

Podstawowe własności języków obiektowych



Programowanie obiektowe w C++



AGENDA

- Programowanie proceduralne a programowanie obiektowe
- Język C++
- Praktyczny przykład 1
- Różnica między C a C++
- Ewolucja języka C++
- -Kluczowe koncepcje programowania obiektowego
- Zalety i wady programowania obiektowego
- Praktyczny przykład 2



PROGRAMOWANIE PROCEDURALNE

- Oznacza „*zbiór procedur*”, który jest „*zbiorem podprogramów*” lub „*zbiorem funkcji*”.
- Funkcje są wywoływane wielokrotnie w programie w celu wykonania zadań przez nie wykonywanych
- **Przykład:** Program może obejmować zbieranie danych od użytkownika (czytanie), wykonywanie pewnego rodzaju obliczeń na zebranych danych (obliczanie), a na koniec wyświetlanie wyniku użytkownikowi na żądanie (drukowanie). Wszystkie 3 zadania odczytu, obliczania i drukowania można zapisać w programie za pomocą 3 różnych funkcji, które wykonują te 3 różne zadania.



PRZYKŁAD ZE ŚWIATA RZECZYWISTEGO

- Pracujemy dla producenta części samochodowych, który musi zaktualizować swój internetowy system inwentaryzacji. Szef każe nam zaprogramować dwa podobne, ale oddzielne formularze dla strony internetowej, jeden formularz, który przetwarza informacje o samochodach i jeden, który robi to samo dla ciężarówek
- W przypadku samochodów będziemy musieli zarejestrować następujące informacje:
- Kolor, rozmiar silnika, typ skrzyni biegów, liczba drzwi.
- W przypadku ciężarówek informacje będą podobne, ale nieco inne. Potrzebujemy:
- Kolor, Rozmiar silnika, Typ skrzyni biegów, Rozmiar kabiny, Zdolność holownicza



SCENARIUSZ - 1

- Założmy, że nagle musimy dodać formularz autobusu, który rejestruje następujące informacje:
- Kolor, Rozmiar silnika, Typ skrzyni biegów, Liczba pasażerów
- *Proceduralne:*
- Musimy odtworzyć cały formularz, powtarzając kod dla koloru, rozmiaru silnika i typu skrzyni biegów.

Typ skrzyni biegów.

- *PO:*

- Po prostu rozszerzamy klasę pojazdu o klasę autobusu i dodajemy metodę

`numberOfPassengers`



SCENARIUSZ - 2

Zamiast przechowywać kolory w bazie danych, jak to robiliśmy wcześniej, z jakiegoś dziwnego powodu nasz klient chce, aby kolor został mu przesłany e-mailem.

- Proceduralne:

Zmieniamy trzy różne formularze: samochody, ciężarówki i autobusy, aby wysłać kolor do klienta e-mailem

zamiast przechowywać go w bazie danych.

-PO:

-Zmieniamy metodę koloru w klasie pojazdu, a ponieważ klasy samochodu, ciężarówki i autobusu rozszerzają (lub dziedziczą, mówiąc inaczej) klasę pojazdu, są one automatycznie aktualizowane.



SCENARIUSZ - 3

- Chcemy odejść od samochodów generycznych na rzecz konkretnych marek, np: Nissan i Creta

- ***Proceduralne:***

- Tworzymy nowy formularz dla każdej marki, powtarzając cały kod dla ogólnych informacji o samochodzie i dodając kod specyficzny dla każdej marki.

i dodając kod specyficzny dla każdej marki

- ***PO:***

- Rozszerzamy klasę samochodu o klasę Nissan i klasę Creta i dodajemy metody dla każdego zestawu unikalnych informacji dla danej marki.

zestaw unikalnych informacji dla danej marki samochodu



SCENARIUSZ - 4

- Znaleźliśmy błąd w obszarze typu transmisji naszego formularza i musimy go naprawić.
- ***Proceduralne:***
 - Otwieramy i aktualizujemy każdy formularz
- ***PO:***
 - Naprawiamy metodę transmission type w klasie pojazdu, a zmiana utrwała się w każdej klasie, która po niej dziedziczy.



Programowanie proceduralne | programowanie obiektowe

Programowanie proceduralne	Programowanie obiektowe
W programowaniu proceduralnym program jest podzielony na małe części zwane funkcjami .	W programowaniu obiektowym program jest podzielony na małe części zwane obiektami .
Programowanie proceduralne opiera się na podejściu odgórnym .	Programowanie obiektowe opiera się na podejściu oddolnym .
W programowaniu proceduralnym nie ma specyfikatora dostępu.	Programowanie obiektowe ma specyfikatory dostępu, takie jak prywatny, publiczny, chroniony itp.
Dodawanie nowych danych i funkcji nie jest łatwe.	Dodawanie nowych danych i funkcji jest łatwe.
Programowanie proceduralne nie ma odpowiedniego sposobu ukrywania danych, więc jest mniej bezpieczne .	Programowanie obiektowe zapewnia ukrywanie danych, dzięki czemu jest bezpieczniejsze .
W programowaniu proceduralnym przeciążanie nie jest możliwe.	Przeciążanie jest możliwe w programowaniu obiektowym.
W programowaniu proceduralnym funkcja jest ważniejsza niż dane.	W programowaniu obiektowym dane są ważniejsze niż funkcja.
Programowanie proceduralne opiera się na nierzeczywistym świecie .	Programowanie obiektowe opiera się na świecie rzeczywistym .
Przykłady: C, FORTRAN, Pascal, Basic etc.	Przykłady: C++, Java, Python, C# etc.



Co to jest C++?



- ▶ C++ to wieloplatformowy język, który może być wykorzystywany do tworzenia wysokowydajnych aplikacji.
- ▶ C++ została opracowana przez Bjarne Stroustrupa jako rozszerzenie języka C.
- ▶ C++ daje programistom wysoki poziom kontroli nad zasobami systemowymi i pamięcią.



Dlaczego używać C++?

- ▶ C++ to jeden z najpopularniejszych języków programowania na świecie.
- ▶ Język C++ można znaleźć w dzisiejszych systemach operacyjnych, graficznych interfejsach użytkownika i systemach wbudowanych.
- ▶ C++ jest obiektowym językiem programowania, który nadaje programom przejrzystą strukturę i pozwala na ponowne wykorzystanie kodu, obniżając koszty rozwoju.
- ▶ Język C++ jest przenośny i może być używany do tworzenia aplikacji, które można dostosować do wielu platform.
- ▶ Język C++ jest przyjemny i łatwy do nauczenia!



JAK PRZEJŚĆ NA C++

1. Poznanie różnica.
 - a. Nowe narzędzia (kompilatory, debuggery itp.)
 - b. Nowe biblioteki
 - c. Nowe konwencje nazewnictwa plikówd.
 - d. Nowa składniae.
 - e. Dostępne standardy
1. Przemyślenie podejścia do programowania



1A. NOWE NARZĘDZIAC++

- Kompilatory obsługujące C++:

- **GCC (GNU Compiler Collection):** Popularny i darmowy kompilator, który wspiera zarówno C, jak i C++.

- Kompilacja programów C++:

```
bash
```

[Copy code](#)

```
g++ program.cpp -o program
```

- **Clang:** Nowoczesny kompilator, który również wspiera oba języki, ale z lepszą optymalizacją i szybkością kompilacji.
- **MSVC (Microsoft Visual C++):** Kompilator dostarczany z Visual Studio, często używany na systemach Windows.
- **Różnice w kompilacji:**
 - W C++ rozszerzono możliwości kompilatora, m.in. o obsługę przestrzeni nazw, wyjątków, szablonów (templates), przeciążania funkcji itp.
 - W C++, w przeciwieństwie do C, należy kompilować pliki z rozszerzeniem `.cpp`.



1A. NOWE NARZĘDZIA C++

2. Debuggery dla C++

C++ oferuje szerokie wsparcie dla zaawansowanego debugowania, co ułatwia pracę nad bardziej skomplikowanymi projektami.

- **GDB (GNU Debugger):** Najczęściej używany debugger dla C i C++, pozwala na krokowe wykonywanie kodu, ustawianie punktów przerwania i inspekcję wartości zmiennych.
- Debugowanie programów C++:

```
bash
```

[Copy code](#)

```
gdb ./program
```

- **LLDB:** Debugger rozwijany przez społeczność LLVM, zapewnia nowoczesne wsparcie dla programów napisanych w C++.
- **Microsoft Visual Studio Debugger:** Zintegrowany z Visual Studio, oferuje wygodny i graficzny interfejs do debugowania kodu w C++.

Nowości w debugowaniu C++ w porównaniu do C:

- **Obsługa klas i obiektów:** Możliwość inspekcji obiektów, ich metod i atrybutów.
- **Śledzenie wyjątków:** Możliwość śledzenia wyjątków (`try` , `catch`) w czasie rzeczywistym.



1A. NOWE NARZĘDZIA C++

3. IDE (Zintegrowane Środowisko Programistyczne)

Przy przejściu na C++ warto skorzystać z bardziej rozbudowanych narzędzi, takich jak **IDE**, które wspierają zaawansowane funkcje C++ i oferują wygodę pracy w nowoczesnych projektach.

- **Visual Studio Code (VS Code):** Popularne, lekkie IDE, które wspiera rozszerzenia dla C++ (np. C/C++ Extension Pack). Oferuje:
 - **IntelliSense:** Automatyczne uzupełnianie kodu i odpowiedzi dla C++.
 - **Zintegrowany debugger:** Obsługuje debugowanie z GDB i LLDB.
- **CLion:** Komercyjne IDE od JetBrains, znane z zaawansowanych funkcji C++:
 - Automatyczne refaktoryzacje.
 - Ścisła integracja z CMake.
 - Zaawansowane narzędzia do debugowania.
- **Microsoft Visual Studio:** Bogate środowisko do pracy w C++, szczególnie na systemie Windows:
 - Graficzny debugger.
 - Wsparcie dla projektów z wykorzystaniem szablonów C++ i klas.
 - Wbudowane narzędzia do pracy z CMake i zarządzania zależnościami.



1B. Nowe biblioteki C++


1. Standard Template Library (STL)

Jedną z kluczowych różnic między C a C++ jest **Standardowa Biblioteka Szablonów (STL)**, która zapewnia gotowe, generyczne komponenty do obsługi struktur danych i algorytmów.

- **Kontenery:** STL oferuje zróżnicowane kontenery, takie jak `std::vector`, `std::list`, `std::map`, czy `std::set`, które eliminują potrzebę ręcznego zarządzania pamięcią i pozwalają na dynamiczne operacje na danych.

- **Przykład w C:**

c

 Copy code

```
int arr[10]; // Statyczna tablica o ustalonym rozmiarze
```

- **Przykład w C++:**

cpp

 Copy code

```
std::vector<int> arr; // Dynamiczny wektor, który automatycznie zarządza rozmiarem
```



1B. Nowe biblioteki C++


1. Standard Template Library (STL)

Jedną z kluczowych różnic między C a C++ jest **Standardowa Biblioteka Szablonów (STL)**, która zapewnia gotowe, generyczne komponenty do obsługi struktur danych i algorytmów.

- **Kontenery:** STL oferuje zróżnicowane kontenery, takie jak `std::vector`, `std::list`, `std::map`, czy `std::set`, które eliminują potrzebę ręcznego zarządzania pamięcią i pozwalają na dynamiczne operacje na danych.

- **Przykład w C:**


c

 Copy code

```
int arr[10]; // Statyczna tablica o ustalonym rozmiarze
```

- **Przykład w C++:**

cpp

 Copy code

```
std::vector<int> arr; // Dynamiczny wektor, który automatycznie zarządza rozmiarem
```

- **Algorytmy:** STL zawiera zestaw gotowych algorytmów (np. sortowanie, wyszukiwanie, kopiowanie), które współpracują z kontenerami.



1B. Nowe biblioteki C++

2. Stringi w C++ (`std::string`)

W C praca z łańcuchami znaków (`char[]`) jest trudna, wymaga ręcznego zarządzania pamięcią i korzystania z funkcji, takich jak `strcpy` czy `strcat`. C++ upraszcza to, wprowadzając klasę `std::string`.

- Przykład w C:

c

Copy code

```
char name[50];  
strcpy(name, "Jan Kowalski");
```

- Przykład w C++:

cpp

Copy code

```
std::string name = "Jan Kowalski";
```

Klasa `std::string` automatycznie zarządza pamięcią, oferuje intuicyjne metody manipulacji tekstem (np. `substr()`, `find()`, `append()`), a także integruje się z innymi elementami STL.




1B. Nowe biblioteki C++

3. `std::unique_ptr` i zarządzanie pamięcią

C++ wprowadza nowoczesne narzędzia do zarządzania dynamiczną pamięcią, takie jak **inteligentne wskaźniki** (`smart pointers`), co eliminuje konieczność ręcznego używania `malloc` i `free` , jak w C.

- `std::unique_ptr` : Wskaźnik zarządzany, który automatycznie zwalnia pamięć, gdy przestaje być używany.
 - Przykład w C (ręczne zarządzanie pamięcią):


c

 Copy code

```
int* ptr = (int*)malloc(sizeof(int));  
free(ptr);
```

- Przykład w C++ (z `std::unique_ptr`):

cpp

 Copy code

```
std::unique_ptr<int> ptr = std::make_unique<int>(42);
```

Inteligentne wskaźniki zapobiegają wyciekowi pamięci, co stanowi znaczną poprawę w stosunku do tradycyjnych wskaźników w C.




1B. Nowe biblioteki C++

4. Biblioteka standardowych funkcji IO (`<iostream>`)

W C używa się funkcji takich jak `printf()` i `scanf()`, które mogą być niewygodne i mniej bezpieczne. C++ oferuje bardziej intuicyjne podejście do wejścia/wyjścia dzięki `std::cin` i `std::cout`.

- Przykład w C:

c

 Copy code

```
printf("Podaj imię: ");  
scanf("%s", name);
```

- Przykład w C++:

cpp

 Copy code

```
std::cout << "Podaj imię: ";  
std::cin >> name;
```

Dzięki `std::cout` i `std::cin`, obsługa wejścia i wyjścia w C++ staje się bardziej bezpieczna i elastyczna.




1B. Nowe biblioteki C++

5. `std::thread` – Programowanie wielowątkowe

C++ wprowadza wsparcie dla programowania wielowątkowego za pomocą `std::thread`, co ułatwia tworzenie i zarządzanie wątkami.

- Przykład w C (POSIX Threads):


c

 Copy code

```
pthread_t thread;  
pthread_create(&thread, NULL, &function, NULL);
```

- Przykład w C++ (z `std::thread`):

cpp

 Copy code

```
std::thread t(function);  
t.join();
```

`std::thread` oferuje prostszy i bardziej zintegrowany sposób na tworzenie i synchronizację wątków w porównaniu do bibliotek w C, takich jak POSIX Threads.



1B. Nowe biblioteki C++

Podsumowanie: Nowe biblioteki w C++

- STL ułatwia operacje na danych dzięki gotowym kontenerom i algorytmom.
- `std::string` eliminuje problemy z ręcznym zarządzaniem pamięcią dla tekstu.
- Inteligentne wskaźniki w C++ automatyzują zarządzanie pamięcią.
- `std::cout` i `std::cin` upraszczają operacje wejścia/wyjścia.
- `std::thread` zapewnia wbudowane wsparcie dla wielowątkowości.

C++ nie tylko dziedziczy funkcje z C, ale wprowadza nowoczesne narzędzia, które usprawniają proces programowania, czyniąc go bardziej intuicyjnym i bezpiecznym.



1C. NOWE KONWENCJE NAZEWNICTWA PLIKÓW C++

1. Pliki źródłowe w C++

W C++ pliki źródłowe zazwyczaj mają rozszerzenia `.cpp`, które odróżniają je od plików C z rozszerzeniem `.c`.

- C:
 - Pliki źródłowe mają rozszerzenie `.c` (np. `main.c`).
- C++:
 - Pliki źródłowe w C++ mają rozszerzenie `.cpp` (np. `main.cpp`).
 - Alternatywnie mogą też być używane rozszerzenia `.cc`, `.cxx`, ale `.cpp` jest najczęściej stosowane.

Przykład nazwy pliku w C++:

`main.cpp` – zawiera definicje klas, funkcji, główne metody programu.



1C. NOWE KONWENCJE NAZEWNICTWA PLIKÓW C++

2. Pliki nagłówkowe

Zarówno w C, jak i C++, pliki nagłówkowe zawierają deklaracje funkcji, klas oraz stałych, które są wykorzystywane w plikach źródłowych. W C++ istnieje jednak dodatkowy nacisk na struktury obiektowe, co powoduje bardziej rygorystyczne zasady.

- C:
 - Pliki nagłówkowe mają rozszerzenie `.h` (np. `my_header.h`).
- C++:
 - Pliki nagłówkowe również często używają rozszerzenia `.h` (np. `my_class.h`).
 - C++ wprowadza również rozszerzenie `.hpp` jako bardziej specyficzne dla C++ nagłówki (np. `my_class.hpp`), aby podkreślić, że nagłówek dotyczy specyficznego dla C++ kodu.

Przykład nazwy pliku w C++:

`Car.hpp` – zawiera deklarację klasy `Car`.



1C. NOWE KONWENCJE NAZEWNICTWA PLIKÓW C++

3. Pliki nagłówkowe z szablonami (templates)

C++ oferuje wsparcie dla **szablonów (templates)**, które pozwalają tworzyć klasy i funkcje dla różnych typów danych. W przypadku plików nagłówkowych, które zawierają szablony, stosuje się te same rozszerzenia co dla zwykłych nagłówków (`.h` lub `.hpp`).

- Przykład:

cpp

 Copy code

```
// Car.hpp
template<typename T>
class Car {
    T model;
    // ...
};
```




1C. NOWE KONWENCJE NAZEWNICTWA PLIKÓW C++

4. Rozdzielanie plików nagłówkowych i źródłowych

W C++ nacisk kładzie się na oddzielanie deklaracji od definicji. Pliki nagłówkowe (`.hpp`) zawierają tylko deklaracje klas, funkcji i zmiennych, a pliki źródłowe (`.cpp`) zawierają ich definicje.

- Plik nagłówkowy (`.hpp`): Zawiera deklaracje klasy `Car` :

cpp

 Copy code

```
// Car.hpp
class Car {
public:
    void drive();
};
```

- Plik źródłowy (`.cpp`): Definicja metody klasy `Car` :

cpp

 Copy code

```
// Car.cpp
#include "Car.hpp"

void Car::drive() {
    std::cout << "Driving..." << std::endl;
}
```



1C. NOWE KONWENCJE NAZEWNICTWA PLIKÓW C++

5. Przestrzeganie konwencji nazewnictwa plików

Konwencje nazewnictwa plików w C++ podążają za bardziej **zorganizowanym i modularnym** podejściem. Organizacja projektu ma kluczowe znaczenie, zwłaszcza w większych systemach. Istnieje kilka popularnych konwencji:

- **Snake case:** Wszystkie litery są małe, słowa są oddzielone podkreśleniami (np. `my_class.hpp`, `utils.cpp`).
- **Camel case:** Słowa są łączone, zaczynając od małej litery, a kolejne słowa zaczynają się z wielkiej litery (np. `myClass.cpp`).
- **Pascal case:** Każde słowo zaczyna się z wielkiej litery (np. `Car.hpp`, `MainApplication.cpp`).



1C. NOWE KONWENCJE NAZEWNICTWA PLIKÓW C++

6. Przykład organizacji projektu w C++

Przy większych projektach w C++ ważne jest odpowiednie rozdzielanie plików nagłówkowych i źródłowych oraz grupowanie ich w folderach.

Struktura projektu:

bash

 Copy code

```
/project
```

```
  /src
```

```
    main.cpp
```

```
    Car.cpp
```

```
    Engine.cpp
```

```
  /include
```

```
    Car.hpp
```

```
    Engine.hpp
```



1C. NOWE KONWENCJE NAZEWNICTWA PLIKÓW C++

Podsumowanie: Nowe konwencje nazewnictwa plików w C++

- **Rozszerzenia plików:** W C++ używamy `.cpp` dla plików źródłowych i `.hpp` lub `.h` dla plików nagłówkowych.
- **Oddzielenie deklaracji od definicji:** C++ promuje oddzielanie deklaracji (w plikach nagłówkowych) od definicji (w plikach źródłowych).
- **Organizacja projektu:** Ważne jest utrzymanie dobrze zorganizowanej struktury plików i folderów, co jest kluczowe w większych projektach C++.

Przejsie na C++ wymaga adaptacji do nowych konwencji nazewnictwa plików, które promują modularność i lepsze zarządzanie kodem w dużych projektach.




1D. NOWY SYNTAX

1. Definicja klas i obiektów

Jedną z największych różnic między C i C++ jest wprowadzenie klas i obiektów. W C++ możesz tworzyć klasy z metodami i atrybutami, co umożliwia bardziej złożone operacje na danych.

- Przykład w C (struktura):


c

 Copy code

```
struct Car {  
    char* make;  
    int year;  
};
```

- Przykład w C++ (klasa):

cpp

 Copy code

```
class Car {  
public:  
    std::string make;  
    int year;  
  
    void drive() {  
        std::cout << "Driving the car!" << std::endl;  
    }  
};
```



1D. NOWY SYNTAX

2. Konstruktor i destruktor

C++ wprowadza **konstruktor** i **destruktor**, które automatycznie zarządzają tworzeniem i usuwaniem obiektów. W C takie operacje wymagają ręcznego zarządzania pamięcią za pomocą `malloc()` i `free()`.

- **Konstruktor:** Służy do inicjalizacji obiektów w momencie ich tworzenia.
- **Destruktor:** Służy do czyszczenia pamięci lub innych zasobów, gdy obiekt przestaje być potrzebny.
- **Przykład w C++:**

cpp

Copy code

```
class Car {  
public:  
    std::string make;  
    int year;  
  
    // Konstruktor  
    Car(std::string m, int y) : make(m), year(y) {}  
  
    // Destruktor  
    ~Car() {  
        std::cout << "Destruktor wywołany!" << std::endl;  
    }  
};
```






1D. NOWY SYNTAX

3. Operator zakresu ::

C++ wprowadza **operator zakresu** (`::`), który jest używany do odwoływania się do elementów klas, przestrzeni nazw oraz funkcji globalnych.

- Przykład:

cpp

 Copy code

```
class Car {  
public:  
    void drive();  
};  
  
// Definicja metody poza klasą z użyciem operatora zakresu  
void Car::drive() {  
    std::cout << "Driving the car!" << std::endl;  
}
```

W C nie ma odpowiednika tego operatora, ponieważ nie ma koncepcji klas ani przestrzeni nazw.




1D. NOWY SYNTAX

4. Szablony (Templates)

C++ pozwala na tworzenie generycznych funkcji i klas za pomocą **szablonów (templates)**, co nie jest dostępne w C. Szablony umożliwiają pisanie funkcji lub klas, które mogą działać na różnych typach danych bez potrzeby wielokrotnego pisania kodu.

- Przykład szablonu funkcji:

cpp

 Copy code

```
template<typename T>
T max(T a, T b) {
    return (a > b) ? a : b;
}

int main() {
    std::cout << max(5, 10) << std::endl;    // Działa z typem int
    std::cout << max(3.5, 2.1) << std::endl; // Działa z typem double
}
```

Szablony w C++ zastępują konieczność pisania osobnych funkcji dla różnych typów danych, co w C jest rozwiązane przez ręczne definiowanie różnych wersji funkcji.



1D. NOWY SYNTAX

5. Obsługa wyjątków (try-catch)

C++ wprowadza strukturalną obsługę wyjątków za pomocą bloków `try-catch`, co pozwala na bardziej bezpieczne zarządzanie błędami w programie. W C takie operacje wymagały ręcznego sprawdzania kodów błędów i skomplikowanych warunków.

- Przykład w C (sprawdzanie kodu błędu):

c

Copy code

```
if (openFile() == -1) {  
    printf("Błąd otwarcia pliku!");  
}
```

- Przykład w C++ (obsługa wyjątków):

cpp

Copy code

```
try {  
    openFile();  
} catch (std::exception& e) {  
    std::cerr << "wyjątek: " << e.what() << std::endl;  
}
```



1D. NOWY SYNTAX

6. Funkcje przeciążone (Function Overloading)

W C++ można przeciążać funkcje, co oznacza, że mogą one mieć tę samą nazwę, ale różnić się liczbą lub typami argumentów. W C nie ma możliwości przeciążania funkcji – każda funkcja musi mieć unikalną nazwę.

- Przykład w C++:

cpp

 Copy code

```
int add(int a, int b) {  
    return a + b;  
}  
  
double add(double a, double b) {  
    return a + b;  
}
```

W C musiałbyś definiować osobne funkcje o różnych nazwach, np. `add_int()` i `add_double()`.



1D. NOWY SYNTAX

7. Przestrzenie nazw (Namespaces)

C++ wprowadza **przestrzenie nazw (namespaces)**, które pomagają organizować kod i unikać konfliktów nazw, szczególnie w dużych projektach.

- Przykład w C++:

cpp

 Copy code

```
namespace MyNamespace {  
    int myFunction() {  
        return 42;  
    }  
}  
  
int main() {  
    std::cout << MyNamespace::myFunction() << std::endl;  
}
```

W C przestrzenie nazw nie istnieją, co może prowadzić do konfliktów nazw w większych projektach.



1D. NOWY SYNTAX

Podsumowanie: Nowa składnia w C++

- **Klasy i obiekty:** C++ wprowadza bardziej zaawansowaną strukturę kodu dzięki klasom i obiektom.
- **Konstruktor i destruktor:** Automatyczne zarządzanie pamięcią dzięki konstruktorom i destruktorom.
- **Operator zakresu `::`:** Ułatwia definiowanie metod klas i korzystanie z przestrzeni nazw.
- **Szablony (templates):** Umożliwiają pisanie generycznych funkcji i klas.
- **Obsługa wyjątków:** C++ wprowadza strukturalną obsługę błędów za pomocą `try-catch`.
- **Funkcje przeciążone:** Możliwość definiowania funkcji o tej samej nazwie, ale różnych parametrach.
- **Przestrzeń nazw:** Pomagają organizować kod i unikać konfliktów nazw.
- **`auto`:** Ułatwia deklarację zmiennych, pozwalając kompilatorowi automatycznie określić ich typ.

Przejście na C++ otwiera nowe możliwości związane z elastycznością i organizacją kodu, co znacząco poprawia produktywność i czytelność dużych projektów.



1E. DOSTĘPNE STANDARDY

1. Standardy języka C

C posiada kilka kluczowych standardów, jednak rozwój języka jest wolniejszy w porównaniu do C++.

Główne standardy C to:

- **C89/C90 (ANSI C)**: Pierwszy ustandaryzowany standard C (1989/1990). Definiuje podstawowe elementy, które są używane do dziś.
- **C99**: Wprowadził kilka nowych funkcji, takich jak typy stałoprzecinkowe (`long long int`), deklaracje zmiennych w dowolnym miejscu oraz funkcje z różną liczbą argumentów (variadic macros).
- **C11**: Najnowszy standard języka C, który dodał m.in. obsługę wielowątkowości oraz nowe funkcje do zarządzania pamięcią.



1E. DOSTĘPNE STANDARDY

2. Standardy języka C++

C++ rozwija się znacznie dynamiczniej niż C. Każda nowa wersja standardu wprowadza ulepszenia w składni, nowe biblioteki i narzędzia, które zwiększają produktywność programistów. Oto kluczowe standardy C++:

- **C++98**: Pierwsza oficjalna wersja standardu C++ (wydana w 1998 roku). Oparta na pracy nadrozszerzeniami języka C. Zawierała fundamentalne elementy takie jak klasy, dziedziczenie, polimorfizm, przestrzenie nazw oraz szablony (templates).
- **C++03**: Poprawka standardu C++98, która wprowadziła głównie usprawnienia związane z zgodnością oraz naprawiała drobne błędy w standardzie C++98.
- **C++11**: Jeden z najważniejszych standardów, który wprowadził rewolucyjne zmiany:
 - **Ruchome semantyki (move semantics)**: Optymalizacja wydajności przy pracy z obiektami.
 - `auto`: Automatyczne dedukowanie typów zmiennych.
 - **Lambda expressions**: Umożliwiają definiowanie funkcji anonimowych.
 - **Smart pointers (inteligentne wskaźniki)**: Ułatwiają zarządzanie pamięcią (`std::unique_ptr`, `std::shared_ptr`).
 - `nullptr`: Wprowadzono dedykowany typ `nullptr`, aby unikać problemów z tradycyjnym `NULL`.





1E. DOSTĘPNE STANDARDY

- C++14: Rozszerzenie i udoskonalenie C++11:
 - **Ogólne wyrażenia lambda**: Możliwość dedukcji typu w lambdaach.
 - **Rozwinięcia dla constexpr**: Zwiększono możliwości używania `constexpr` do bardziej skomplikowanych operacji.
- C++17: Wprowadził nowe narzędzia:
 - `std::optional`: Przechowywanie wartości opcjonalnych.
 - `std::variant`: Zastępuje używanie union w bardziej bezpieczny sposób.
 - `std::string_view`: Efektywne zarządzanie łańcuchami znaków bez kopiowania danych.
 - **Zasady porządkowania wywołań funkcji (order of evaluation)**: Ustalono jasne reguły w jakiej kolejności są wykonywane operacje w wyrażeniach.
- C++20: Znacznie rozwija standard, dodając nowoczesne narzędzia i funkcje:
 - **Moduły (Modules)**: Zastępują pliki nagłówkowe i przyspieszają kompilację.
 - **Coroutines**: Wsparcie dla kooperatywnej wielozadaniowości.
 - **Concepts**: Ułatwiają tworzenie bardziej przejrzystych szablonów.
 - **Range-based for loops**: Nowe metody pracy z kontenerami.
- C++23 (planowany): Jeszcze bardziej rozwija i optymalizuje funkcje wprowadzone w poprzednich standardach.





1E. DOSTĘPNE STANDARDY

3. Jak C++ różni się od C pod względem standardów

- **Częstotliwość aktualizacji:** C++ jest regularnie aktualizowany o nowe funkcje, podczas gdy rozwój C jest wolniejszy i bardziej konserwatywny.
- **Zaawansowane narzędzia:** C++ stale wprowadza nowe mechanizmy, takie jak **smart pointers**, **lambdy**, **constexpr**, czy **moduły**, które upraszczają programowanie w porównaniu do manualnych operacji w C.
- **Nowoczesne wzorce projektowe:** Dzięki ciągłym ulepszeniom, C++ wspiera nowoczesne wzorce projektowe i architektoniczne, co sprawia, że kod jest bardziej skalowalny i łatwiejszy do utrzymania.



1E. DOSTĘPNE STANDARDY

4. Jak wybrać odpowiedni standard w C++?

- **C++11:** Dla większości współczesnych projektów zaleca się używanie standardu C++11 jako minimalnego, ponieważ wprowadza on kluczowe funkcje, takie jak semantyka przenoszenia, smart pointers i lambdy.
- **C++17/C++20:** Najnowsze projekty mogą korzystać z C++17 lub C++20, które wprowadzają bardziej zaawansowane narzędzia, takie jak `std::optional`, `std::variant` i moduły.
- **Zgodność z wcześniejszymi standardami:** C++ jest wstecznie kompatybilny, co oznacza, że starszy kod z C lub wcześniejszych wersji C++ powinien działać na nowszych kompilatorach.



1E. DOSTĘPNE STANDARDY

Podsumowanie: Dostępne standardy dla C++

- C++ jest dynamicznie rozwijany, z regularnymi aktualizacjami standardów, co pozwala na bardziej nowoczesne i efektywne programowanie.
- Każdy nowy standard wprowadza usprawnienia w zakresie zarządzania pamięcią, zarządzania typami danych, obsługi wyjątków, wielozadaniowości i wielu innych obszarów.
- Zastosowanie nowych standardów pozwala pisać bardziej czytelny, bezpieczny i wydajny kod, dostosowany do współczesnych potrzeb programistycznych.

C++ dzięki ciągłym aktualizacjom standardów oferuje coraz więcej nowoczesnych funkcji, które znacznie upraszczają i usprawniają pracę w porównaniu do C.



Wprowadzenie do języka programowania C++

- ▶ C++ to język programowania ogólnego przeznaczenia, który został opracowany jako rozszerzenie języka C o paradygmat obiektowy.
- ▶ Jest to język kompilowany.





Praktyczny przykład w visual studio kodzie

Programowanie obiektowe w C++



Konwertowanie języka programowania C++

```
g++ -S main.cpp
```

```
g++ -c main.cpp
```

```
g++ main.cpp -o main
```

```
g++ -include iostream second.cpp -o second.exe
```



Objaśnienie

main.cpp ×

main.cpp > ...

```
1  #include <iostream>
2  // Plik nagłówkowy iostream jest dołączony do programu, aby użyć obiektu cout.
3  // Ten plik nagłówkowy zawiera deklaracje typów danych używanych do operacji wejściowych i wyjściowych.
4  // Plik nagłówkowy iostream musi być dołączony do programu, aby korzystać z obiektu cout.
5  // Obiekt cout służy do wyświetlania danych wyjściowych na ekranie.
6  // Obiekt cout jest obiektem klasy ostream. Klasa ostream jest zdefiniowana w pliku nagłówkowym iostream.
7  // Klasa iostream służy do wykonywania operacji wyjściowych. Klasa ostream wywodzi się z klasy ios.
8  //Klasa iostream jest zdefiniowana w pliku nagłówkowym iostream. Klasa ios służy do wykonywania operacji wejścia i wyjścia.
9
10 // Instrukcja using namespace std; służy do włączenia przestrzeni nazw std do programu.
11 // Przestrzeń nazw std zawiera wszystkie klasy i funkcje biblioteki standardowej języka C++.
12 using namespace std;
13
```

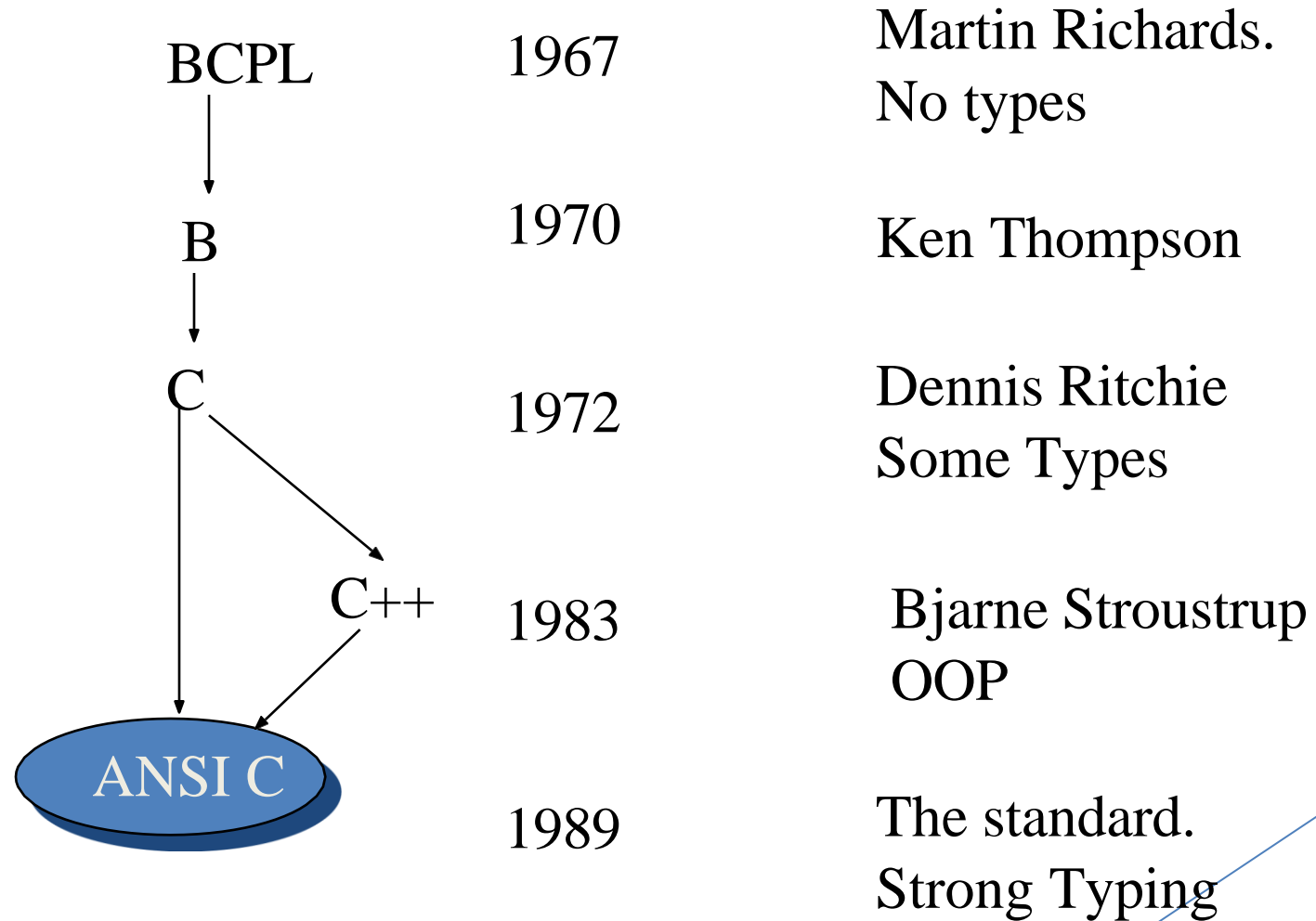



Objaśnienie

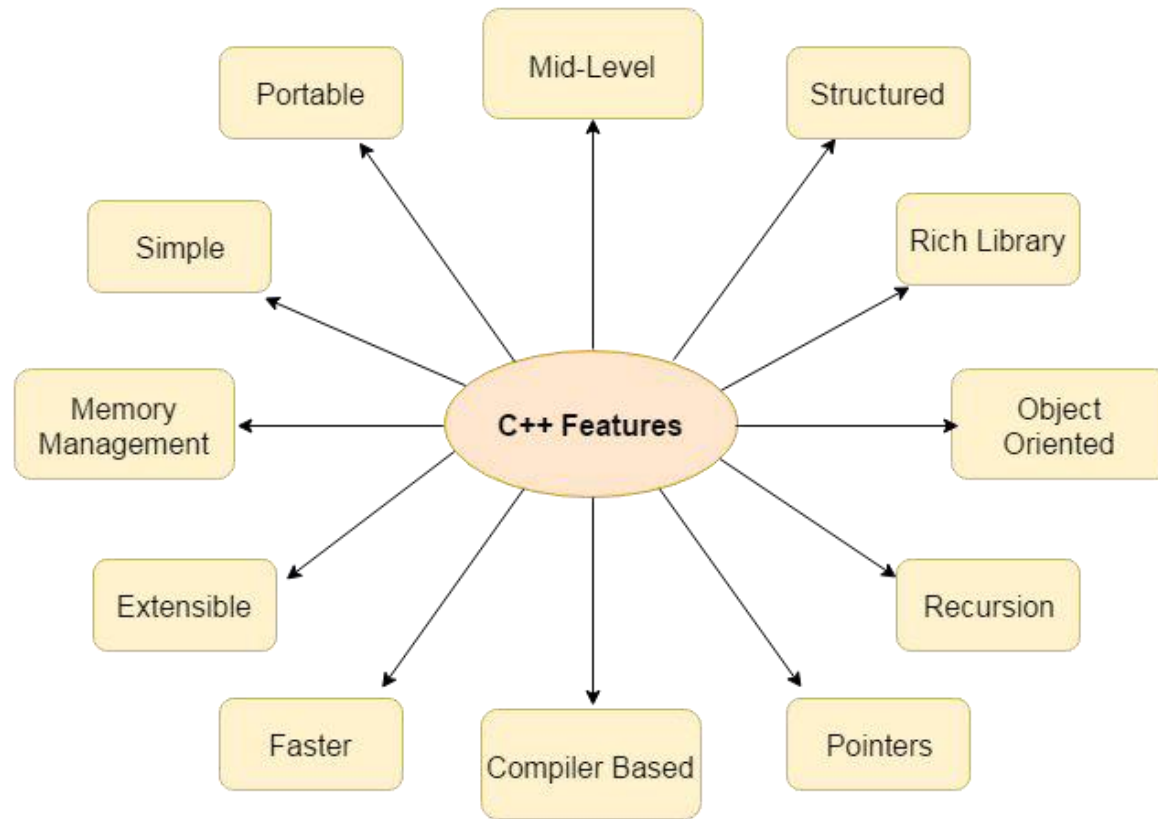
```
14 // Funkcja main() jest punktem wejścia programu w języku C++.
15 // Program wykonuje swoje działanie od tej funkcji.
16 // Funkcja main() zwraca wartość całkowitą, która jest kodem wyjścia programu.
17 // Kod wyjścia 0 oznacza, że program zakończył się poprawnie, a inny kod oznacza, że program zakończył się z błędem.
18 // Funkcja main() jest funkcją typu int, co oznacza, że zwraca wartość całkowitą.
19 // Funkcja main() nie przyjmuje żadnych argumentów.
20 int main() {
21     // Obiekt cout służy do wyświetlania danych wyjściowych na ekranie w języku C++.
22     // Operator << jest używany do przesyłania danych do obiektu cout.
23     // W tym przypadku, napis "Hello, World!" jest przesyłany do obiektu cout, który wyświetla go na ekranie.
24     // endl jest manipulatorem strumienia, który dodaje znak nowej linii po wyświetleniu tekstu.
25     // Manipulator endl powoduje, że kursor przechodzi do nowej linii po wyświetleniu tekstu.
26     // W ten sposób kolejne dane wyświetlane są w nowej linii.
27     // W ten sposób, napis "Hello, World!" zostanie wyświetlony na ekranie w nowej linii.
28     cout << "Hello, World!" << endl;
29     // Instrukcja return służy do zwracania statusu zakończenia programu do systemu operacyjnego.
30     return 0;
31 }
```



Historia C++



C++ Features

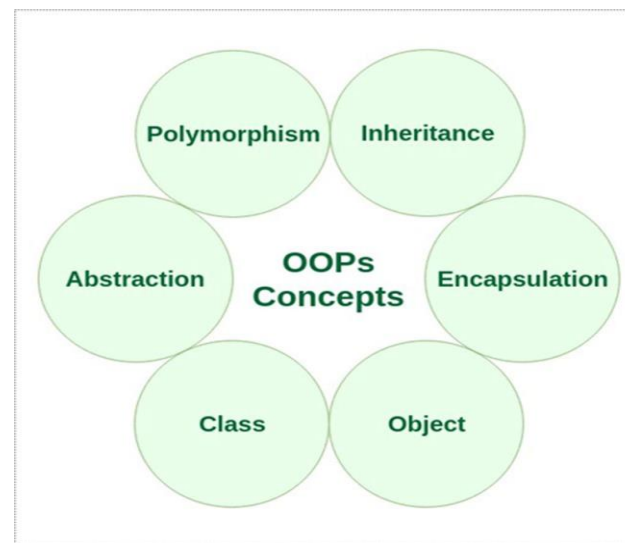


Programowanie obiektowe w C++



sPO (system programowania obiektowego)

- ▶ Obiekt oznacza rzeczywistą jednostkę słowną, taką jak długopis, krzesło, stół itp.
- ▶ Programowanie obiektowe to metodologia lub paradygmat projektowania programu przy użyciu klas i obiektów.
- ▶ Upraszcza tworzenie i utrzymywanie oprogramowania poprzez zapewnienie pewnych koncepcji:
- ▶ Obiekt
- ▶ Class
- ▶ Dziedziczenie
- ▶ Polimorfizm
- ▶ Abstrakcja
- ▶ Enkapsulacja





Koncepcje sPO

- ▶ **Obiekt:**
 - ▶ Każda jednostka, która ma stan i zachowanie, jest znana jako obiekt. Na przykład: krzesło, długopis, stół, klawiatura, rower itp. Obiekt może być fizyczny lub logiczny.
- ▶ **Klasa:**
 - ▶ Zbiór obiektów nazywany jest klasą. Jest to byt logiczny.
- ▶ **Dziedziczenie:**
 - ▶ Gdy jeden obiekt nabywa wszystkie właściwości i zachowania obiektu nadrzędnego, jest to znane jako dziedziczenie. Zapewnia możliwość ponownego wykorzystania kodu. Służy do osiągnięcia polimorfizmu w czasie wykonywania.
- ▶ **Polimorfizm:**
 - ▶ Gdy jedno zadanie jest wykonywane na różne sposoby, jest to znane jako polimorfizm. Na przykład: aby przekonać klienta w inny sposób, aby narysować coś np. kształt lub prostokąt itp.
 - ▶ W C++ używamy przeciążania funkcji i nadpisywania funkcji, aby osiągnąć polimorfizm.



Koncepcje sPO

- ▶ Abstrakcja:
- ▶ Ukrywanie wewnętrznych szczegółów i pokazywanie funkcjonalności jest znane jako abstrakcja. Na przykład: połączenie telefoniczne, nie znamy wewnętrznego przetwarzania. W C++ używamy klas abstrakcyjnych i interfejsów, aby osiągnąć abstrakcję.
- ▶ Enkapsulacja:
- ▶ Wiązanie (lub zawijanie) kodu i danych w pojedynczą jednostkę jest znane jako enkapsulacja. Na przykład: kapsułka jest opakowana w różne leki.



Przewaga PO nad językiem programowania zorientowanym na procedury

- ▶ PO ułatwia rozwój i utrzymanie, podczas gdy w przypadku języku programowania zorientowanym na procedury nie jest łatwo zarządzać, jeśli kod rośnie wraz ze wzrostem rozmiaru projektu.
- ▶ Operacje operacyjne zapewniają ukrywanie danych, podczas gdy w języku programowania proceduralnego dostęp do danych globalnych można uzyskać z dowolnego miejsca.
- ▶ PO zapewniają możliwość znacznie bardziej efektywnej symulacji rzeczywistych zdarzeń. Możemy zapewnić rozwiązanie rzeczywistego problemu, jeśli używamy języka programowania obiektowego.



C++ - Download & Installation

Installing Visual Studio Code for C++

1. Download Visual Studio Code:

- Go to [Visual Studio Code Download](#).
- Choose the installer for your operating system (Windows, Mac, or Linux).

2. Install Visual Studio Code:

- Run the downloaded installer and follow the installation prompts.

3. Install the Necessary Extensions:

- C++ Extension:
 - Open VS Code.
 - Go to the Extensions panel (`Ctrl + Shift + X` or click the Extensions icon).
 - Search for and install C/C++ IntelliSense by Microsoft.



C++ - Download & Installation

Setting Up Visual Studio Code for C++ (Windows, Mac, Linux)

1. Install C++ Compiler:

- Windows:
 - Install MinGW or Visual Studio Build Tools:
 - Download MinGW from [MinGW](#).
 - Install `gcc`, `g++`, and `gdb`.
- Mac:
 - Install Xcode Command Line Tools by running:

```
bash
```

[Copy code](#)

```
xcode-select --install
```

- Linux:
 - Install `g++` using the following command:

```
bash
```

[Copy code](#)

```
sudo apt install build-essential
```



C++ - Download & Installation

Setting Up Visual Studio Code for C++ (Windows, Mac, Linux)

2. Configure C++ Build Task:

- In VS Code, open the **Command Palette** (`Ctrl + Shift + P`).
- Type and select "Tasks: Configure Default Build Task".
- Choose "C++: g++ build active file".
- This will create a `tasks.json` file for your project.

3. First-Time Setup:

- Open VS Code.
- Create a folder for your C++ project.
- Inside the folder, create a new file `main.cpp` and write your C++ code.

4. Run the Code:

- Compile the code by pressing `Ctrl + Shift + B` (runs the build task).
- If successful, an executable file will be generated. Run the file from the terminal:

```
bash
```

[Copy code](#)

```
./your_executable_name
```



Praktyczny przykład w Wisual Studio Kodzie

Programowanie obiektowe w C++



Objaśnienie

```
11 // Klasa Room jest klasą użytkownika, która zawiera pola i metody do obliczania pola powierzchni i objętości pokoju.
12 // Klasa Room zawiera trzy pola: length, breadth i height, które przechowują długość, szerokość i wysokość pokoju.
13 // Klasa Room zawiera dwie metody: calculate_area() i calculate_volume(), które obliczają pole powierzchni i objętość pokoju.
14 class Room {
15     // Sekcja prywatna klasy Room zawiera pola i metody, które są dostępne tylko w obrębie klasy.
16     public:
17     // Pola klasy Room przechowują długość, szerokość i wysokość pokoju.
18         double length;
19         double breadth;
20         double height;
21
22         // Metoda calculate_area() oblicza pole powierzchni pokoju, mnożąc długość i szerokość.
23         double calculate_area(){
24             return length * breadth;
25         }
26
27         // Metoda calculate_volume() oblicza objętość pokoju, mnożąc długość, szerokość i wysokość.
28         double calculate_volume(){
29             return length * breadth * height;
30         }
31     };
```



Objaśnienie

```
37 // Funkcja main() jest punktem wejścia programu w języku C++.
38 // Program wykonuje swoje działanie od tej funkcji.
39 // Funkcja main() zwraca wartość całkowitą, która jest kodem wyjścia programu.
40 // Kod wyjścia 0 oznacza, że program zakończył się poprawnie, a inny kod oznacza, że program zakończył się z błędem.
41 // Funkcja main() jest funkcją typu int, co oznacza, że zwraca wartość całkowitą.
42 // Funkcja main() nie przyjmuje żadnych argumentów.
43 int main(){
44     // Obiekt room1 klasy Room jest tworzony, aby obliczyć pole powierzchni i objętość pokoju.
45     Room room1;
46     // Pola obiektu room1 są inicjalizowane z wartościami długości, szerokości i wysokości pokoju.
47     room1.length = 42.5;
48     room1.breadth = 30.8;
49     room1.height = 19.2;
50     // Obliczone pole powierzchni i objętości pokoju są wyświetlane na ekranie.
51     cout << "Area of Room = " << room1.calculate_area() << endl;
52     cout << "Volume of Room = " << room1.calculate_volume() << endl;
53     // Instrukcja return służy do zwracania statusu zakończenia programu do systemu operacyjnego.
54     // W tym przypadku, zwracana jest wartość 0, co oznacza, że program zakończył się poprawnie.
55     return 0;
56 }
```



Dziękuję za uwagę!