuje kompilatorowi, aby oczekiwał, że w tym pliku będą używane dane ze standardowej biblioteki C++ (`using namespace std;`). Bez tego wiersza każde słowo kluczowe z biblioteki musi być poprzedzone `std::`, aby określić jego zakres. Na przykład bez tego wiersza każde odwołanie do `cout` musi być zapisane jako `std::cout`. Instrukcja `using` zostanie dodana, aby kod wyglądał na bardziej czysty.
- Słowo `cout` jest używane do drukowania danych wyjściowych w języku C++. Operator nakazuje kompilatorowi wysłanie wszystkiego, co znajduje się po `<<` prawej stronie, do standardowych danych wyjściowych.
- `endl` — słowo kluczowe jest podobne do klawisza Enter; kończy wiersz i przenosi kursor do następnego wiersza. Lepszym rozwiązaniem jest włożenie wewnątrz ciągu (zawartego w cudzysłowie), aby zrobić to samo, ponieważ zawsze opróżnia bufor i może zaszkodzić wydajności programu, ale ponieważ jest to bardzo mała aplikacja, jest używana zamiast tego w celu lepszej czytelności.
- Wszystkie instrukcje języka C++ muszą kończyć się średnikami, a wszystkie aplikacje języka C++ muszą zawierać funkcję `main()`. Ta funkcja jest tym, co program uruchamia na początku. Aby można było z niego korzystać, cały kod musi być dostępny z `main()`.

2. Aby zapisać plik, naciśnij klawisze **Ctrl+S** lub wybierz ikonę Zapisz w górnej części środowiska IDE, ikonę dyskietki na pasku narzędzi na pasku menu.
3. Aby uruchomić aplikację, naciśnij klawisze **Ctrl+F5** lub przejdź do menu **Debugowanie** i wybierz polecenie **Uruchom bez debugowania**. Powinno zostać wyświetlone okno konsoli z tekstem określonym w kodzie.
4. Gdy wszystko będzie gotowe, zamknij okno konsoli.



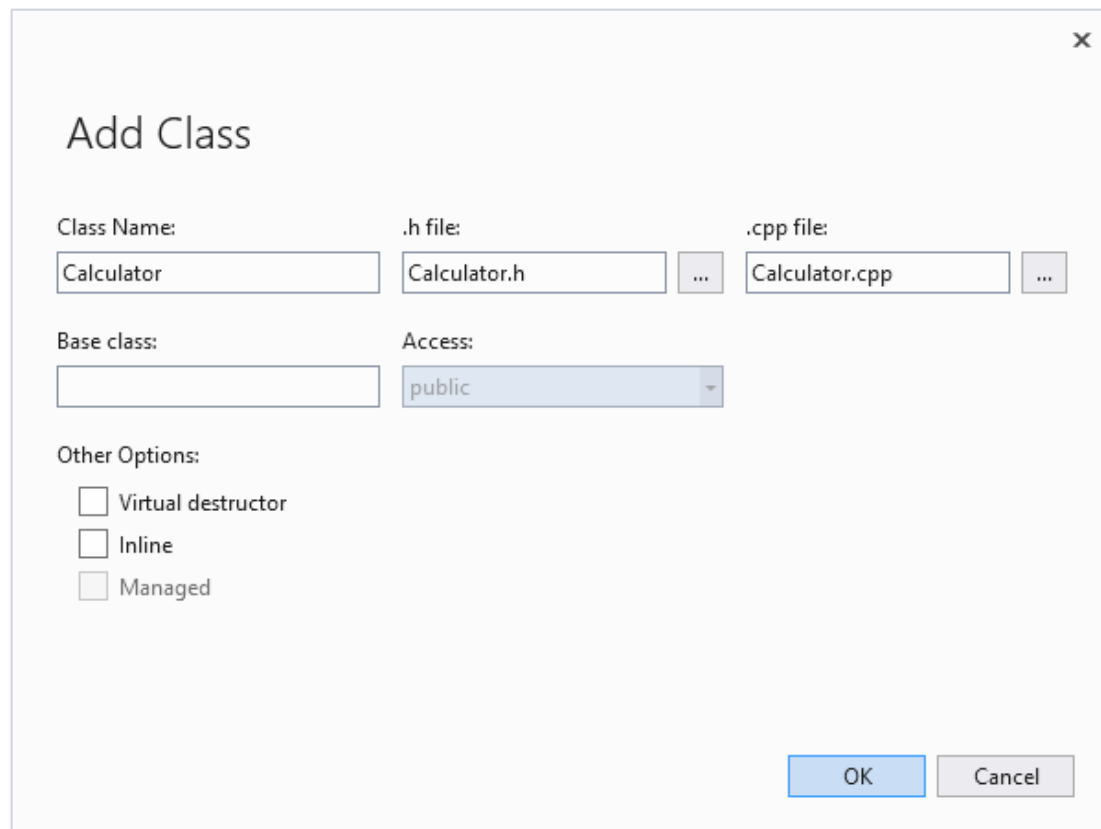
Dodawanie kodu dla wykonania obliczeń matematycznych



MS Visual
Studio

Aby dodać klasę Calculator

1. Przejdź do menu **Project** i wybierz pozycję **Dodaj klasę**. W polu edycji **Nazwa klasy** wprowadź *Calculator*. Wybierz przycisk **OK**. Do projektu zostaną dodane dwa nowe pliki. Aby zapisać wszystkie zmienione pliki jednocześnie, naciśnij **klawisze Ctrl+Shift+S**. Jest to skrót klawiaturowy dla pliku > **Zapisz wszystko**. Obok przycisku **Zapisz** znajduje się również przycisk paska narzędzi **Zapisz wszystko**, ikona dwóch dyskielek. Ogólnie rzecz biorąc, dobrą praktyką jest częste zapisywanie wszystkich, aby nie pomijać żadnych plików podczas zapisywania.



Add Class

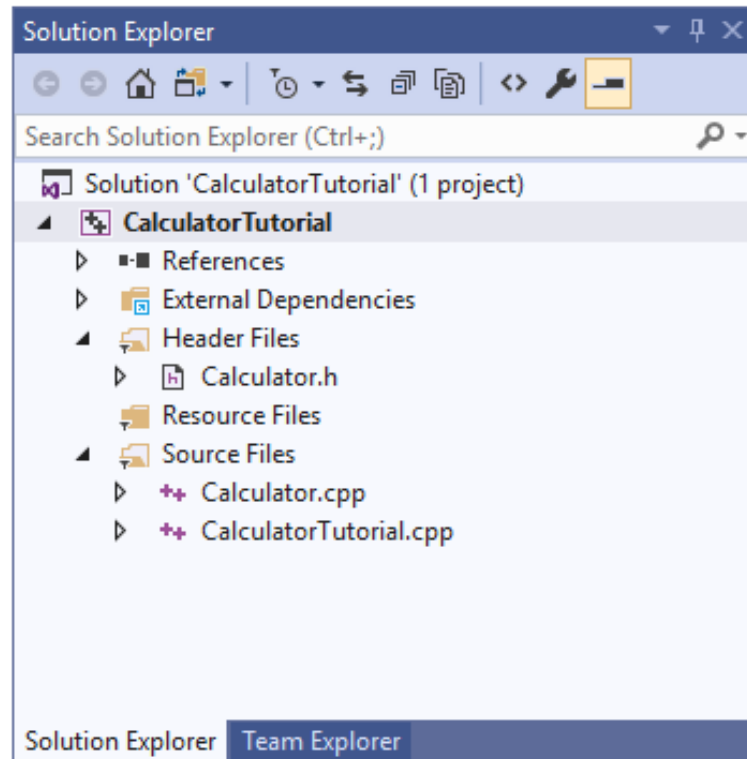
Class Name: .h file: cpp file: ...

Base class: Access: public

Other Options:

- ☐ Virtual destructor
- ☐ Inline
- ☐ Managed

Klasa jest jak strategia dla obiektu, który coś robi. W tym przypadku definiujemy kalkulator i sposób jego działania. Kreator **dodawania klasy**, który został użyty powyżej, utworzył pliki h i cpp o takiej samej nazwie jak klasa. Pełna lista plików projektu jest widoczna w oknie Eksplorator rozwiązań **widocznym** po stronie środowiska IDE. Jeśli okno nie jest widoczne, możesz je otworzyć na pasku menu: wybierz pozycję > **Wyświetl Eksplorator rozwiązań**.



Eksplorator rozwiązań

W edytorze powinny być teraz otwarte trzy karty: *CalculatorTutorial.cpp*, *Calculator.h* i *Calculator.cpp*. Jeśli przypadkowo zamkniesz jedną z nich, możesz otworzyć ją ponownie, klikając ją dwukrotnie w **Eksplorator rozwiązań** okna.

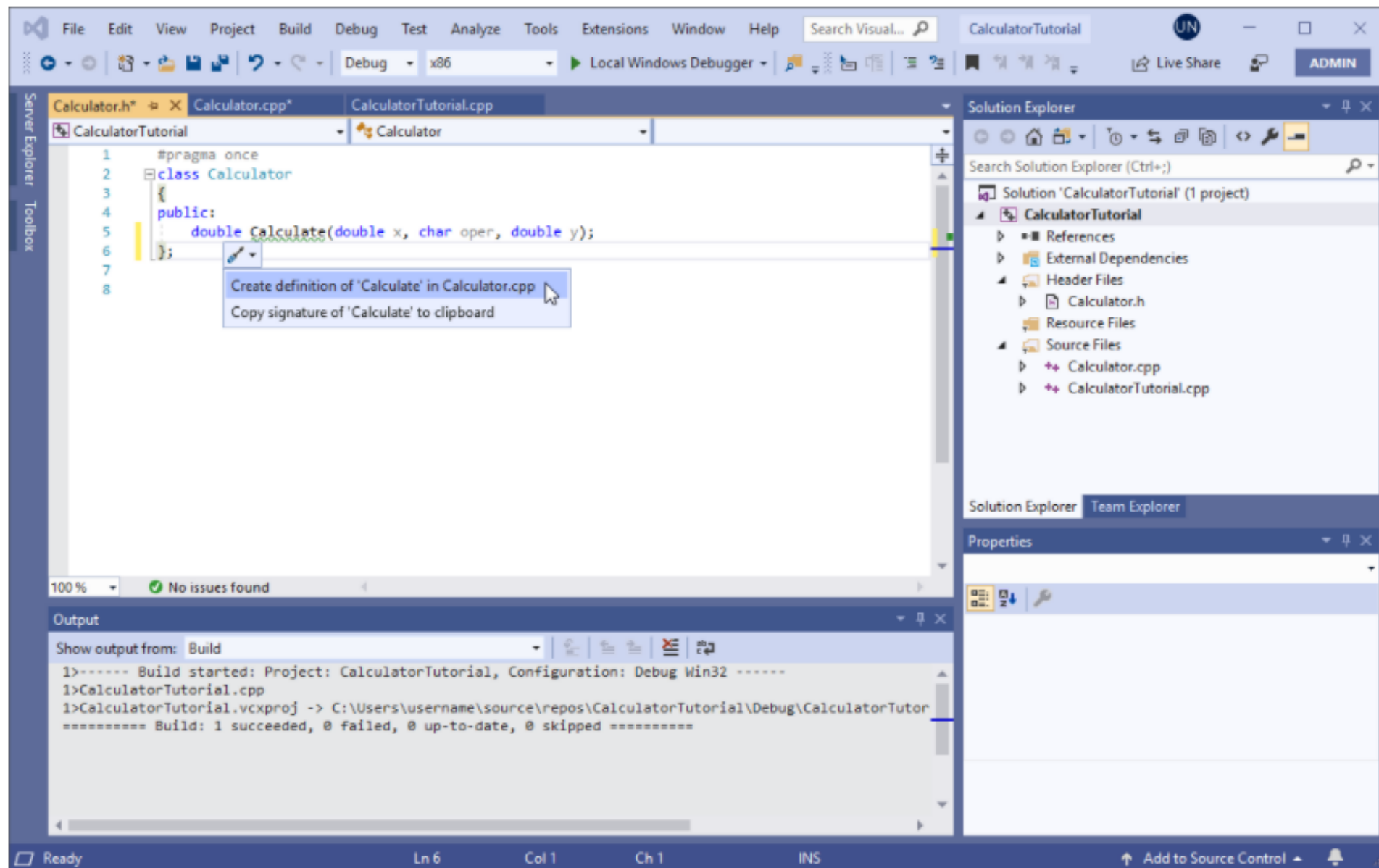
2. W aplikacji `Calculator.h` usuń wygenerowane wiersze i , ponieważ `Calculator();` nie będą one potrzebne w tym `~Calculator();` miejscu. Następnie dodaj następujący wiersz kodu, aby plik wyglądał następująco:

```
C++ Kopiuuj  
  
#pragma once  
class Calculator  
{  
public:  
    double Calculate(double x, char oper, double y);  
};
```

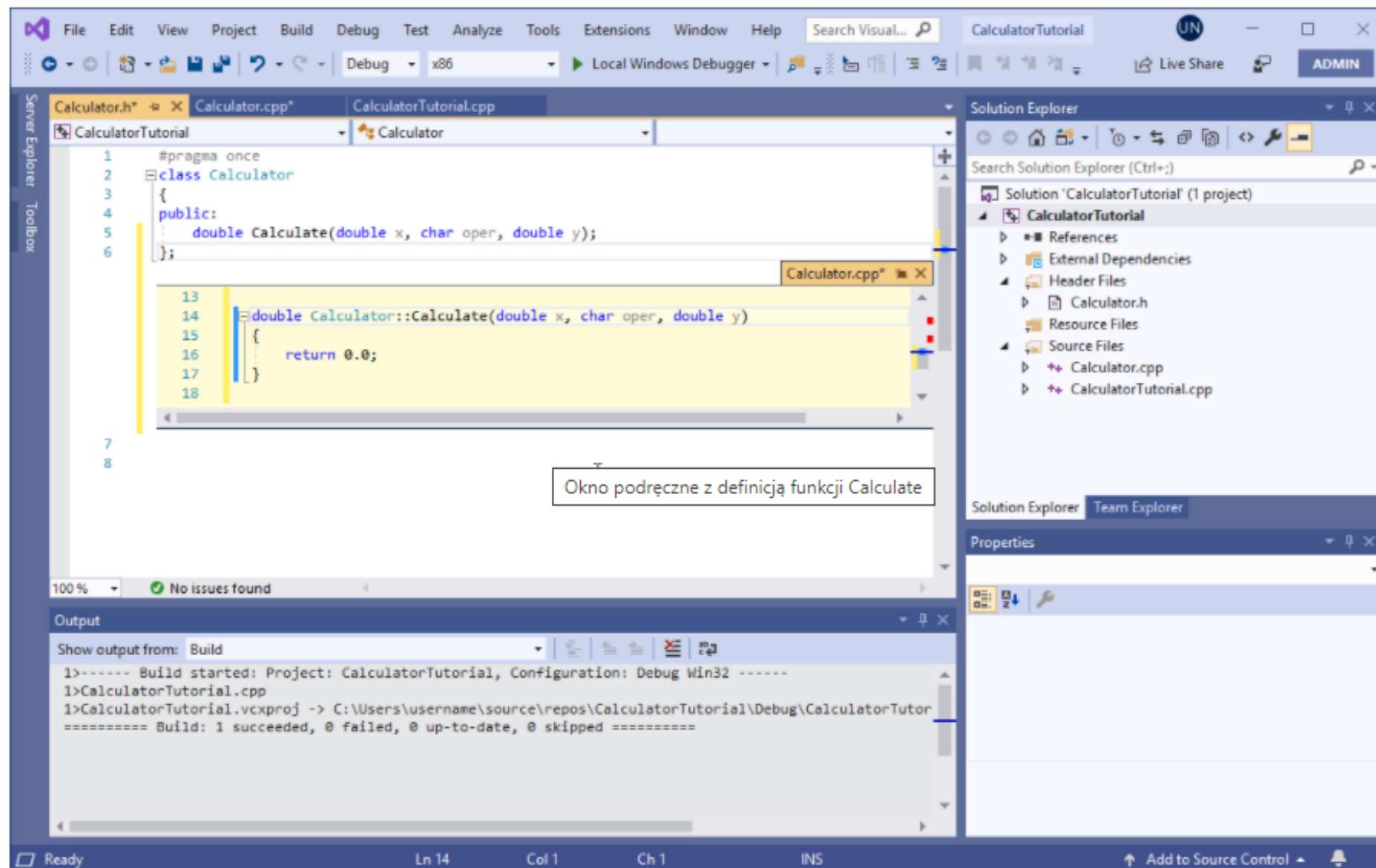
Omówienie kodu

- Dodany wiersz deklaruje nową funkcję o nazwie , której użyjemy do uruchamiania operacji matematycznych do dodawania, odejmowania, `Calculate` mnożenia i dzielenia.
- Kod C++ jest zorganizowany w pliki nagłówkowe (h) i pliki źródłowe (cpp). Kilka innych rozszerzeń plików jest obsługiwanych przez różne kompilatory, ale są to najważniejsze rozszerzenia, o których należy wiedzieć. Funkcje i zmienne są zwykle deklarowane , czyli nadawać nazwę i typ w plikach nagłówkowych i implementowane lub nadawać definicję w plikach źródłowych. Aby uzyskać dostęp do kodu zdefiniowanego w innym pliku, można użyć , gdzie "filename.h" to nazwa pliku, który deklaruje zmienne lub funkcje, których `#include "filename.h"` chcesz użyć.
- Dwa usunięte wiersze zadeklarowały konstruktor i destruktor dla klasy . W przypadku prostej klasy, takiej jak ta, kompilator tworzy je dla Ciebie, a ich zastosowania wykraczają poza zakres tego samouczka.
- Dobrym rozwiązaniem jest zorganizowanie kodu w różne pliki w zależności od jego działania, dzięki czemu będzie można łatwo znaleźć potrzebny kod później. W naszym przypadku definiujemy klasę oddzielnie od pliku zawierającego funkcję, ale planujemy odwołanie `Calculator` do klasy w klasie `main()` `Calculator` `main()` .

3. W obszarze `. Calculate` Jest to spowodowane tym, że nie zdefiniowaliśmy funkcji `Calculate` w pliku `cpp`. Zatrzymaj wskaźnik myszy na wyrazie, kliknij ikonę żarówki (w tym przypadku śrubokręt), która się pojawi, a następnie wybierz pozycję Utwórz definicję funkcji "Calculate" w oknie `Calculator.cpp`.



Zostanie wyświetlone okno podręczne, które umożliwia podgląd zmiany kodu wprowadzonej w innym pliku. Kod został dodany do *calculator.cpp*.



4. Przejdź do pliku *Calculator.cpp* w oknie edytora. Usuń `Calculator()` sekcje i `~Calculator()` (tak jak w pliku h) i dodaj następujący kod do pliku `Calculate()` :

```
C++Kopiuuj  
  
#include "Calculator.h"  
  
double Calculator::Calculate(double x, char oper, double y)  
{  
    switch(oper)  
    {  
        case '+':  
            return x + y;  
        case '-':  
            return x - y;  
        case '*':  
            return x * y;  
        case '/':  
            return x / y;  
        default:  
            return 0.0;  
    }  
}
```

Omówienie kodu

- Funkcja zużywa liczbę, operator i drugą liczbę, a następnie wykonuje `Calculate` żadaną operację na liczbach.
- Instrukcja switch sprawdza, który operator został dostarczony, i wykonuje tylko przypadek odpowiadający tej operacji. Wartość domyślna: przypadek jest rezerwowym w przypadku, gdy użytkownik typi operator, który nie jest akceptowany, więc program nie przerwie. Ogólnie rzecz biorąc, najlepszym rozwiązaniem jest obsługa nieprawidłowych danych wejściowych użytkownika w bardziej elegancki sposób, ale wykracza to poza zakres tego samouczka.
- Słowo `double` kluczowe oznacza typ liczby, który obsługuje liczby dziesiętne. W ten sposób kalkulator może obsługiwać zarówno obliczenia matematyczne na liczbach dziesiętnych, jak i na liczbach całkowitych. Funkcja musi zawsze zwracać taką liczbę ze względu na wartość na samym początku kodu (oznacza to typ zwracany funkcji), dlatego zwracamy wartość `Calculate` 0.0 nawet w przypadku `double` domyślnym.
- Plik .h deklaruje prototyp funkcji , który z góry informuje kompilator o wymaganych parametrach i typach zwracanych przez ten plik. Plik cpp zawiera wszystkie szczegóły implementacji funkcji.

W przypadku skompilowania i uruchomienia kodu ponownie w tym momencie kod nadal będzie kończyć działanie po pytaniu, którą operację wykonać. Następnie zmodyfikujesz `main` funkcję , aby wykonać kilka obliczeń.

Wywoływanie funkcji

MS Visual
Studio

```

C++ Kopiuuj

// CalculatorTutorial.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include <iostream>
#include "Calculator.h"

using namespace std;

int main()
{
    double x = 0.0;
    double y = 0.0;
    double result = 0.0;
    char oper = '+';

    cout << "Calculator Console Application" << endl << endl;
    cout << "Please enter the operation to perform. Format: a+b | a-b | a*b | a/b"
         << endl;

    Calculator c;
    while (true)
    {
        cin >> x >> oper >> y;
        result = c.Calculate(x, oper, y);
        cout << "Result is: " << result << endl;
    }

    return 0;
}

```

Omówienie kodu

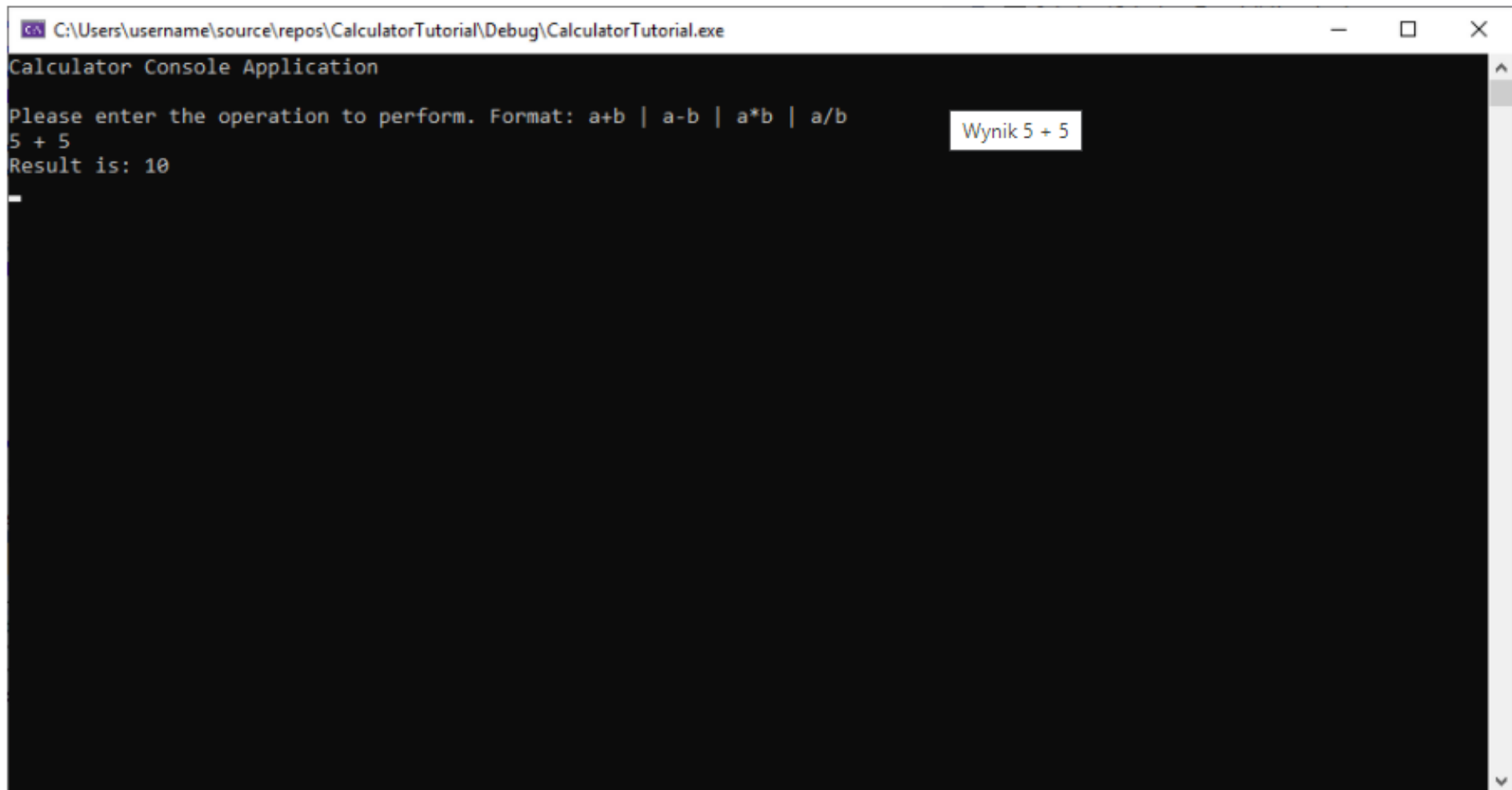
- Ponieważ programy w języku C++ zawsze zaczynają się od funkcji, musimy wywołać inny kod z tego miejsca, więc potrzebna `main()` `#include` jest instrukcja .
- Niektóre zmienne początkowe , , i są deklarowane do przechowywania odpowiednio pierwszej liczby, drugiej `x` `y` `oper` `result` liczby, operatora i wyniku końcowego. Zawsze dobrym rozwiązaniem jest nadaj im pewne wartości początkowe, aby uniknąć niezdefiniowanych zachowań, co jest tutaj wykonywane.
- Wiersz `Calculator c;` deklaruje obiekt o nazwie "c" jako wystąpienie `Calculator` klasy . Sama klasa jest tylko planem działania kalkulatorów. obiekt jest określonym kalkulatorem, który oblicza obliczenia matematyczne.
- Instrukcja `while (true)` jest pętlą. Kod wewnątrz pętli jest nadal wykonywany 2018 000 000 000 000, o ile warunek wewnątrz pętli `()` ma wartość `true`. Ponieważ warunek jest po prostu wymieniony jako `true` , zawsze jest to prawda, więc pętla jest uruchamiana w nieskończoność. Aby zamknąć program, użytkownik musi ręcznie zamknąć okno konsoli. W przeciwnym razie program zawsze czeka na nowe dane wejściowe.
- Słowo `cin` kluczowe jest używane do akceptowania danych wejściowych od użytkownika. Ten strumień wejściowy jest wystarczająco inteligentny, aby przetworzyć wiersz tekstu wprowadzony w oknie konsoli i umieścić go wewnątrz każdej z wymienionych zmiennych w podanej kolejności, przy założeniu, że dane wejściowe użytkownika są zgodne z wymaganą specyfikacją. Możesz zmodyfikować ten wiersz tak, aby akceptował różne typy danych wejściowych, na przykład więcej niż dwie liczby, chociaż funkcja również musi zostać zaktualizowana, aby to `Calculate()` obsłużyć.
- Wyrażenie `c.Calculate(x, oper, y);` wywołuje `Calculate` zdefiniowaną wcześniej funkcję i dostarcza wprowadzone wartości wejściowe. Następnie funkcja zwraca liczbę, która jest przechowywana w `result` .
- Na koniec zostanie wyświetlony w konsoli, aby użytkownik widział `result` wynik obliczenia.

Kompilowanie i testowanie kodu

MS Visual
Studio

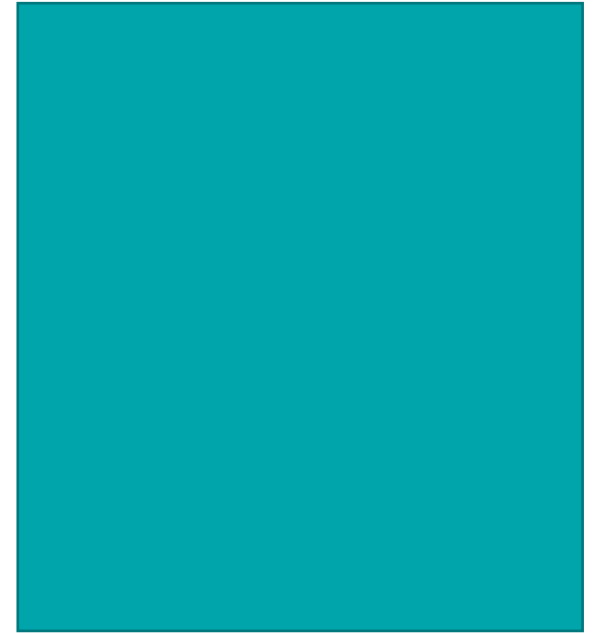
Teraz na czas ponownie przetestować program, aby upewnić się, że wszystko działa prawidłowo.

1. Naciśnij klawisze **Ctrl+F5**, aby ponownie skompilować i uruchomić aplikację.
2. Wprowadź `5 + 5` , a następnie naciśnij klawisz **Enter**. Sprawdź, czy wynik wynosi 10.



```
C:\Users\username\source\repos\CalculatorTutorial\Debug\CalculatorTutorial.exe
Calculator Console Application
Please enter the operation to perform. Format: a+b | a-b | a*b | a/b
5 + 5
Result is: 10
```

Zadania do wykonania



Termin nadsyłania rozwiązań

Piątek przed kolejnym zjazdem.

Zadanie 1.1

Napisz program w C++, który wypisze w osobnych liniach Twoje imię, nr studenta i nazwę uczelni.

(1 pkt)

Zadanie 1.2

Napisz program, który sprawdzi, czy liczba całkowita jest dodatnia, ujemna czy może jest 0.
(1 pkt)

Zadanie 1.3

Napisz program, który sprawdzi, czy podana liczba całkowita jest parzysta, nieparzysta, czy żadna (0).
(1 pkt)

Zadanie 1.4*

Do
samodzielnego
wykonania na
zajęciach (max.
3 punkty) lub
jako zadanie
domowe (1
punkt)

Zadanie 1.4

Napisz program do znajdowania największej liczby wśród trzech liczb, podanych przez użytkownika. (3 rozwiązania)
(1 pkt + 2 za wszystkie 3 rozwiązania)

** wskazówka: && to operator logiczny AND*