

C++ wciągnie Was

tutor version 1.23

środowiska programistyczne dla obiektowych aplikacji konsolowych

zintegrowane środowisko programistyczne (posiada narzędzia potrzebne przy pisaniu programu: edytor kodu z kolorowaną składnią, kompilator, debugger, narzędzia do refaktoryzacji kodu itp.)



[Visual Studio](#) (Windows) – IDE (Integrated Development Environment), Community 2019 – free (C#, F#, Visual Basic, C++, Python, JavaScript/TypeScript)



[Visual Studio](#) dla komputerów Mac (macOS) - IDE



[Visual Studio Code](#) (Windows, Linux, macOS) – edytor open source (free) z wieloma rozszerzeniami (m.in. JavaScript, Python, Java, C/C++, C#, Powershell, HTML/CSS. PHP) //trzeba samodzielnie zainstalować kompilator i debugger



[Code::Blocks](#) - The free C/C++ and Fortran IDE

środowiska programistyczne dla obiektowych aplikacji konsolowych



[Apache NetBeans](#) – IDE (Java, php, C++ ..., free)



[Eclipse](#) – IDE (C++, Java ..., free)



[CodeLite](#) - cross platform IDE specialized in C, C++, PHP and JavaScript (free)
//trzeba samodzielnie zainstalować kompilator i debugger



[CLion](#) - A cross-platform IDE for C and C++ (komercyjne)

[C++ compiler online](#) (free)

Functional programming

Programowanie funkcyjne

(programy są konstruowane przez stosowanie i komponowanie funkcji np.:
programista stosuje funkcję, która zwraca wynik swojego działania,
nie interesuje go, jak dokładnie przebiega wykonanie krok po kroku. Język np. SQL)

Declarative programming

Programowanie deklaratywne

(programista jedynie deklaruje właściwości pożądanego wyniku, ale nie jak go obliczyć,
skupia się nad tym **co chce osiągnąć**)

Programming paradigm

Paradygmat programowania

(podejście do rozwiązania problemu,
konstrukcja programu)

Imperative programming

Programowanie imperatywne

(sekwencja instrukcji zmieniająca stan programu,
programista pisze szczegółową sekwencję kroków wykonującą zadanie
używając instrukcji sterujących (np. if, for, while),
skupia się nad tym **jak zrobić zadanie**)

Procedural programming

Programowanie proceduralne

(instrukcje do wykonania
są grupowane w procedury i funkcje)

Object-oriented programming

Programowanie zorientowane obiektowo

(grupuje instrukcje oraz stan programu na którym operują,
czyli podstawą są obiekty (z metodami i polami) i ich wzajemne interakcje)

C++ język kompilowany do kodu maszynowego (uruchamianego przez procesor) *dyrektywa preprocesora*
preprocessor directive

Od kodu źródłowego do pliku wykonywalnego

From source code to exe

Kod źródłowy
Source code

```
main.cpp x
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      cout << "Hello world!" << endl;
8      return 0;
9  }
10
```

tu działa preprocesor

Preprocessing

kompilacja

Compilation

Kompilator zamienia kod źródłowy na kod w asemblerze – języku danego procesora

Compiler converts source code into assembly code (for given processor)

Asembler zamienia kod w asemblerze na kod binarny (zero jedynekowy), wynikiem jest tzw. plik obiektowy

Assembler assembles that code into machine code producing binary file (object file)

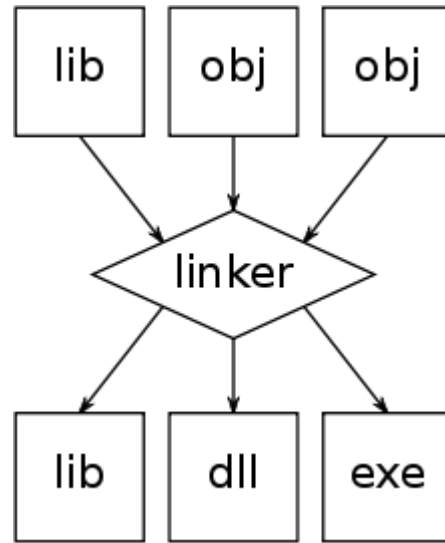
linkowanie

Linking

Linker tworzy z plików obiektowych bibliotekę współdzieloną lub plik wykonywalny (.exe w Windows)

Linker produces the final compilation output (shared library or executable) from the object files

*wkleja zawartość biblioteki (pliku) iostream
do kodu źródłowego
replaces #include directives
with the content of the respective files,
handles other directives too*



An illustration of the linking process. Object files and [static libraries](https://en.wikipedia.org/wiki/Static_library) are assembled into a new library or executable

Java język kompilowany
do kodu bajtowego
uruchamianego
przez JVM

Java source files
(.java)

```
class Foo {  
    /* ... */  
}
```

Python source files
(.py)

```
def f(x):  
    print x  
    ...
```

Kod źródłowy
Source code

javac

jython

*Kompiluje pliki źródłowe Pythona do kodu
maszyny wirtualnej (bytecode)*
Compiler

*Kompiluje pliki źródłowe Javy do kodu
maszyny wirtualnej (bytecode)*
Compiler

Java bytecode files
(.class/.jar)

Java bytecode files
(.class/.jar)

```
...  
iconst_0  
iload  
istore_1  
jsr 19  
iload_1  
...
```

```
...  
istore_1  
iload_1  
jsr 19  
iconst_0  
iload  
...
```

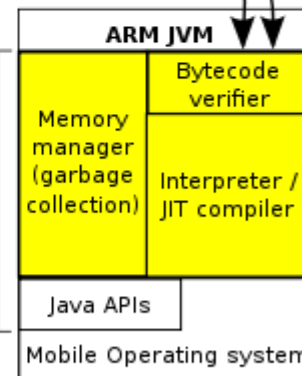
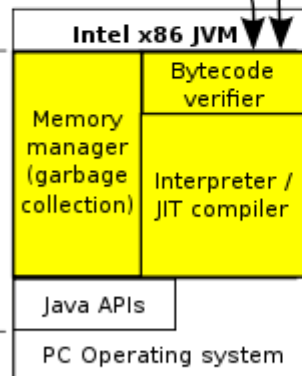
*Wirtualna Maszyna Javy wykonuje
kod bajtowy*
Java Virtual Machine

Java Virtual Machine

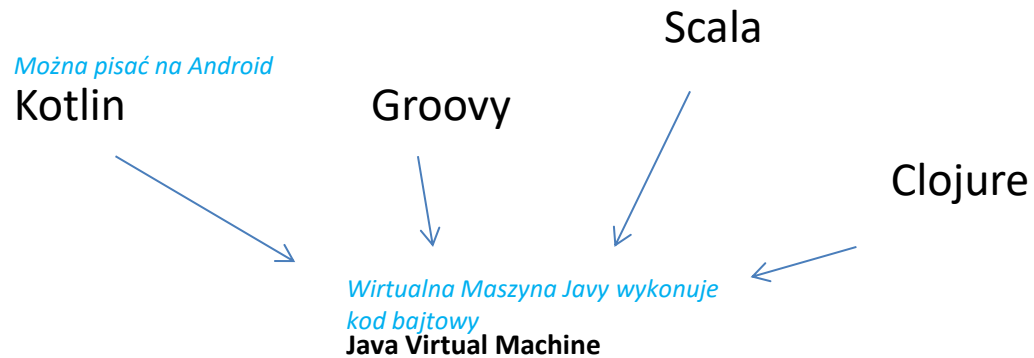
Środowisko uruchomieniowe
Java Runtime Environment

JRE

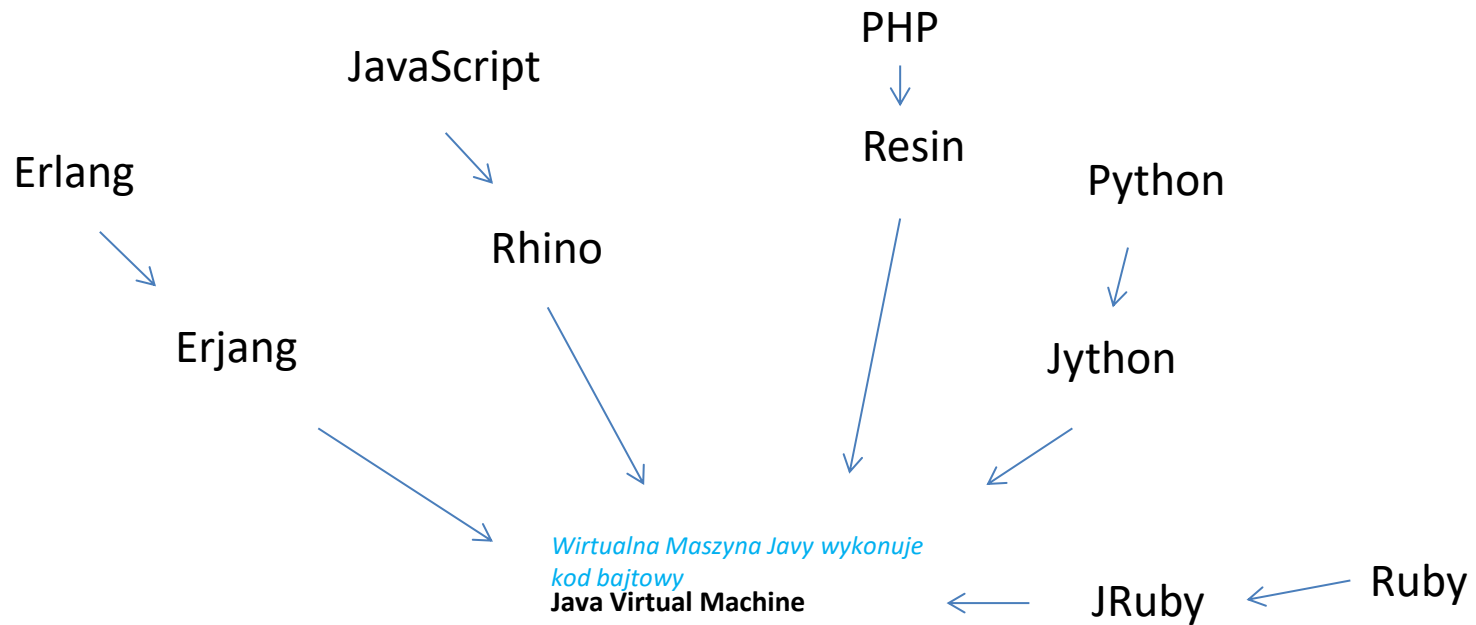
JRE



Języki (nowe) wykonywane na JVM

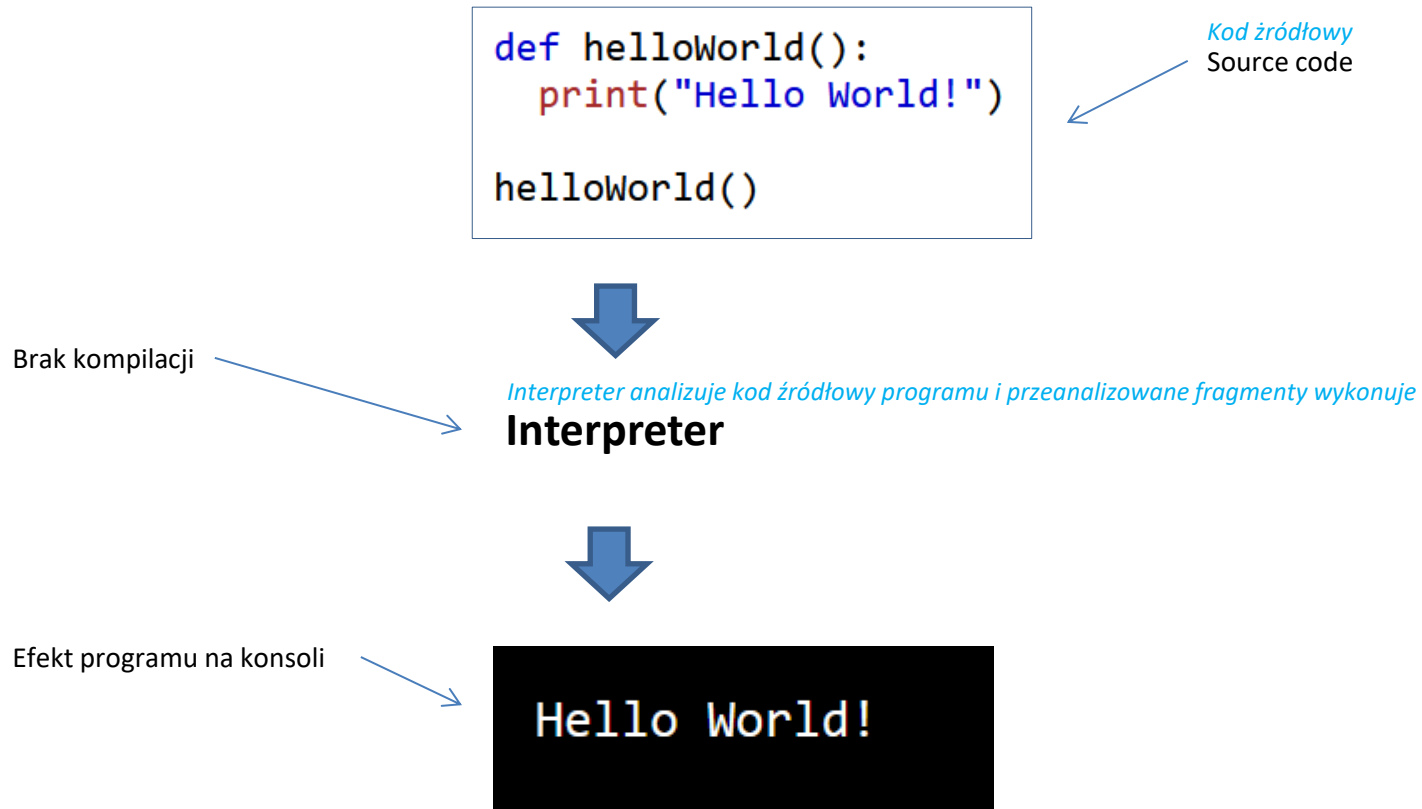


Implementacje istniejących języków na JVM



Python

język interpretowany



C# język kompilowany do kodu pośredniego CIL wykonywanego w środowisku uruchomieniowym np .NET Framework

```
class Program
{
    static void Main()
    {
        System.Console.WriteLine("Hello World!");
    }
}
```

Kod źródłowy
Source code



kompilacja
Compilation

W wyniku kompilacji kodu źródłowego powstaje kod pośredni CIL
(Common Intermediate Language)



.NET Framework

środowisko uruchomieniowe (jest ich więcej)
zainstalowane na komputerze



Efekt programu na konsoli

Hello World!

Hour of code

Learn

← → ↺ 🏠

🔒 https://hourofcode.com/pl/learn

☆ 🔍 Szukaj

🔒 📖 ☰

Stwórz grę Flapp

Klasy 2+ | Bloki

Kodable (zaprasza)

Od dzieci, które uczą się nowoczesne przeglądani

CODE COME



Napisz swój pierwszy program komputerowy

Code.org

Klasy 2+ | Bloki

Poznaj podstawowe pojęcia informatyki, przeciągając i upuszczając bloki z kodem. Jest to samouczek w stylu gry, wykorzystujący filmiki z udziałem Billa Gatesa, Marka Zuckerberga, Angry Birds i Plants vs. Zombies. Poznaj pętle "powtarzaj", warunki oraz podstawowe algorytmy. Dostępne w 37 językach.

Rozpocznij

Więcej zasobów	🔗 Notatki nauczyciela
Krótki link	https://hourofcode.com/code
Przygotowanie ucznia	Początkujący
Technologia w klasie	Wszystkie współczesne przeglądarki, Android, iOS
Tematy	Informatyka tylko
Rodzaj aktywności	Samouczek
Długość	Jedna godzina, Godzina z kontynuacją
Języki	Angielski, arabski, bułgarski, chiński, chorwacki, duński, holenderski, fiński, francuski, niemiecki, grecki, islandzki, indonezyjski, włoski, japoński, koreański, norweski, perski, polski, portugalski, rumuński, rosyjski, hiszpański, szwedzki, turecki, ukraiński i 19 więcej

move forward by 100 pixels

turn right by 90 degrees



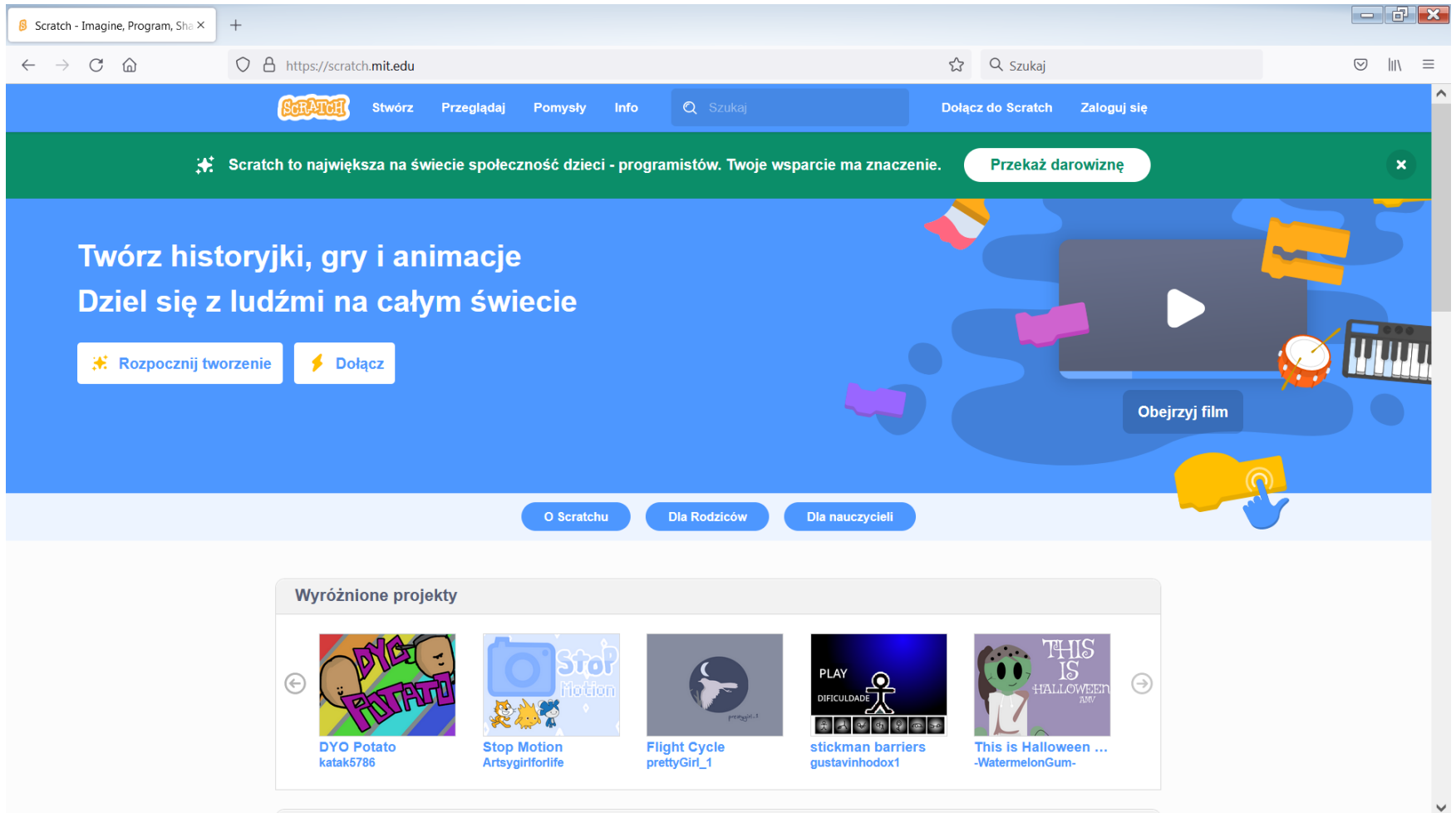








Scratch



The screenshot shows the Scratch website homepage in a web browser. The browser's address bar displays <https://scratch.mit.edu>. The website has a blue header with the Scratch logo and navigation links: Stwórz, Przeglądaj, Pomysły, Info, and a search bar labeled 'Szukaj'. On the right of the header are links for 'Dołącz do Scratch' and 'Zaloguj się'. A green banner below the header contains the text 'Scratch to największa na świecie społeczność dzieci - programistów. Twoje wsparcie ma znaczenie.' and a button 'Przełącz darowiznę'. The main content area has a blue background with the text 'Twórz historyjki, gry i animacje' and 'Dziel się z ludźmi na całym świecie'. Below this are two buttons: 'Rozpocznij tworzenie' and 'Dołącz'. To the right is a large video player with a play button and a button 'Obejrzyj film'. Below the video player are three buttons: 'O Scratchu', 'Dla Rodziców', and 'Dla nauczycieli'. At the bottom, there is a section titled 'Wyróżnione projekty' showing five project thumbnails with their titles and usernames: 'DYO Potato' by katak6786, 'Stop Motion' by Artsygirlforlife, 'Flight Cycle' by prettyGirl_1, 'stickman barriers' by gustavinhodox1, and 'This is Halloween ...' by -WatermelonGum-.

Scratch - Imagine, Program, Sha X

← → ↺ 🏠

🔒 <https://scratch.mit.edu> ☆ 🔍 Szukaj

Scratch Stwórz Przeglądaj Pomysły Info 🔍 Szukaj Dołącz do Scratch Zaloguj się

🌟 Scratch to największa na świecie społeczność dzieci - programistów. Twoje wsparcie ma znaczenie. [Przełącz darowiznę](#) ✕

Twórz historyjki, gry i animacje
Dziel się z ludźmi na całym świecie

🌟 Rozpocznij tworzenie ⚡ Dołącz

Obejrzyj film

O Scratchu Dla Rodziców Dla nauczycieli

Wyróżnione projekty

←

DYO Potato
katak6786

Stop Motion
Artsygirlforlife

Flight Cycle
prettyGirl_1

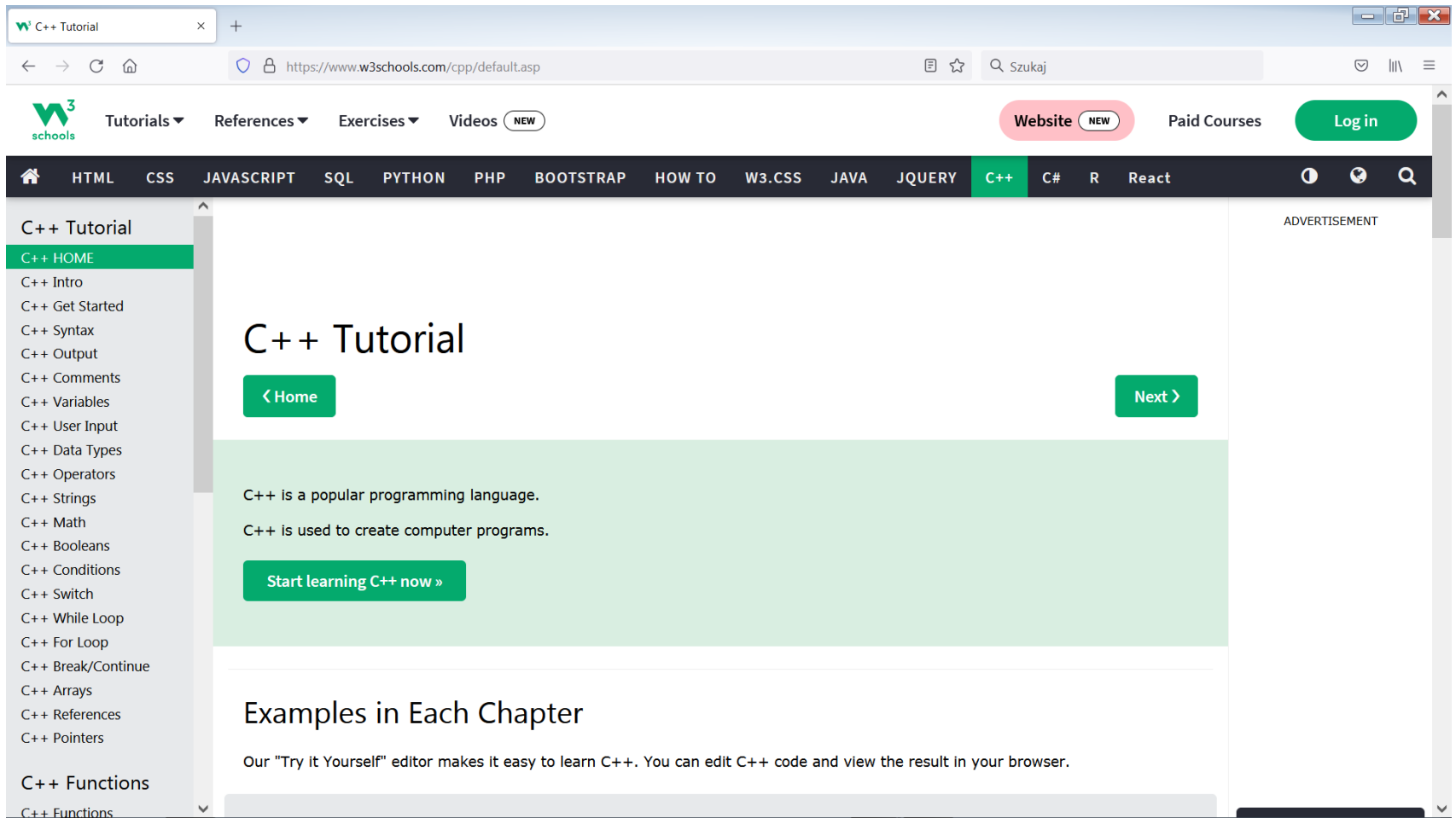
PLAY
DIFICULDADE
stickman barriers
gustavinhodox1

THIS IS HALLOWEEN ...
-WatermelonGum-

→

<https://scratch.mit.edu/>

www.w3schools.com



The screenshot shows the W3Schools website with the C++ Tutorial page selected. The browser's address bar shows the URL <https://www.w3schools.com/cpp/default.asp>. The website's navigation bar includes links for Tutorials, References, Exercises, and Videos, along with buttons for Website, Paid Courses, and Log in. The C++ Tutorial page features a sidebar with a list of topics, a main content area with a large heading and a green call-to-action box, and a right sidebar for advertisements.

C++ Tutorial

C++ HOME

- C++ Intro
- C++ Get Started
- C++ Syntax
- C++ Output
- C++ Comments
- C++ Variables
- C++ User Input
- C++ Data Types
- C++ Operators
- C++ Strings
- C++ Math
- C++ Booleans
- C++ Conditions
- C++ Switch
- C++ While Loop
- C++ For Loop
- C++ Break/Continue
- C++ Arrays
- C++ References
- C++ Pointers
- C++ Functions**
- C++ Functions

C++ Tutorial

[< Home](#)

[Next >](#)

C++ is a popular programming language.

C++ is used to create computer programs.

[Start learning C++ now »](#)

Examples in Each Chapter

Our "Try it Yourself" editor makes it easy to learn C++. You can edit C++ code and view the result in your browser.

ADVERTISEMENT

www.w3schools.com

Zadanie

System operacyjny



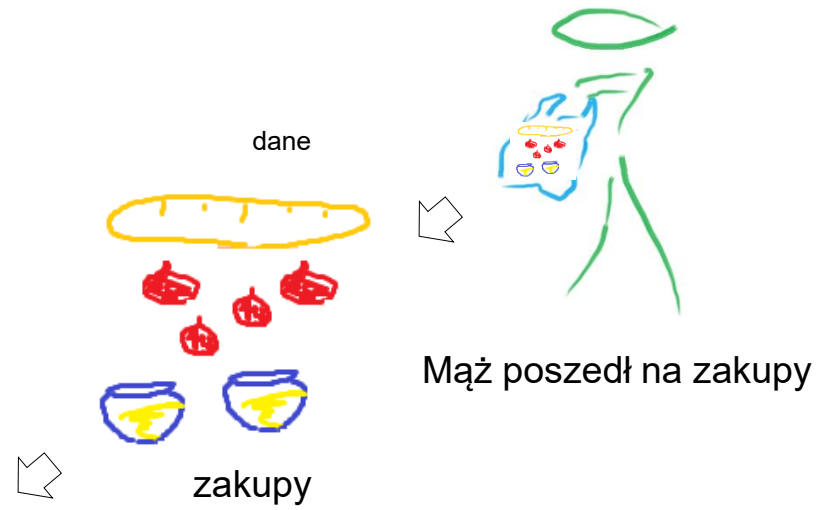
Żona mówi do męża:
„Przygotuj śniadanie dziecku”

program

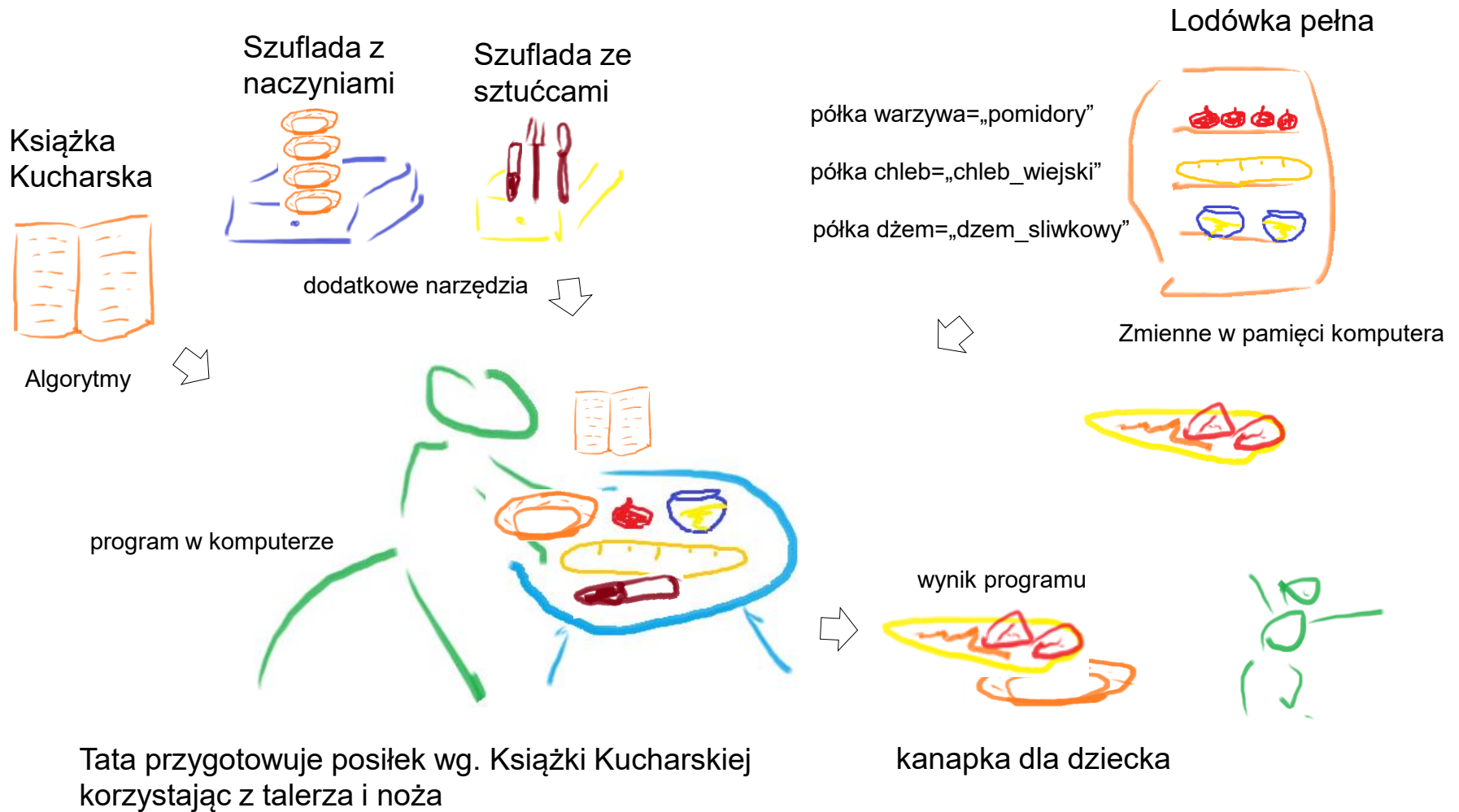


Mąż:
„dobrze”

Zmienne



Program



Zadanie wykonane

System operacyjny



program



Mąż:

return 0

„Synek zjadł kanapkę i nie zwymiotował”

Żona mówi do męża:
„super”

Program kanapka

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

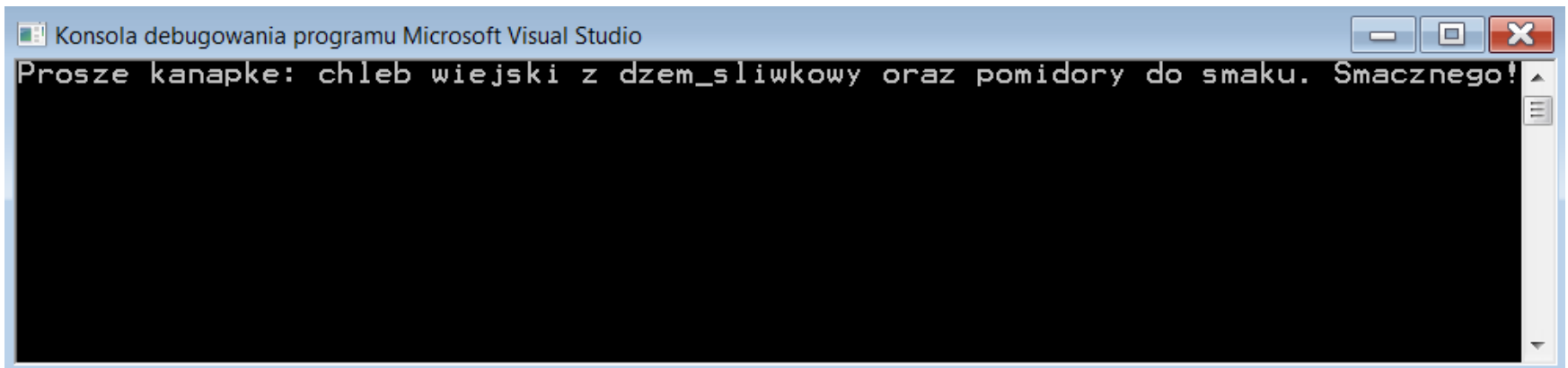
int main()
{
    // lodówka
    string warzywa = "pomidory";
    string chleb = "chleb wiejski";
    string dzem = "dzem_sliwkowy";

    // przygotowanie kanapki z Książki Kucharskiej
    string kanapka = chleb + " z " + dzem;

    // serwujemy kanapkę dziecku
    cout << "Proszę kanapkę: " << kanapka << " oraz " << warzywa << " do smaku. Smacznego!" << endl;

    // dziecko po zjedzeniu kanapki nie z wymiotowało
    return 0;
}
```

Program kanapka



Konsola debugowania programu Microsoft Visual Studio

```
Prosze kanapke: chleb wiejski z dzem_sliwkowy oraz pomidory do smaku. Smaczneho!
```

The image shows a screenshot of a Visual Studio debug console window. The title bar reads 'Konsola debugowania programu Microsoft Visual Studio'. The console output displays the text 'Prosze kanapke: chleb wiejski z dzem_sliwkowy oraz pomidory do smaku. Smaczneho!' in a monospaced font. The window includes standard Windows window controls (minimize, maximize, close) and a vertical scrollbar on the right side.

Bjarne Stroustrup

Bjarne Stroustrup's Homepage

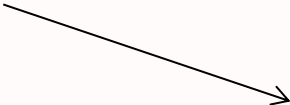
← → ↻ 🏠 🔒 https://www.stroustrup.com 80% ☆ 🔍 Szukaj 📧 📖 ☰

[Morgan Stanley](#) | [Columbia University](#) | [Churchill College, Cambridge](#)

[home](#) | [C++](#) | [FAQ](#) | [technical FAQ](#) | [publications](#) | [WG21 papers](#) | [TC++PL](#) | [Tour++](#) | [Programming](#) | [D&E](#) | [bio](#) | [interviews](#) | [videos](#) | [quotes](#) | [applications](#) | [guidelines](#) | [compilers](#)

Welcome to Bjarne Stroustrup's homepage!

Twórca C++





I'm a Technical Fellow and a Managing Director in the technology division of [Morgan Stanley](#) in New York City and a Visiting Professor in Computer Science at [Columbia University](#).

I designed and implemented [the C++ programming language](#). To make C++ a stable and up-to-date base for real-world software development, I have stuck with its ISO standards effort for 30+ years (so far).

- [A Tour of C++ \(2nd edition\)](#) (a brief - 240 page - tour of the C++ Programming language and its standard library for experienced programmers)
- [The C++ Programming Language \(4th edition\)](#) (an exhaustive description of the C++ Programming language, its standard library, and fundamental techniques for experienced programmers)
- [Programming: Principles and Practice using C++ \(2nd edition\)](#) (a programming text book aimed at beginners who want eventually to become professionals)
- [The Design and Evolution of C++](#) (a book presenting the rationale and design criteria for C++ and its evolution up until 1994).
- [Research and popular papers](#)
- [Technical reports and proposals for the ISO C++ Standard](#)
- [Videos](#)
- [Interviews](#)
- [Biographical material](#).

[Morgan Stanley](#) | [Columbia University](#) | [Churchill College, Cambridge](#)

[home](#) | [C++](#) | [FAQ](#) | [technical FAQ](#) | [publications](#) | [WG21 papers](#) | [TC++PL](#) | [Tour++](#) | [Programming](#) | [D&E](#) | [bio](#) | [interviews](#) | [videos](#) | [quotes](#) | [applications](#) | [guidelines](#) | [compilers](#)

C++ standards

C++ standards		
Year	C++ Standard	Informal name
1998	ISO/IEC 14882:1998	C++98
2003	ISO/IEC 14882:2003	C++03
2011	ISO/IEC 14882:2011	C++11 , C++0x
2014	ISO/IEC 14882:2014	C++14 , C++1y
2017	ISO/IEC 14882:2017	C++17 , C++1z
2020	ISO/IEC 14882:2020	C++20 , C++2a

ISO - International Organization for Standardization

IEC - International Electrotechnical Commission

nazwy zmiennych składają się z liter, cyfr oraz znaku podkreślenia (_).
sequence of one or more letters, digits, or underscore characters

nazwy zmiennych muszą się zaczynać od litery
identifiers shall always begin with a letter

Nazwy zmiennych
Identifiers

ściśła kontrola typu (trzeba zadeklarować typ zmiennej)
strongly-typed language

C++

język C++ rozróżnia małe i duże litery w nazwach zmiennych
„case sensitive” language.

biblioteka
`#include<iostream>`

directive for the preprocessor
dyrektywa preprocesora
(dołączenie pliku iostream - rozszerzenie możliwości c++)

standardowa przestrzeń nazw
(rozwój c++ - dużo bibliotek – brakuje unikatowych nazw zmiennych i funkcji => wymyślono przestrzeń nazw)
`using namespace std;`

typ zwracanej wartości

funkcja main
`int main() {`

instrukcja

komentarz

`cout<<"Hello world!"<<endl;`

`//wypisz na konsoli`

standardowy strumień
wyjściowy

ciąg znaków

end of line

operator wyjścia
(przeciążony operator przesunięcia)

`return 0;`
`}`

instrukcja zwracająca zero

(informacja dla środowiska systemowego, że program zakończył się poprawnie)

Standardowe wejście/wyjście

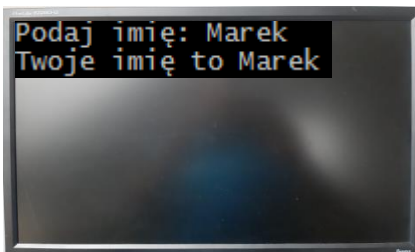
```
#include <iostream>
```

```
using namespace std;
```

standardowe urządzenie wyjściowe



standardowe urządzenie wejściowe



```
int main()  
{
```

```
    string imie;
```

standard output stream

standardowy strumień wyjściowy (ciąg bajtów „Podaj imię: ”)

```
    cout << "Podaj imię: ";
```

insertion operator

operator wstawiania

standard input stream

standardowy strumień wejściowy (ciąg bajtów z klawiatury „Marek”)

```
    cin >> imie;
```

extraction operator

operator wyodrębnienia

```
    cout << "Twoje imię to " << imie << endl;
```

```
    return 0;
```

```
}
```

Standardowe typy danych

Group	Type names*	Notes on size / precision
Character types	<code>char</code>	Exactly one byte in size. At least 8 bits.
	<code>char16_t</code>	Not smaller than <code>char</code> . At least 16 bits.
	<code>char32_t</code>	Not smaller than <code>char16_t</code> . At least 32 bits.
	<code>wchar_t</code>	Can represent the largest supported character set.
Integer types (signed)	<code>signed char</code>	Same size as <code>char</code> . At least 8 bits.
	<code>signed short int</code>	Not smaller than <code>char</code> . At least 16 bits.
	<code>signed int</code>	Not smaller than <code>short</code> . At least 16 bits.
	<code>signed long int</code>	Not smaller than <code>int</code> . At least 32 bits.
	<code>signed long long int</code>	Not smaller than <code>long</code> . At least 64 bits.
Integer types (unsigned)	<code>unsigned char</code>	(same size as their signed counterparts)
	<code>unsigned short int</code>	
	<code>unsigned int</code>	
	<code>unsigned long int</code>	
	<code>unsigned long long int</code>	
Floating-point types	<code>float</code>	
	<code>double</code>	Precision not less than <code>float</code>
	<code>long double</code>	Precision not less than <code>double</code>
Boolean type	<code>bool</code>	
Void type	<code>void</code>	no storage
Null pointer	<code>decltype(nullptr)</code>	

* Nazwa typu pisana pochyło jest opcjonalna

<https://www.cplusplus.com/doc/tutorial/variables/>

Standardowe typy danych – przykładowe zakresy

Typ	Typowy rozmiar w bajtach *	Zakres	zawartość
char	1	-128...127	znak
unsigned char	1	0...255	znak
int	2,4 lub 8	-2 147 483 648...2 147 483 647 (dla 4 B)	liczba całkowita ze znakiem
unsigned int	2,4 lub 8	0...4 294 967 295 (dla 4B)	liczba całkowita dodatnia
short int	2	-32 768...32 767	liczba całkowita ze znakiem
unsigned short int	2	0...65 535	liczba całkowita dodatnia
long int	4	-2 147 483 648...2 147 483 647	liczba całkowita ze znakiem
unsigned long int	4	0...4 294 967 295	liczba całkowita dodatnia
long long int	8	-9.223.372.036.854.775.808... 9.223.372.036.854.775.807	liczba całkowita ze znakiem
unsigned long long int	8	0...18.446.744.073.709.551.615	liczba całkowita dodatnia
float	4	siedem, osiem cyfr znaczących -3.4e38...3.4e38	liczba rzeczywista
double	8	szesnaście cyfr znaczących -1.7e308...1.7e308	liczba rzeczywista
long double	8	szesnaście cyfr znaczących -1.7e308...1.7e308	liczba rzeczywista
bool	1	true, false	wartość logiczna

* zależy od użytego procesora, kompilatora i systemu operacyjnego – sprawdzić za pomocą sizeof()

** zakresy nie są ściśle określone w standardzie języka

Wielkości i limity zmiennych

```
cout << "Wielkości zmiennych:\n\n";
cout << "char " << sizeof(char) << "B" << endl;
cout << "signed short int: " << sizeof(signed short int) << "B" << endl;
cout << "int: " << sizeof(int) << "B" << endl;
cout << "int min: " << INT_MIN << endl;
cout << "int max: " << INT_MAX << endl;
cout << "long long: " << sizeof(long long) << "B" << endl;
cout << "float: " << sizeof(float) << "B" << endl;
cout << "Number of digits, q, such that a floating-point number with q decimal digits\n"
    "can be rounded into a floating-point representation and back without loss of precision:" << endl;
cout << "FLT_DIG - FLoat DIGits (decimal digits of precision): " << FLT_DIG << " cyfr " << endl;
cout << "FLT_MANT_DIG - FLoat MANTissa DIGits (precision): " << FLT_MANT_DIG << "bits " << endl;
cout << "double: " << sizeof(double) << "B" << endl;
cout << "DBL_DIG - DouBLE DIGits (decimal digits of precision): " << DBL_DIG << " cyfr " << endl;
cout << "DBL_MANT_DIG - DouBLE MANTissa DIGits (precision): " << DBL_MANT_DIG << "bits" << endl;
cout << "long double: " << sizeof(long double) << "B" << endl;
cout << "bool: " << sizeof(bool) << "B" << endl;
```

Wielkości zmiennych:

```
char 1B
signed short int: 2B
int: 4B
int min: -2147483648
int max: 2147483647
long long: 8B
float: 4B
Number of digits, q, such that a floating-point number with q decimal digits
can be rounded into a floating-point representation and back without loss of pre
cision:
FLT_DIG - FLoat DIGits (decimal digits of precision): 6 cyfr
FLT_MANT_DIG - FLoat MANTissa DIGits (precision): 24bits
double: 8B
DBL_DIG - DouBLE DIGits (decimal digits of precision): 15 cyfr
DBL_MANT_DIG - DouBLE MANTissa DIGits (precision): 53bits
long double: 8B
bool: 1B
```

program zmiennie

<https://www.cplusplus.com/reference/climits/>

<https://www.cplusplus.com/reference/cfloat/>

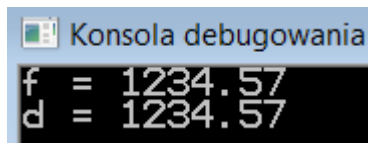
float, double - precision

```
#include <iostream>

int main()
{
    using std::cout;
    using std::endl;

    // bez f na końcu literału liczba byłaby traktowana jak double
    // i konwertowana do float
    float f = 1234.5678901234567890f;
    double d = 1234.5678901234567890;

    cout << "f = " << f << endl;
    cout << "d = " << d << endl;
}
```



Konsola debugowania

f = 1234.57
d = 1234.57

← kompilator zaokrąglił liczbę do 6 cyfr (default)

<https://www.programiz.com/cpp-programming/float-double>

https://eduinf.waw.pl/inf/utils/001_2008/0119.php

float, double - precision

```
#include <iostream>
#include <iomanip> // setprecision

using namespace std;

int main(){

    // bez f na końcu literału liczba byłaby traktowana jak double
    // i konwertowana do float
    float f = 1234.5678901234567890f;
    double d = 1234.5678901234567890;

    // wymusza 20 cyfr po przecinku
    cout << fixed;
    // ustawiamy 20 cyfr po przecinku
    cout << setprecision(20);

    cout << "f = " << f << endl;
    cout << "d = " << d << endl;
}
```

Konsola debugowania programu Microsoft Visual Studio

```
f = 1234.56787109375000000000
d = 1234.56789012345689116046
```

float: 8 cyfr znaczących

double: 16 cyfr znaczących



always use
double

Konwersja dec to bin

W przypadku Integera każda liczba dec może być dokładnie reprezentowana w bin.
Problem jest w przypadku niektórych ułamków:

$$0.1_{10} \Rightarrow 000(1100)_2$$

liczba okresowa, nieskończona

reprezentacja liczby dziesiętnej w komputerze
(w tym przypadku musi zostać zaokrąglona)

<https://docs.python.org/release/2.5.1/tut/node16.html>

https://books.google.pl/books?id=fKE68pdJCD4C&pg=PA14&lpg=PA14&dq=fractions+dec+to+bin+problems&source=bl&ots=nUXrx6yo6-&sig=ACfU3U2UfyAvyM4QdXYpF5DNpjECmUK0Xw&hl=pl&sa=X&ved=2ahUKewi_hbLZylP0AhVHAxAlHVW7AJwQ6AF6BAgeEAM#v=onepage&q=fractions%20dec%20to%20bin%20problems&f=false

Konwersja dec to bin

```
double a = 0;
```

```
while (a != 1)
```

```
{
```

```
    a += 0.1;
```

```
    cout << a << endl;
```

```
}
```

powielamy błąd



```
1330.7
1330.8
1330.9
1331
1331.1
1331.2
1331.3
1331.4
1331.5
1331.6
1331.7
1331.8
1331.9
1332
1332.1
1332.2
1332.3
1332.4
1332.5
1332.6
1332.7
1332.8
1332.9
1333
1333.1
```


Konwersja dec to bin

równe?

7

```
double a = floor((0.1+0.7)*10);  
double b = 8;
```



Liczby zmiennoprzecinkowe - porównanie

```
double a;  
double b;  
//.....
```

```
if (a == b) {  
    //.....  
}  
else {  
    //.....  
}
```



```
double a;  
double b;  
const double epsilon = 0.00001;  
//.....
```

```
if (abs(a - b) <= epsilon) {  
    //.....  
}  
else {  
    //.....  
}
```

Liczby zmiennoprzecinkowe - porównanie

```
double a;  
double b;  
//.....
```

```
if (a != b) {  
    //.....  
}  
else {  
    //.....  
}
```



```
double a;  
double b;  
const double epsilon = 0.00001;  
//.....
```

```
if (abs(a - b) > epsilon) {  
    //.....  
}  
else {  
    //.....  
}
```

Konwersja typu type conversion

konwersja niejawna

implicit conversion

```
short s = 10;
```

```
int i ;
```

konwersja standardowa (dotyczy podstawowych typów danych: short, int, float, double, bool itp.)
standard conversion

```
i = s;
```

promocja typu (wartość s jest promowana z short do int, z typu mniejszego do większego)
type promotion

Promocja (int -> double)

Promocja (float -> double)

```
double d = i + 1.34f //konwersja niejawna zmiennej i z int do double oraz literału zmiennoprzecinkowego float do double  
// promocja typu
```

https://edu.pjwstk.edu.pl/wyklady/pro/scb/PRG2CPP_files/node55.html

<https://www.cplusplus.com/doc/tutorial/typecasting/>

Konwersja typu type conversion

konwersja niejawna

implicit conversion

Konwersja 64-bitowej liczby float na 16-bitowy int spowodowała jeden z najdroższych błędów w historii softwaru – katastrofę rakiety Ariane 5 w 1996 roku

[katastrofa Ariane 5](#)

```
short s = 10;
```

```
int i = INT_MAX // 2 147 483 647
```

```
S = i; // przypisanie max int do short – utrata precyzji (int 4B, short 2B),  
// niezdefiniowane zachowanie programu  
// Visual Studio nie protestuje: printuje s jako -1)
```

utrata dokładności

loss of precision

```
s = {i}; // inicjalizacja klamrowa zmiennej s, kompilator wyrzuci błąd:  
// Visual Studio: Błąd C2397konwersja z „int” do „short” wymaga konwersji zawężającej
```

W przypadku składni inicjalizacji listą (klamrowej) kompilator nie pozwoli na zawężanie typów w ramach niejawnej konwersji (chyba, że wartość z listy mieści się bez utraty dokładności w typie docelowym)

OK

```
s = {10};
```

literał 10 typu integer mieści się w zmiennej typu short

Konwersja typu type conversion

konwersja jawna
explicit conversion

rzutowanie

float f = 3.14F;

double d = (double) f; rzutowanie w stylu języka C // c-like cast notation
lub

double d = double (f); notacja funkcyjna // functional notation
lub

double d = double {f}; // C++11

Deklaracja (i definicja) zmiennej

Aby używać zmiennych, czyli pojemników na dane, trzeba je najpierw zadeklarować:

typ zmiennej



int a;



nazwa

C++ zwraca uwagę na wielkość liter!
(is case sensitive)

//deklaracja i jednocześnie definicja
//zmiennej *a* typu *integer*



ang. liczba całkowita

deklaracja – dzięki niej kompilator wie, że litera *a* oznacza nazwę pojemnika na dane typu całkowitego (na liczby całkowite)

definicja – kompilator rezerwuje w pamięci RAM komputera 4 bajty na zmienną *a* (lub inną wartość bajtów – w zależności od sprzętu i oprogramowania)

Inicjalizacja zmiennej *a*

Po deklaracji (i jednocześnie definicji) zmiennej *a*, **trzeba ją inicjalizować**, czyli przypisać do niej konkretną wartość (liczbę całkowitą):

operator przypisania

literał (stała)



```
a = 1;
```

//inicjalizacja zmiennej *a* liczbą 1

Dlaczego?

po definicji, w pamięci komputera, w miejscu, gdzie jest nasza zmienna, występują „śmieci”, czyli przypadkowe „zera i jedynki”.

Oczywiście program może je zinterpretować jako przypadkową liczbę całkowitą – ale czy o to nam chodziło? Dlatego „wrzucamy” do zmiennej liczbę, żebyśmy wiedzieli co w niej tak naprawdę jest.

Deklaracja (i definicja) zmiennych

```
char znak;           //deklaracja i definicja zmiennej
                     //znakowej znak
```

```
char znak = 'a';     /* inicjalizacja zmiennej znak,
                     //czyli przypisanie litery 'a' do
                     //zmiennej znak
                     */
```

znak w apostrofach



```
char znak= 97;       //przypisanie litery 'a' do
                     //zmiennej znak - kod ASCII
                     //litery podajemy w systemie
                     //dziesiętnym
```

Inicjalizacja zmiennych

```
float a=1.34;    //deklaracja, definicja i inicjalizacja  
                //zmiennej a typu rzeczywistego  
                //(stworzyliśmy pojemnik na liczby  
                //rzeczywiste – z przecinkiem  
                //(w C++ stosujemy kropkę!))
```

```
double b=1.34;  //deklaracja, definicja i inicjalizacja  
                //zmiennej a typu rzeczywistego  
                //podwójnej precyzji (w pojemniku b  
                //będziemy mogli trzymać liczby  
                //dokładniejsze niż w pojemniku a)
```

Inicjalizacja zmiennych

```
bool a=true;           //deklaracja, definicja i inicjalizacja  
                        //zmiennej a typu logicznego  
                        //(stworzyliśmy pojemnik  
                        //na wartości logiczne true lub false)
```

Inicjalizacja zmiennych

*Wyrażenie to sekwencja operatorów i ich operandów,
która określa obliczenie.*

expression

typ_zmiennej nazwa_zmiennej = wyrażenie

int
double
char

a
x
litera

40
c + d
liczba * liczba

operand
operand

operator
operator

operand
operand

wynikiem wyrażenia najczęściej jest jakaś wartość

argumenty operatora

Sposoby inicjalizacji zmiennych

C-inicjalizacja

(inicjalizacja rodem z języka C)

c-like initialization

```
int a = 1;
```

Inicjalizacja kopiująca

copy initialization

```
decltype(a) x = 1;
```

x ma typ taki sam
jak zmienna a
czyli int

// aby zobaczyć jaki ma typ
// zmienna x trzeba znaleźć
// zmienną a – to może być
// niewygodne

Inicjalizacja bezpośrednia

direct initialization

Inicjalizacja konstruktorowa

constructor initialization

```
int a(1);
```

Inicjalizacja klamrowa

brace initialization

Inicjalizacja jednolita

uniform initialization

```
int a{1};
```

można też tak

```
int a = {1};
```

to samo

```
auto a = 1;
```

dedukcja typu
(w tym przypadku
kompilator wydedukuje int)
// zwiększa czytelność kodu
// ale może być potrzebny typ
// zmiennej i wtedy mamy problem

inicjalizacja klamrowa

```
int a = {25};  
double d = {13.5};  
short tab[3] {5,7,9};  
int * ptr = new int[3] {20, 30, 40};
```

przykłady



```
char c {12.5e20} //niejawna konwersja z double na char (liczba inicjująca double przekracza  
zakres char (1 B) - Błąd C2397konwersja z „double” do „char”  
wymaga konwersji zawężającej)
```

Aliasy – nazwy zastępcze zmiennych

alias typu int



```
typedef int integer;  
integer a = 1;
```

alias typu int



```
using calkowita = int;  
calkowita a = 1;
```

Literały (stałe)

Literały to tokeny (znaki) języka C++ reprezentujące stałe wartości osadzone w kodzie języka

Literals are the tokens of a C++ program that represent constant values embedded in the source code.

literał całkowity
literal constant
a = 25;

Podział literałów

Literal constants can be classified into:

- Integer *literał całkowity*
 - 25 // decimal int *system liczbowy dziesiętny*
 - 031 // octal integer *system liczbowy ósemkowy*
 - 0x19 // hexadecimal integer *system liczbowy szesnastkowy*
 - 25u // unsigned int
 - 25U // unsigned int
 - 25l // long
 - 25L // long
 - 25lu // unsigned long
 - 25ul // unsigned long

<https://en.cppreference.com/w/cpp/language/expressions>

<https://www.cplusplus.com/doc/tutorial/constants/>

Literały (stałe)

- floating-point *literał zmiennoprzecinkowy*
 - 3.14 // double
 - 3.14e2 // 3.14x10² double
 - 3.14e-2 // 3.14x10⁻² double
 - 3.14f // float
 - 3.14F // float
 - 3.14l // long double
 - 3.14L // long double
- characters *literał znakowy*
 - 'a' // litera w apostrofach

scientific notation *notacja naukowa*



Znaki specjalne

znak	znaczenie	nazwa angielska	przykład
\n	nowy wiersz	<i>new-line NL (LF)</i>	cout << '\n'; (wypisze znak nowej linii – enter)
\t	tabulacja pozioma	<i>horizontal tab HT</i>	
\v	tabulacja pionowa	<i>vertical tab VT</i>	
\b	cofnięcie kursora o jeden znak	<i>backspace BS</i>	
\r	powrót karetki	<i>carriage return CR</i>	
\f	nowa strona	<i>form feed FF</i>	
\a	dzwonek	<i>alert BEL</i>	
\\	backslash (lewy ukośnik)	<i>backslash \</i>	
\?	znak zapytania	<i>question mark ?</i>	
\'	apostrof	<i>single quote '</i>	
\"	cudzysłów	<i>double quote "</i>	
\ooo	liczba ósemkowa	<i>octal number ooo</i>	cout<<'\141'; (wypisze literę a)
\xhhh	liczba szesnastkowa	<i>hex number hhh</i>	cout<<'\x61'; (wypisze literę a)
\0	NUL, znak o kodzie 0	<i>NUL</i>	

Znaki specjalne

znak nowego wiersza

```
cout << "znaki specjalne\n\n";
cout << "cudzysłów: \\"";
cout << "Hello World! w cudzysłowach: ";
cout << "cout << \\"Hello World!\\""; \\"";
cout << "\\\"Hello World!\\\"\\n\\n";
cout << "tabulator: \\"\\t\\n";
cout << "użycie tabulatora: cout << \\"\\ttabulator\\\";\\n";
cout << "\\ttabulator\\n";
cout << "\\ttabulator\\n\\n";
cout << "a szesnastkowo: \\x61\\n";
cout << "wypisanie litery a: cout << \\\"x61\\\"; \\"";
cout << "\\x61\\n\\n";
```

program znakiSpecjalne

```
znaki specjalne
cudzysłów: \"
Hello World! w cudzysłowach: cout << \"Hello World!\"; \\"Hello World!\"
tabulator: \"
użycie tabulatora: cout << \"\\ttabulator\";
    tabulator
    tabulator

a szesnastkowo: \\x61
wypisanie litery a: cout << \"x61\"; \\" a
```

Literały (stałe)

- strings *literał łańcuchowy*
 - „informatyka” // napis w cudzysłowie
- Boolean *literał logiczny*
 - true // pierwsza z możliwych wartości zmiennej
 - false // druga z możliwych wartości zmiennej
- Pointer
 - nullptr // wartość null pointer

Stałe nazwane

stała nazwana

named constant

```
const int liczba = 25;  
const char nowaLinia = '\\n';
```

// użycie w kodzie stałych nazwanych

```
cout << liczba << nowaLinia;
```

Operator przypisania prostego

operator przypisania

assignment operator

int a = 1;

//inicjalizacja zmiennej a liczbą 1

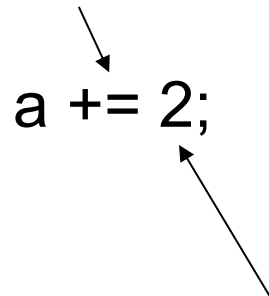
literal całkowity

literal constant integer

Operator przypisania złożonego

operator przypisania złożonego

compound assignment operator


a += 2;

// to samo co a=a+2;

literał całkowity

literal constant integer

wyrażenie	ekwiwalent
a += 2	a = a + 2
a /= 2	a = a / 2
a *= 2	a = a * 2
a -= 2	a = a - 2
a %= 2	a = a % 2

Operatory arytmetyczne dwuargumentowe

+	dodawanie	<i>addition</i>
-	odejmowanie	<i>subtraction</i>
*	mnożenie	<i>multiplication</i>
/	dzielenie	<i>division</i>
	dzielenie całkowite np.: $10/4=2$ (dzielimy 2 liczby całkowite typu int – iloraz jest zaokrąglany do liczby całkowitej)	
%	modulo – reszta z dzielenia liczb całkowitych	<i>modulo</i>

Operatory preinkrementacji i postinkrementacji

operator arytmetyczny
jednoargumentowy

++i

i++

$i = i + 1$

```
int i=1;  
int a=1;  
a=++i;  
cout<<a<<endl;  
cout<<i<<endl;
```

```
int i=1;  
int a=1;  
a=i++;  
cout<<a<<endl;  
cout<<i<<endl;
```

Operatory predekrementacji i postdekrementacji

operator arytmetyczne
jednoargumentowy

--i
i--

i = i - 1

```
int i=1;  
int a=1;  
a=--i;  
cout<<a<endl;  
cout<<i<endl;
```

```
int i=1;  
int a=1;  
a=i--;  
cout<<a<endl;  
cout<<i<endl;
```

Operator znaku plus i znaku minus

operatory arytmetyczne
jednoargumentowe

-i

zwraca wartość ujemną operandu

+i

zwraca wartość operandu

```
char c = 'a';  
cout << +c // 97
```

kod ASCII znaku 'a'

```
int d = 1;  
cout << -d // -1
```

```
unsigned int d = 1;  
cout << -d // 4294967295 //max unsigned int
```

If a negative integer value is converted to an unsigned type, the resulting value corresponds to its 2's complement bitwise representation (i.e., -1 becomes the largest value representable by the type, -2 the second largest, ...).

w Visual Studio błąd

Błąd C4146 jednoargumentowy operator minus zastosowany do typu unsigned, wynik nadal unsigned

https://en.cppreference.com/w/cpp/language/operator_arithmetic#Unary_arithmetic_operators

<https://www.cplusplus.com/doc/tutorial/typecasting/>

Priorytet operatorów

operator precedence

Level	Precedence group	Operator	Description	Grouping
1	Scope	::	scope qualifier	Left-to-right
2	Postfix (unary)	++ --	postfix increment / decrement	Left-to-right
		()	functional forms	
		[]	subscript	
		. ->	member access	
3	Prefix (unary)	++ --	prefix increment / decrement	Right-to-left
		~ !	bitwise NOT / logical NOT	
		+ -	unary prefix	
		& *	reference / dereference	
		new delete	allocation / deallocation	
		sizeof	parameter pack	
		(type)	C-style type-casting	
4	Pointer-to-member	* ->*	access pointer	Left-to-right
5	Arithmetic: scaling	* / %	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	+ -	addition, subtraction	Left-to-right
7	Bitwise shift	<< >>	shift left, shift right	Left-to-right
8	Relational	< > <= >=	comparison operators	Left-to-right
9	Equality	== !=	equality / inequality	Left-to-right
10	And	&	bitwise AND	Left-to-right
11	Exclusive or	^	bitwise XOR	Left-to-right
12	Inclusive or		bitwise OR	Left-to-right
13	Conjunction	&&	logical AND	Left-to-right
14	Disjunction		logical OR	Left-to-right
15	Assignment-level expressions	= *= /= %= += -=	assignment / compound assignment	Right-to-left
		=		
		>>= <<= &= ^=		
		=		
		?:	conditional operator	
16	Sequencing	,	comma separator	Left-to-right

<https://www.cplusplus.com/doc/tutorial/operators/>

Priorytet operatorów *operator precedence*

```
int liczba = 1 + 6 * 5; // różne priorytety  
cout << liczba; //?
```

```
int liczba = 10 * 6 / 5; // jednakowe priorytety, sprawdź grupowanie  
cout << liczba; //?
```

```
int liczba = 10 / 2 * 5; // jednakowe priorytety, sprawdź grupowanie  
cout << liczba; //?
```

```
int a = 1;  
int liczba = a++ + 10; // różne priorytety, uwaga na postinkrementację!  
cout << liczba; //?
```

```
int a = 1;  
int liczba = ++a + 10; // różne priorytety, uwaga na preinkrementację!  
cout << liczba; //?
```

Operatory bitowe

Bitwise operators

&	iloczyn bitowy AND	<i>Bitwise AND</i>
	suma bitowa OR	<i>Bitwise inclusive OR</i>
^	bitowa różnica symetryczna XOR	<i>Bitwise exclusive OR</i>
~	negacja bitowa NOT	<i>Unary complement (bit inversion)</i>
>>	operator przesunięcia bitowego w prawo	<i>Shift bits right</i>
<<	operator przesunięcia bitowego w lewo	<i>Shift bits left</i>

Operator bitowy AND

```
int a = 6;  
int b = 4;  
int c = a & b;  
cout << c; //?
```

6 -> 110

system binarny

4 -> 100

system dziesiętny

110
&100

100

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

Tablica prawdy dla AND

system dziesiętny

4

https://pl.wikipedia.org/wiki/Tablica_prawdy

Operator przesunięcia bitowego w lewo

```
int a = 6;  
int c = a << 1;  
cout << c; //?
```

6 -> 110

system binarny

system dziesiętny

<< 1

1100

system dziesiętny

12

przesuwamy bity o jedną pozycję w lewo, nowe bity z prawej strony uzupełniamy **zerami**

Operatory relacji (porównania)

==	równy	<i>Equal to</i>
!=	różny	<i>Not equal to</i>
>	większy	<i>Greater than</i>
<	mniejszy	<i>Less than</i>
>=	większy lub równy	<i>Greater than or equal to</i>
<=	mniejszy lub równy	<i>Less than or equal to</i>

Operatory relacji

```
int a = 6;  
int b = 4;  
bool c = a == b;  
cout << c; //?
```

sprawdź priorytet operatorów

```
int a = 6;  
int b = 4;  
bool c = a != b;  
cout << c; //?
```

sprawdź priorytet operatorów

Operator przecinkowy *Comma operator*

operator przecinkowy łączy
ze sobą kilka wyrażeń

```
int a, b;
```

```
a = 1;
```

```
b = (++a, a*20, a + 60);
```

```
cout << b;    // 62
```

```
return 0;
```

grupowanie od lewej do prawej

do b przypisywany jest
wynik ostatniego wyrażenia

Przestrzenie nazw Namespaces

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

```
namespace a {
    int liczba = 10;
}
```

```
namespace b {
    int liczba = 12;
}
```

```
int main()
{
    int liczba = 13;
    cout << a::liczba << endl; // 10
    cout << b::liczba << endl; // 12
    cout << liczba;           // 13

    return 0;
}
```

Przestrzenie nazw zapewniają metodę zapobiegania konfliktom nazw w dużych projektach.

Symbole zadeklarowane wewnątrz bloku przestrzeni nazw są umieszczane w nazwanym zakresie, co zapobiega ich pomyleniu z symbolami o identycznych nazwach w innych zakresach.

do zmiennej dostajemy się w programie za pomocą operatora zakresu

Słowa kluczowe

alignas, alignof, and, and_eq, asm, auto, bitand, bitor, bool, break, case, catch, char, char16_t, char32_t, class, compl, const, constexpr, const_cast, continue, decltype, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, noexcept, not, not_eq, nullptr, operator, or, or_eq, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_assert, static_cast, struct, switch, template, this, thread_local, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while, xor, xor_eq



**Tych słów nie można używać
jako nazw zmiennych!**

INSTRUKCJE STERUJĄCE

Wyrażenie expression

Sekwencja operatorów i operandów określająca obliczenie.

operatory arytmetyczne

operandy stałe

compound expression
wyrażenie złożone

$a + b - 5 > 10$

operator relacji

operandy zmienne

wynikiem wyrażenia najczęściej jest jakaś wartość. W powyższym przykładzie jest nią wartość typu bool (true lub false).

Instrukcja warunkowa if Conditional statement

Wyrażenie może przyjąć jedną z dwóch wartości logicznych:

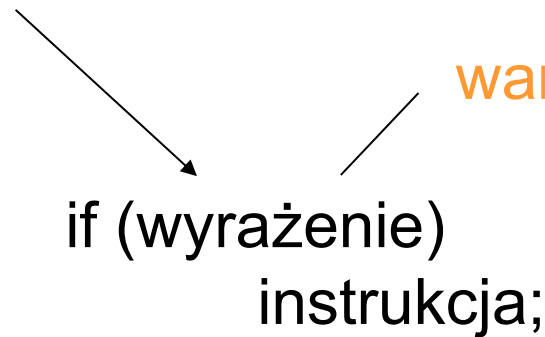
true - wyrażenie jest prawdziwe

false – wyrażenie jest fałszywe

W C++ przyjmuje się, że:

0 – false

liczba różna od zera – true



```
if (wyrażenie)
    instrukcja;
```

warunek Jeśli warunek jest prawdziwy (ma wartość true) wykonywana jest instrukcja. W przeciwnym wypadku instrukcja nie jest wykonywana.

Przykłady:

(a==2)

true dla zmiennej a równej 2

false dla zmiennej a równej 3

(a)

true dla zmiennej a różnej od 0 (np.: 1,2,4, 10 itp.)

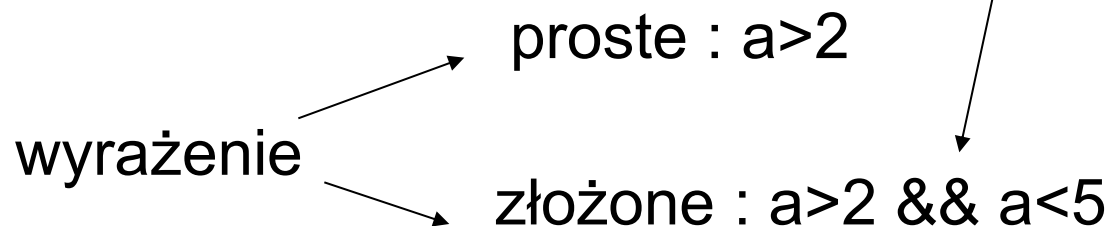
false dla zmiennej a równej 0

a+2*b>c

true jeśli wartość wyrażenia a+2*b jest większa od c

Instrukcja warunkowa if *Conditional statement*

koniunkcja dwóch warunków



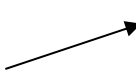
Logical operators

Operatory logiczne

	alternatywa
&&	koniunkcja
!	negacja

Instrukcja warunkowa if Conditional statement

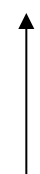
```
if (wyrażenie)  
    instrukcja1;
```



Pojedyncza instrukcja
nie musi mieć nawiasów
klamrowych

```
if (wyrażenie)  
    instrukcja1;  
else  
    instrukcja2;
```

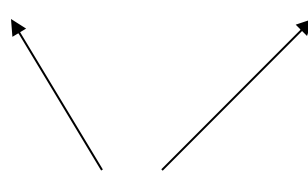
```
if (wyrażenie) {  
    instrukcja1;  
    instrukcja2;  
    instrukcja3;  
}
```



blok instrukcji
(musi być ujęty
w nawiasy klamrowe)

```
if (wyrażenie 1)  
    instrukcja1;  
else if (wyrażenie 2)  
    instrukcja2;  
else if (wyrażenie 3)  
    instrukcja3;
```

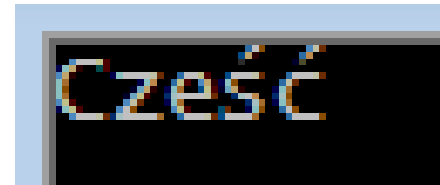
```
if (wyrażenie 1)  
    instrukcja1;  
else if (wyrażenie 2)  
    instrukcja2;  
else if (wyrażenie 3)  
    instrukcja3;  
else  
    instrukcja4;
```



wybór wielowariantowy

Zagnieżdżenie warunków

```
string imie = "Adam";  
string nazwisko = "Kowalski";  
  
if (imie == "Adam") {  
    if (nazwisko == "Kowalski") {  
        cout << "Cześć";  
    }  
}
```



Warunki mogą być zagnieżdżone (jeden w drugim)

Operator warunkowy

Conditional operator, ternary operator

wyrażenie1 ? wyrażenie2 : wyrażenie3

```
int a = 1;
```

```
int b = 2;
```

```
if (a < 10) {
```

```
    b = 12;
```

```
}
```

```
else {
```

```
    b = 13;
```

```
}
```

```
cout << b; // 12
```



warunek

```
b = (a < 10) ? 12 : 13;
```

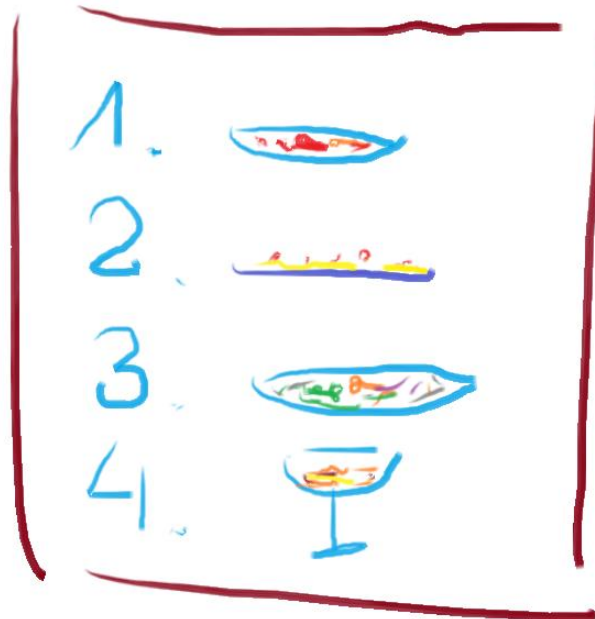
```
cout << b; // 12
```

Instrukcja wyboru

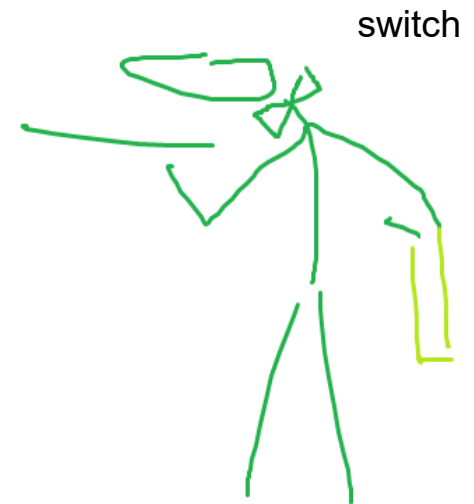
Selection statement

```
switch(wyrażenie)
{
    case wartość1:
        instrukcja1;
        break;
    case wartość2:
        instrukcja2;
        break;
    default:
        instrukcja3;
}
```

switch //przełącznik



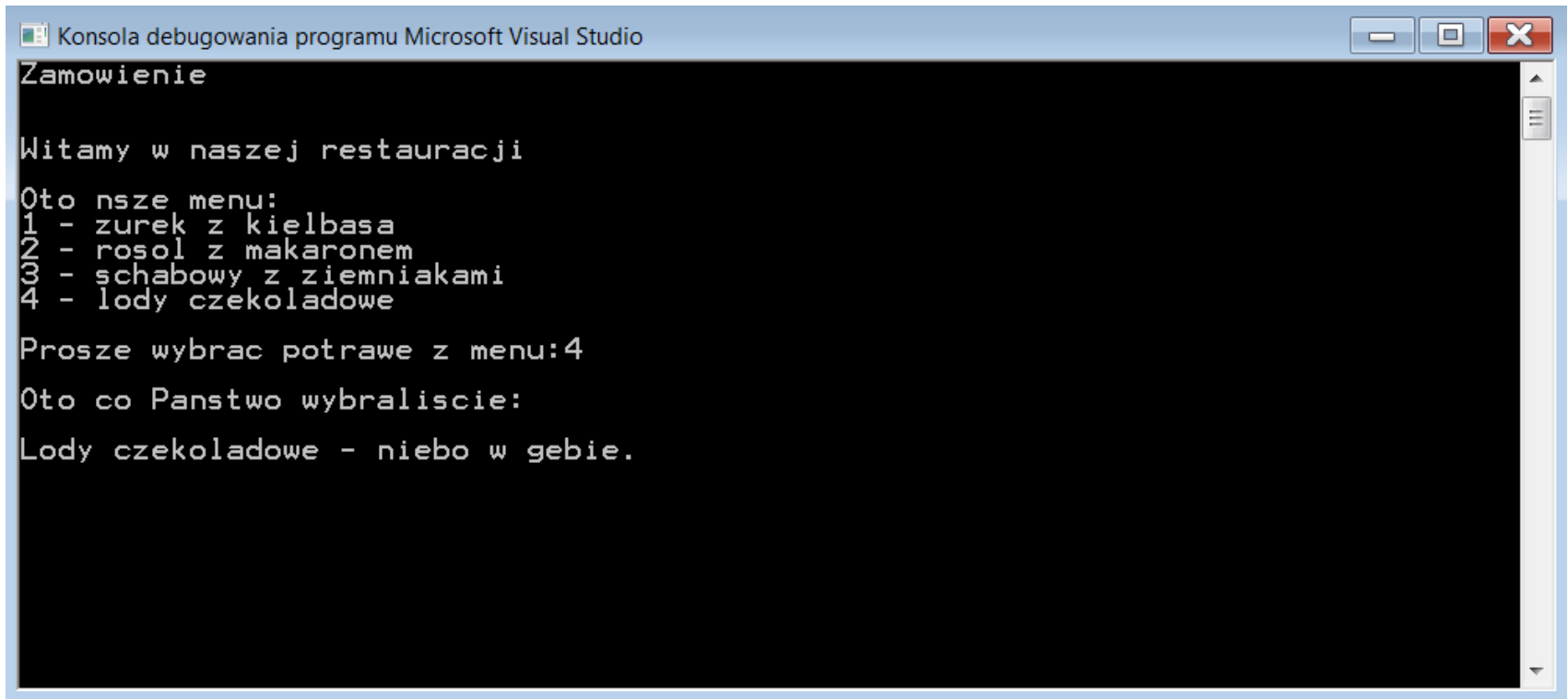
Menu



Kelner

zadanie

zamowienie



```
Konsola debugowania programu Microsoft Visual Studio

Zamowienie

Witamy w naszej restauracji

Oto nsze menu:
1 - zurek z kielbasa
2 - rosol z makaronem
3 - schabowy z ziemniakami
4 - lody czekoladowe

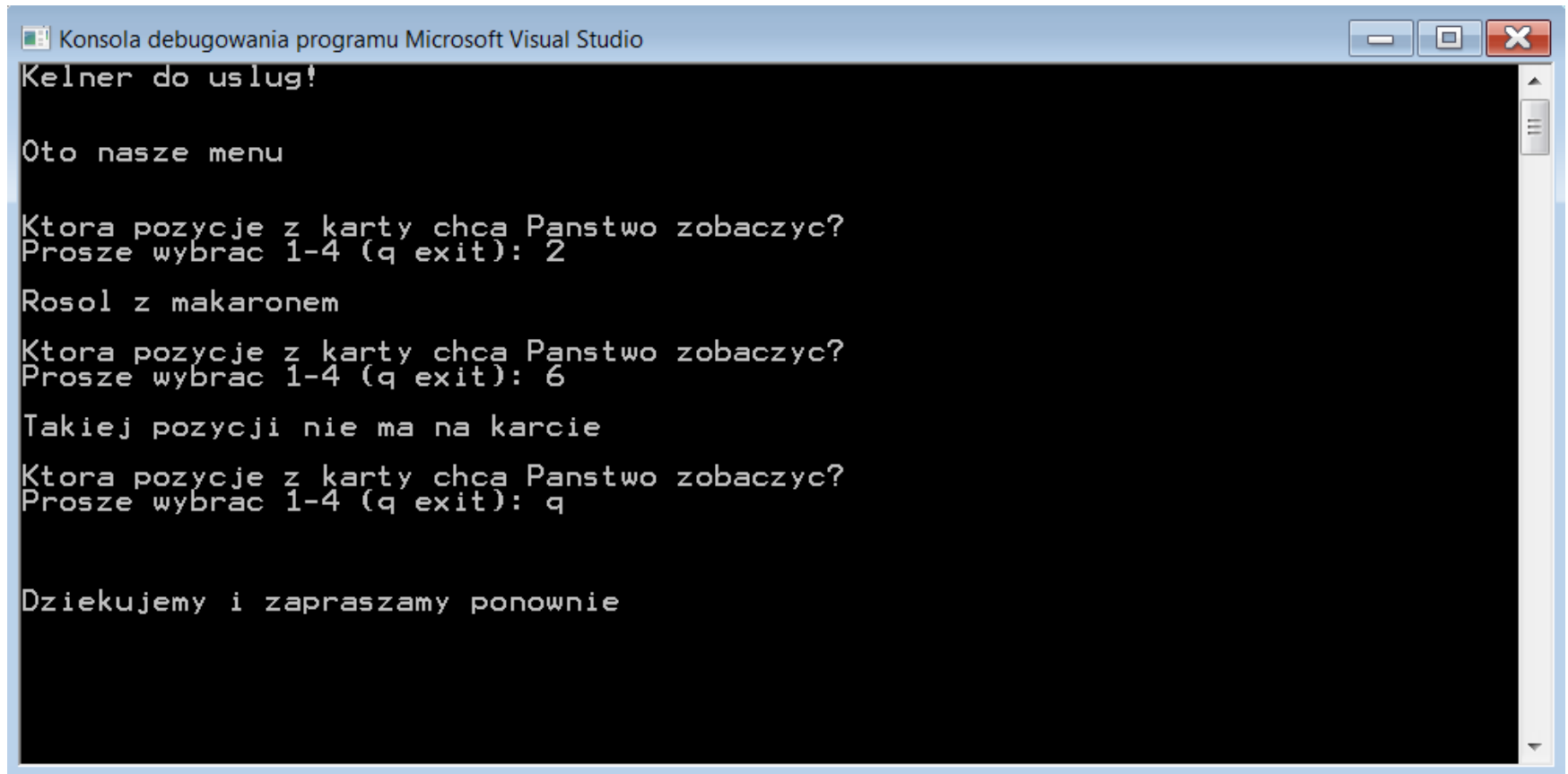
Prosze wybrac potrawe z menu:4

Oto co Panstwo wybraliscie:

Lody czekoladowe - niebo w gebie.
```

zadanie

menu



```
Konsola debugowania programu Microsoft Visual Studio

Kelner do uslug!

Oto nasze menu

Ktora pozycje z karty chca Panstwo zobaczyc?
Prosze wybrac 1-4 (q exit): 2

Rosol z makaronem

Ktora pozycje z karty chca Panstwo zobaczyc?
Prosze wybrac 1-4 (q exit): 6

Takiej pozycji nie ma na karcie

Ktora pozycje z karty chca Panstwo zobaczyc?
Prosze wybrac 1-4 (q exit): q

Dziekujemy i zapraszamy ponownie
```


Pętla for

Przed pierwszym uruchomieniem pętli wykonywana jest ta instrukcja

Gdy jest tylko jedna instrukcja nawiasów może nie być

Warunek pętli – wyrażenie sprawdzane przed każdorazowym obiegiem pętli. Jeśli jest prawdziwe wykonywana jest treść pętli.

```
for (int i=1; i<=10; i++)
```

```
{
```

```
    cout<<"Licznik pętli: ";
```

```
    cout << i << endl;
```

```
}
```

Instrukcja wykonywana każdorazowo na zakończenie obiegu pętli

Treść pętli
(instrukcje ujęte w nawiasach klamrowych)

Pętla for

Deklaracja i zainicjowanie licznika pętli

Warunek

Krok pętli

```
for (int i=1; i<=10; i++)
```

```
cout << i;
```

Ciało pętli

Gdy jest tylko
jedna instrukcja
nawiasów klamrowych
może nie być

Pętla for

Kolejność wykonywania instrukcji

```
for (int 1i=1; i2<=10; i4++)  
{  
    cout << i; 3  
}
```

pierwszy obieg pętli 1234

drugi obieg pętli 234

trzeci obieg pętli 234

kolejne obiegi pętli 234

Pętla while

dopóki jest spełniony warunek

Gdy jest tylko
jedna instrukcja
nawiasów
może nie być

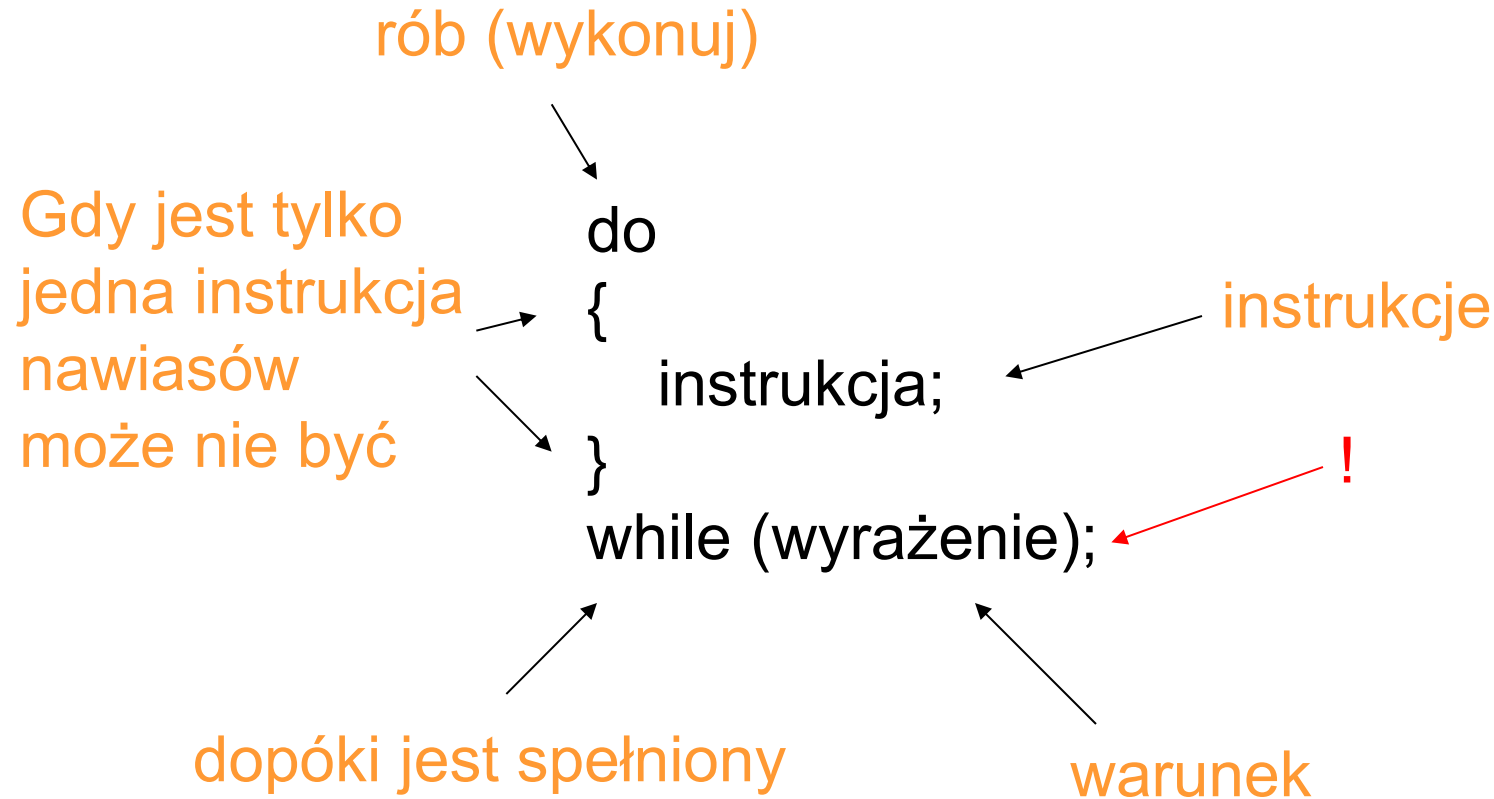
```
while (wyrażenie)
{
    instrukcja;
}
```

wykonuj instrukcje

wyrażenie może przyjmować wartość logiczną:

- prawda
- fałsz

Pętla do while



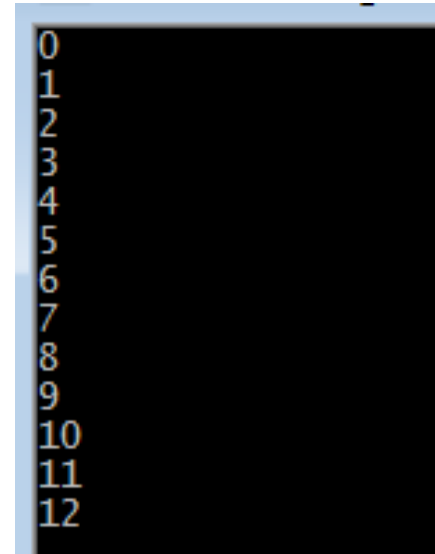
wyrażenie może przyjmować wartość logiczną:

- prawda
- fałsz

Instrukcja break

Pętla nieskończona while

```
int licznik = 0;
while (true) {
    cout << licznik << endl;
    if (licznik == 12)
        break;
    licznik++;
}
```



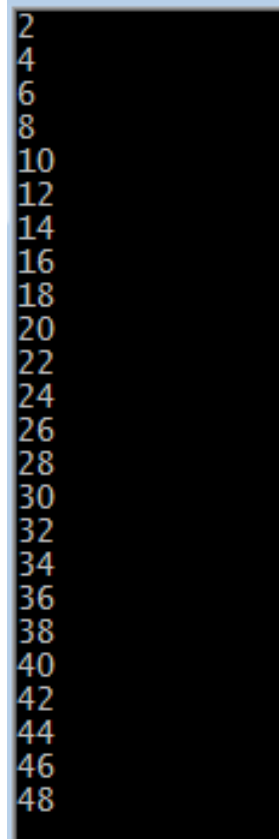
A terminal window with a black background and yellow text. It displays the numbers 0 through 12, each on a new line, arranged vertically. The text is: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12.

break przerywa wykonywanie pętli

Instrukcja continue

dla liczb nieparzystych warunek będzie równy 1
czyli pominięte będzie cout.

```
for (int i = 1; i < 50; i++)  
{  
    if (i % 2)  
        continue;  
    cout << i << endl;  
}
```

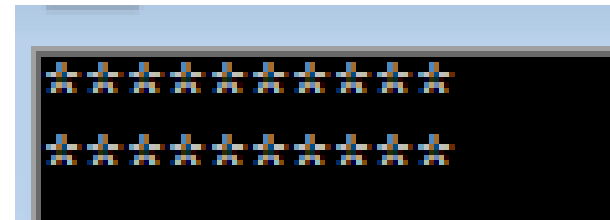


```
2  
4  
6  
8  
10  
12  
14  
16  
18  
20  
22  
24  
26  
28  
30  
32  
34  
36  
38  
40  
42  
44  
46  
48
```

continue pomija wykonywanie pętli od miejsca w którym stoi do końca pętli

Zagnieżdżenie pętli

```
for (int i = 0; i < 2; i++)  
{  
    for (int j = 0; j < 10; j++)  
    {  
        cout << '*';  
    }  
    cout << endl;  
}
```



Pętle mogą być zagnieżdżone (jedna w drugiej)

Struktury

definicja struktury

osoba jest nazwą nowego typu

struct Osoba {

nawias klamrowy
otwierający

słowo kluczowe

string imie;
string nazwisko;
short wiek;

pola struktury

nawias klamrowy
zamykający

};

pola danych różnego typu

!

Struktury

stworzenie egzemplarza
struktury typu osoba
o nazwie pierwsza

Osoba pierwsza;
Osoba druga;

inicjalizacja struktury

pierwsza.imie=„Ola”;
pierwsza.nazwisko=„Kowalska”;
pierwsza.wiek=13;

druga.imie=„Andrzej”;
druga.nazwisko=„Milka”;
druga.wiek=14;

Struktury

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

int main()
{
    struct osoba {
        string imie;
        string nazwisko;
        int wiek;
    };

    osoba Adam;

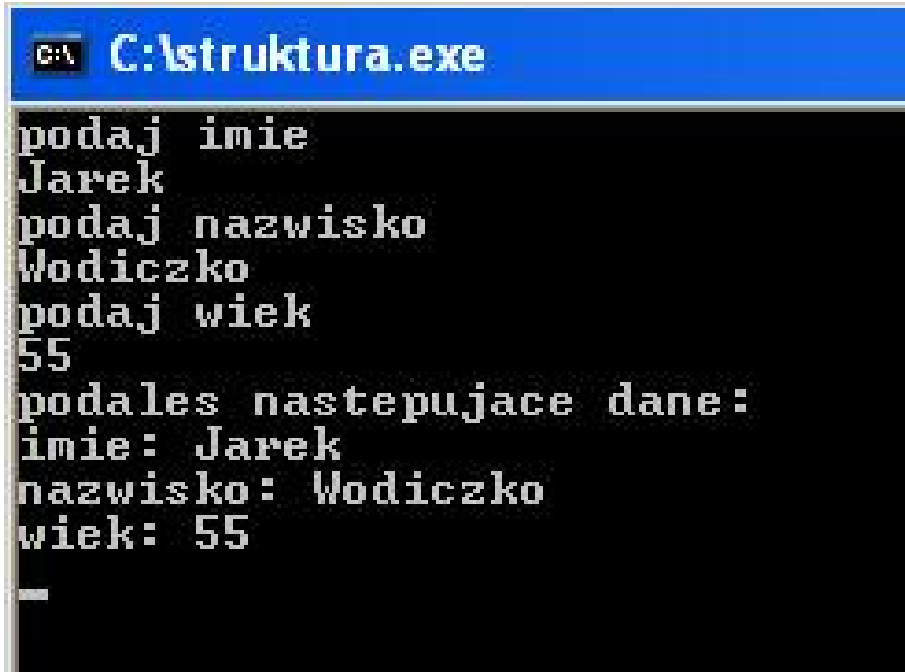
    Adam.imie = "Adam";
    Adam.nazwisko = "Kowalski";
    Adam.wiek = 24;

    //lub
    //osoba Adam = {"Adam", "Kowalski", 24}; //C++11
    //osoba Adam {"Adam", "Kowalski", 24};    //C++11

    cout << Adam.imie << " " << Adam.nazwisko << " " << Adam.wiek;

    return 0;
}
```

Pobierz imię, nazwisko i wiek użytkownika oraz wypisz je na konsoli korzystając ze struktury.



```
C:\struktura.exe
podaj imie
Jarek
podaj nazwisko
Wodiczko
podaj wiek
55
podales nastepujace dane:
imie: Jarek
nazwisko: Wodiczko
wiek: 55
_
```

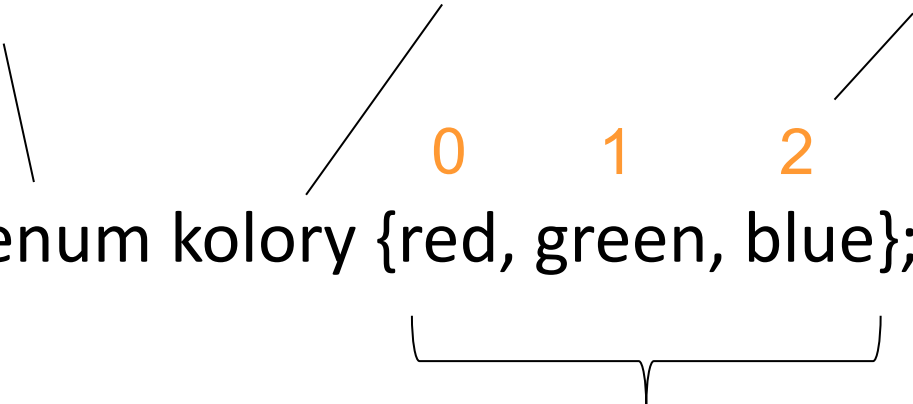
Typ wyliczeniowy enum

słowo kluczowe enum

nazwa typu

enumeratory są
numerowane

enum kolory {red, green, blue};



0 1 2

lista identyfikatorów (enumeratorów)

Typ wyliczeniowy enum Enumerated types

```
//           0      1      2
enum kolory {red, green, blue};

kolory ulubiony = red;

cout << ulubiony; // 0
```

```
//           0      1      2
enum kolory {red, green, blue};

kolory ulubiony = red;

switch (ulubiony)
{
case red:
    cout << "Czerwony"; // Czerwony
    break;
case green:
    cout << "Zielony";
    break;
case blue:
    cout << "Niebieski";
    break;
default:
    break;
}
```

Typ wyliczeniowy enum Enumerated types

```
enum imie {  
    Ania = 8,  
    Jola = 10,  
    Wiktoria,  
    Beata  
};  
  
imie zona = Beata;  
  
cout << zona;  //?  
cout << Beata;  //?
```

Typ wyliczeniowy enum Enumerated types

```
enum imie {  
    Ania = 4,  
    Jola = 8,  
    Wiktor,  
    Beata  
};
```

nie można duplikować identyfikatorów

```
enum kobiety {  
    Ania = 4,  
    Jola = 8,  
    Wiktor,  
    Beata  
};
```

✗	C2365	"main::Ania": zmiana definicji; definicja poprzednia była "moduł wyliczający"
✗	C2365	"main::Jola": zmiana definicji; definicja poprzednia była "moduł wyliczający"
✗	C2365	"main::Wiktor": zmiana definicji; definicja poprzednia była "moduł wyliczający"
✗	C2365	"main::Beata": zmiana definicji; definicja poprzednia była "moduł wyliczający"

```
imie zona = Beata;  
cout << Beata;
```


Enum class

```
enum class Imie {  
    Ania,  
    Jola,  
    Wiktoria,  
    Beata  
};
```

Visual Studio preferuje ten typ enum

```
enum class Kobiety {  
    Ania,  
    Jola,  
    Wiktoria,  
    Beata  
};
```

identyfikator z operatorem zakresu

```
Imie zona = Imie::Beata;  
switch (zona) {  
    case Imie::Beata:  
        cout << "Cześć Beata\n"; // Cześć Beata  
}
```

```
Kobiety kobieta = Kobiety::Beata;
```

```
if (kobieta == Kobiety::Beata) {  
    cout << "Cześć Beata"; // Cześć Beata  
}
```

Tablica jednowymiarowa (statyczna)

fix array
compile-time array

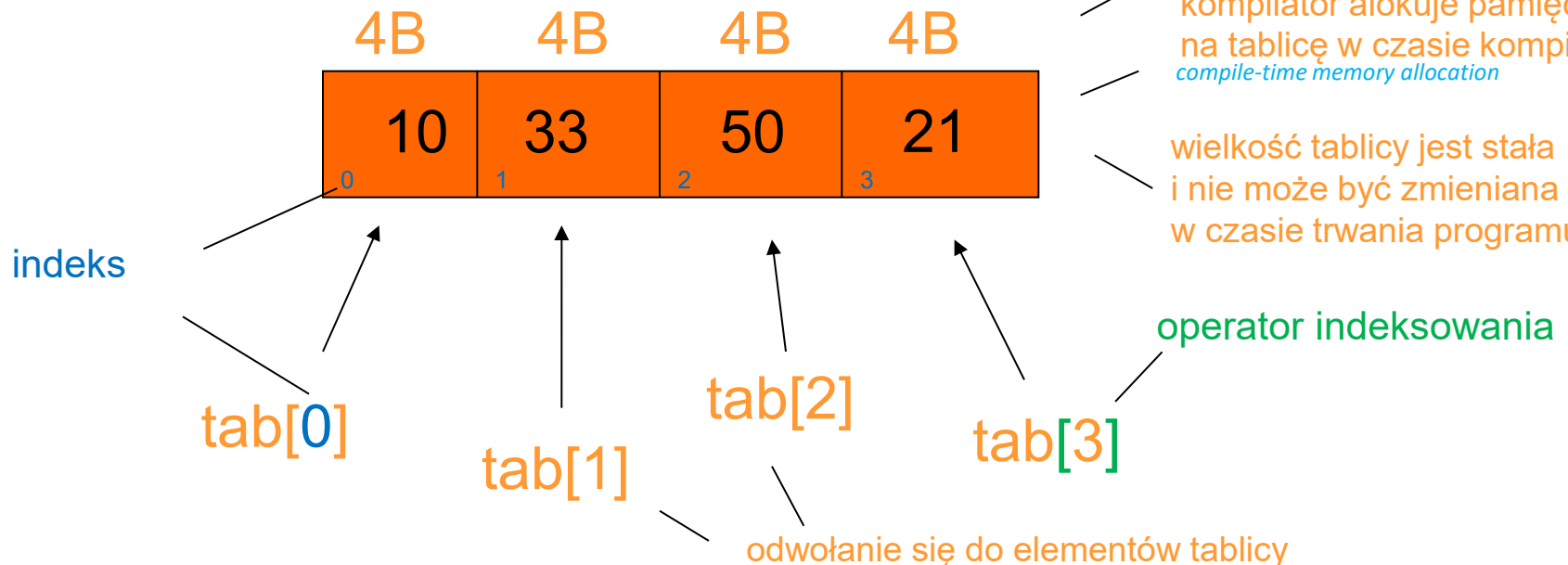
```
int tab[4];    //deklaracja czteroelementowej tablicy typu int
tab[0] = 10;   // inicjalizacja pierwszego elementu tablicy
tab[1] = 33;   // inicjalizacja drugiego elementu tablicy
tab[2] = 50;   // inicjalizacja trzeciego elementu tablicy
tab[3] = 21;   // inicjalizacja czwartego elementu tablicy
```

```
int tab[4] = {10,33,50,21}; //deklaracja oraz inicjalizacja tablicy
int tab[4] {10,33,50,21};   //alternatywnie
int tab[] = {10,33,50,21};  //alternatywnie
int tab[] {10,33,50,21};    //alternatywnie
```

tablica zajmuje ciągły
obszar w pamięci
w obszarze zwanym
stos *stack*

kompilator alokuje pamięć
na tablicę w czasie kompilacji
compile-time memory allocation

wielkość tablicy jest stała
i nie może być zmieniana
w czasie trwania programu



Tablica jednowymiarowa (statyczna)

fix array
compile-time array

nazwa tablicy jest jednocześnie wskaźnikiem na jej pierwszy element

inicjalizowanie tablicy zerami

```
int tab[4]{}; //C++11
```

```
cout << "tablica została wypełniona zerami:" << endl;
```

```
for (int i = 0; i < 4; i++)  
{  
    cout << tab[i] << " ";  
}
```

```
cout << "\nwypełniamy tablicę liczbami:" << endl;
```

```
for (int i = 0; i < 4; i++)  
{  
    tab[i] = i + 1;  
    cout << tab[i] << " ";  
}
```

```
cout << "\nelement tablicy o indeksie zero:\n";
```

```
cout << *tab << endl;
```

```
cout << "\nelement tablicy o indeksie trzy:\n";
```

```
cout << *(tab + 3) << endl;
```

Konsola debugowania programu Microsoft Visual Studio

```
tablica została wypełniona zerami:  
0 0 0 0  
wypełniamy tablicę liczbami:  
1 2 3 4  
element tablicy o indeksie zero:  
1  
  
element tablicy o indeksie trzy:  
4
```

Tablica jednowymiarowa (statyczna)

fix array
compile-time array

```
int a = 4;  
int tab[a];
```

błąd - wielkość tablicy musi być stała w czasie kompilacji
(nie można użyć zmiennej jako wielkości tablicy statycznej)

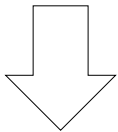
 (zmienna lokalna) `int a`

[Wyszukaj w trybie online](#)

wrażenie musi mieć stałą wartość

wartości elementu zmienna "a" (zadeklarowane w wierszu 14) nie można użyć jako stałej

[Wyszukaj w trybie online](#)



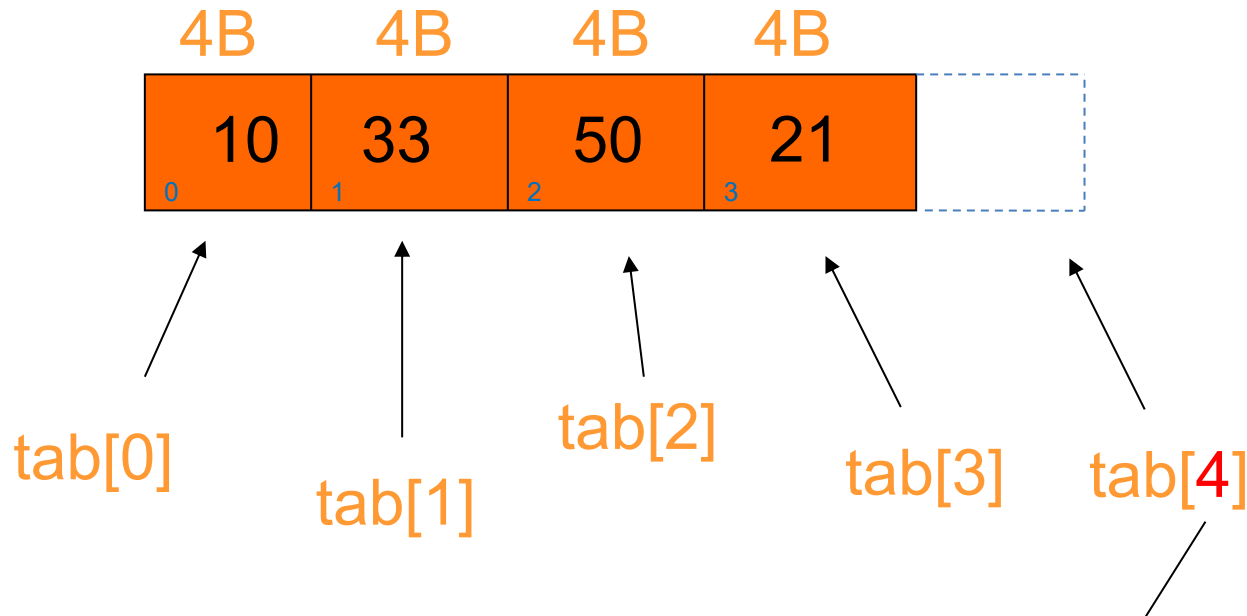
OK - wielkość tablicy jest stała w czasie kompilacji

```
const int a = 4;  
int tab[a];
```

Tablica jednowymiarowa (statyczna)

fix array
compile-time array

```
int tab[4] = {10, 33, 50, 21}; // deklaracja oraz inicjalizacja tablicy
```



Błąd – wyjście poza zakres tablicy

Tablica jednowymiarowa (statyczna)

```
int tab[4];    //deklaracja czteroelementowej tablicy typu int
tab[0] = 10;  // inicjalizacja pierwszego elementu tablicy
tab[1] = 33;  // inicjalizacja drugiego elementu tablicy
tab[2] = 50;  // inicjalizacja trzeciego elementu tablicy
tab[3] = 21;  // inicjalizacja czwartego elementu tablicy
tab[4] = 33;
```

(int)33

[Wyszukaj w trybie online](#)

C6386: Przepelnienie buforu podczas zapisu w elemencie „tab”: rozmiar obszaru do zapisu to „16”, ale zapisana liczba bajtów może wynieść: „20”.

Błąd – wyjście poza zakres tablicy

po skompilowaniu

Microsoft Visual C++ Runtime Library



Debug Error!

Program:

C:\Users\Marek\source\repos\tablice\Debug\tablice.exe

Module: C:\Users\Marek\source\repos\tablice\Debug\tablice.exe

File:

Run-Time Check Failure #2 - Stack around the variable 'tab' was corrupted.

(Press Retry to debug the application)

Przerwij

Ponów próbę

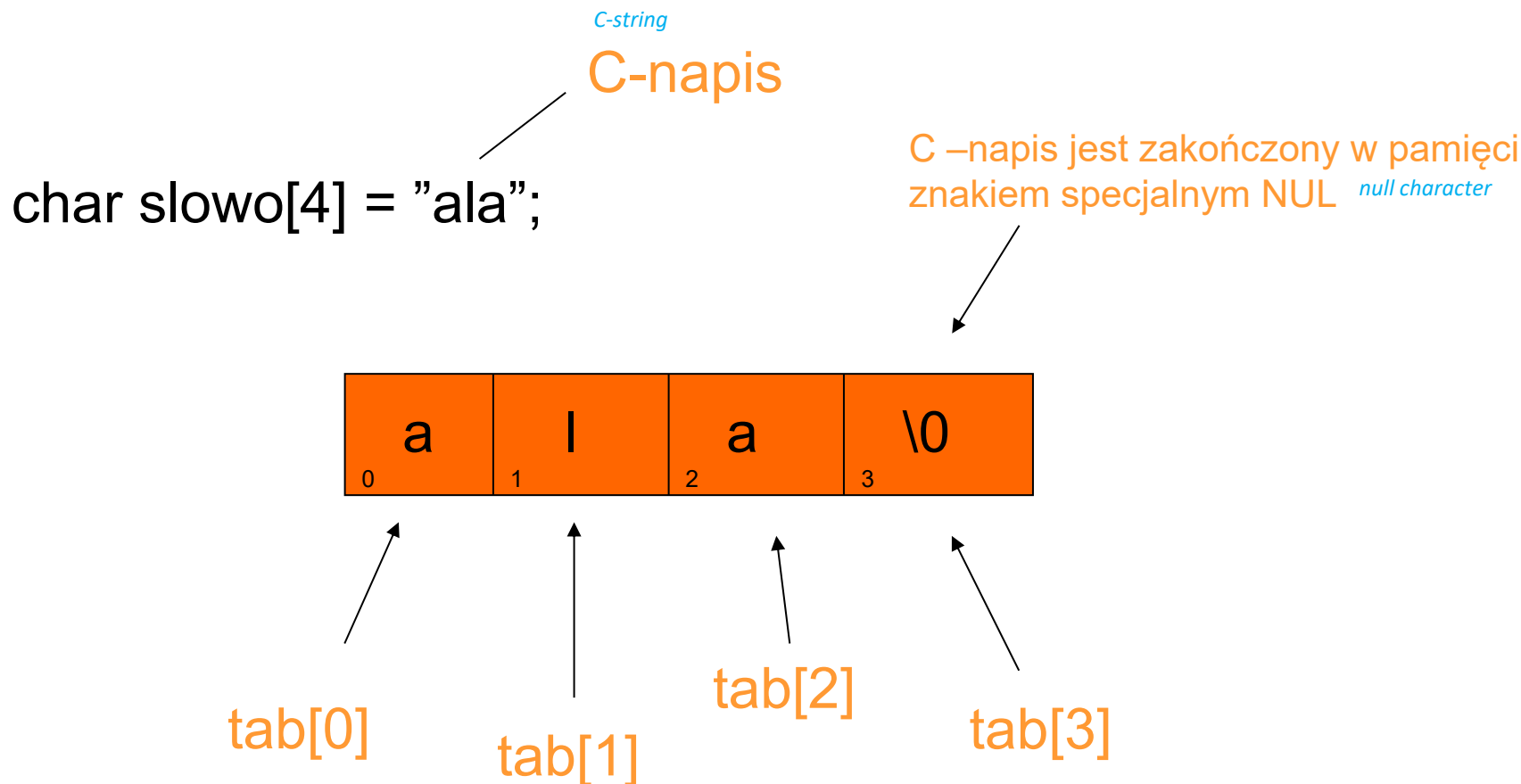
Ignoruj

Tablica jednowymiarowa (statyczna)

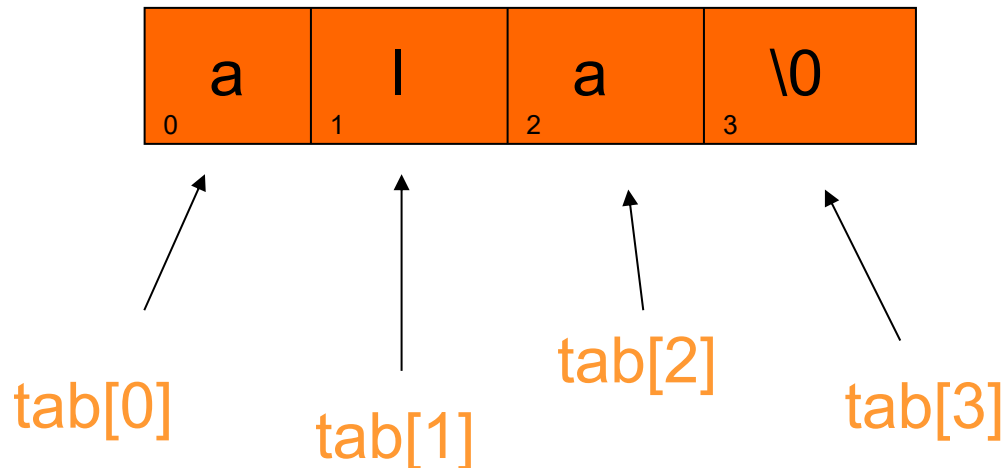
zadanie

Inicjalizuj tablicę czteroelementową cyframi od 0 do 3 za pomocą pętli oraz wypisz ją na konsoli również za pomocą pętli

Tablica jednowymiarowa (statyczna)



Tablica jednowymiarowa (statyczna)



```
char slowo[4] = {"ala"};  
char slowo[ ] = {"ala"};  
char slowo[ ] {"ala"};  
char slowo[ ] = {'a','l','a','\0'};  
const char* slowo = "ala";  
char slowo[ ]("ala");  
char slowo[4]("ala");
```

```
cout << slowo;
```

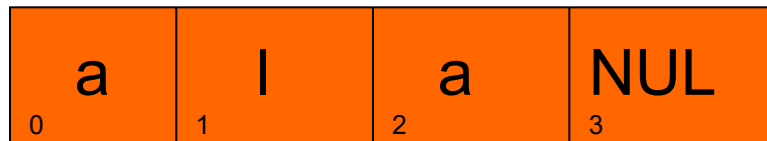
```
//alternatywnie  
//alternatywnie  
//alternatywnie  
//alternatywnie  
//alternatywnie  
//alternatywnie  
//alternatywnie
```

```
//wypisanie słowa na konsoli
```

Tablica jednowymiarowa (statyczna)

```
char slowo[]={"ala"};
```

Bajt zerowy



```
char slowo[]={'a','l','a'};
```



Tablica jednowymiarowa (dynamiczna)

alokujemy tablicę
w trakcie działania
programu



```
// zmienna na rozmiar tablicy
int size = 0;
// pobieramy rozmiar tablicy
cout << "Podaj rozmiar tablicy:" << endl;
cin >> size;
// alokujemy tablicę o wielkości size na stercie
int* tab = new int[size];

// nie da się tego zrobić na stosie
// wielkość tablicy musi być znana w czasie kompilacji
// a size jest zmienną
// int tab[size]; <----- błąd

// wypełniamy tablicę liczbami
for (int i = 0; i < size; i++)
{
    tab[i] = i;
    // wyświetlamy zawartość tablicy
    cout << tab[i] << " " << *(tab + i) << endl;
}

// usuwamy tablicę
delete [] tab;
tab = nullptr;
```

zadanie

Kolej

Wypisz słowo ***kolej*** na konsoli za pomocą pętli tak, aby kolejne litery były wypisane w nowych wierszach.

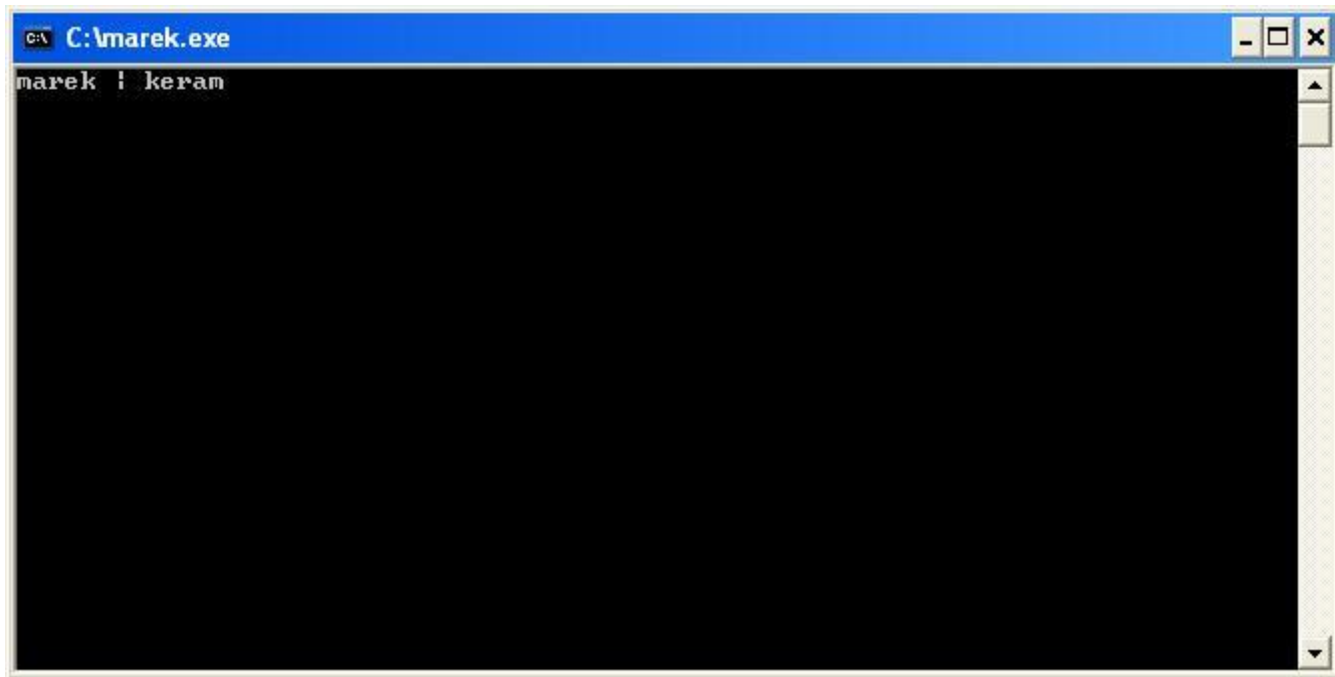


```
C:\K:\kolej.exe
k
o
l
e
j
```

zadanie

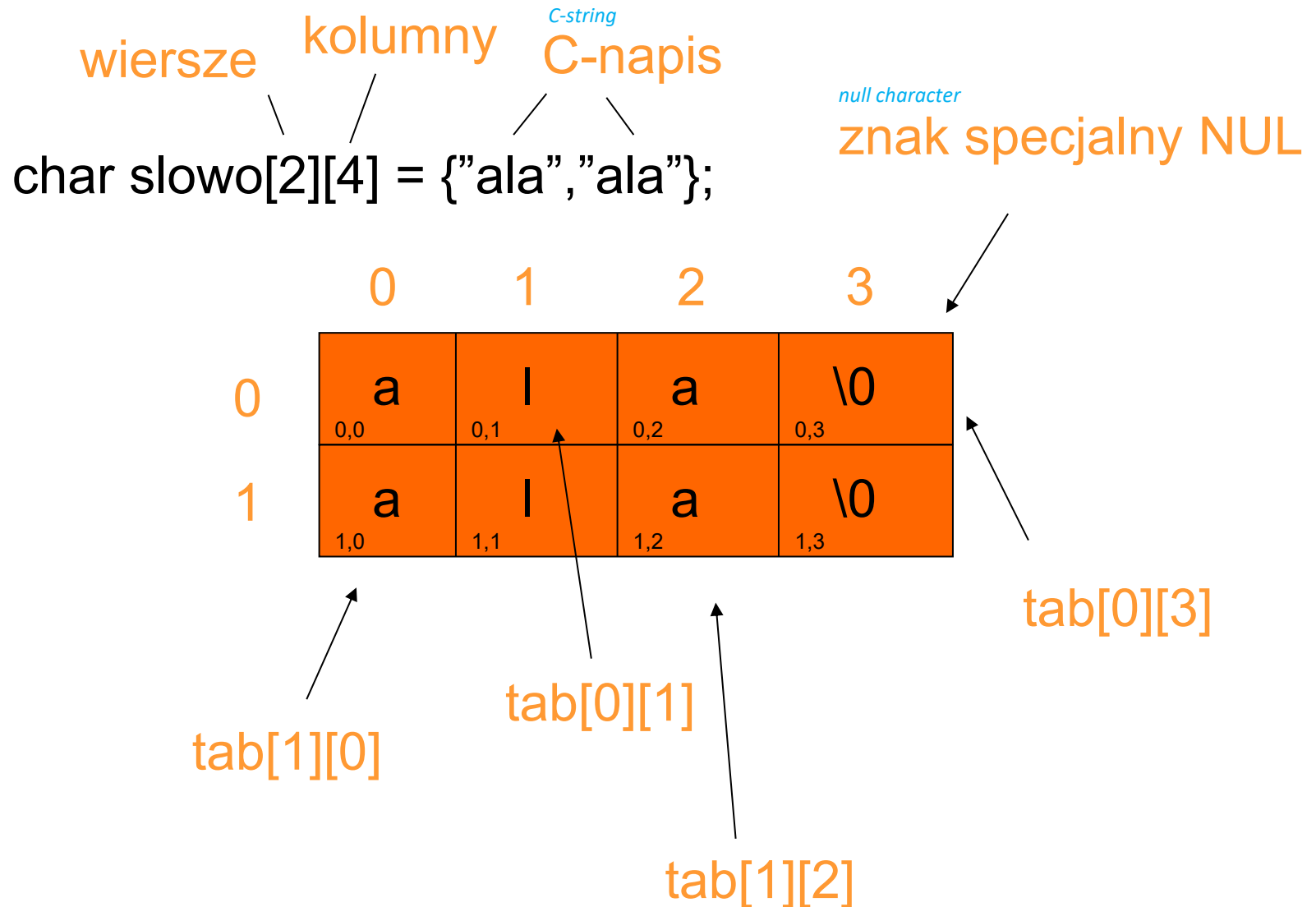
Lustro

Wypisz swoje imię na konsoli oraz jego „odbicie w lustrze”:



```
C:\>marek.exe
marek | keram
```

Tablica dwuwymiarowa (statyczna)

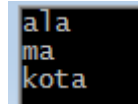


Tablica dwuwymiarowa (statyczna)

```
char tab[3][5] = {"ala ", "ma  ", "kota"};
```

```
/*  
int tab[2][4] = {           // zapis czytelniejszy  
    {"ala "},  
    {"ma  "},  
    {"kota"}  
};  
*/
```

```
for (int i = 0; i < 3; i++) {  
    for (int j = 0; j < 4; j++){  
        cout << tab[i][j];  
    }  
    cout << endl;  
}
```



```
ala  
ma  
kota
```

Tablica dwuwymiarowa (statyczna)

```
int tab[2][4] = {{1,2,3,4},{5,6,7,8}};
```

```
/*
```

```
int tab[2][4] = {                                // zapis czytelniejszy
```

```
    {1,2,3,4},
```

```
    {5,6,7,8}
```

```
};
```

```
*/
```

```
for (int i = 0; i < 2; i++) {
```

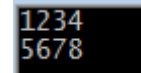
```
    for (int j = 0; j < 4; j++){
```

```
        cout << tab[i][j];
```

```
    }
```

```
    cout << endl;
```

```
}
```



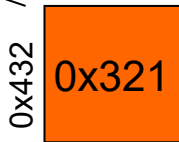
```
1234  
5678
```


Tablica dwuwymiarowa (dynamiczna)

wskaźnik do wskaźnika

tablica wskaźników

tab



```
int** tab = new int* [4];
```

```
tab[0] = new int[3];
```

```
tab[1] = new int[3];
```

```
tab[2] = new int[3];
```

```
tab[3] = new int[3];
```

wskaźnik na nową
tablicę typu int

tab[0][0]

tab[0][1]

tab[0][2]

tab[3][0]

tab[3][1]

tab[3][2]

```
tab[0][0] = 10;
```

```
delete [] tab[0];
```

```
delete [] tab[1];
```

```
delete [] tab[2];
```

```
delete [] tab[3];
```

```
delete [] tab;
```

usunięcie całej tablicy dynamicznej tab

0x121 0	0x125 1	0x129 2	0x12D 3
10 0	12 0	11 0	0 0
20 1	22 1	45 1	18 1
30 2	34 2	67 2	19 2

STRING

klasa stworzona w celu operowania napisami
(ciągami tekstowymi)

Klasa string



C++ : Reference : Strings library : string

Search:

login:
remember me ☐

C++

Information

Documentation

Reference

Articles

Sourcecode

Forums

Reference

C Library

IOstream Library

Strings library

STL Containers

STL Algorithms

Miscellaneous

Strings library

char_traits

classes:

· string

global functions:

· getline

· operator+

· operator<<

· operator>>

· comparison operators

· swap

string

string::string

member constants:

· string::npos

member functions:

· string::append

· string::assign

· string::at

· string::begin

· string::capacity

· string::clear

· string::compare

· string::copy

· string::c_str

· string::data

· string::empty

· string::end

· string::erase

· string::find

· string::find_first_n...

· string::find_first_of

· string::find_last_no...

· string::find_last_of

C# and ASP.NET add-in

ReSharper. Productivity add-in to VS.NET for C# and ASP developers www.jetbrains.com/resharper

Ads by Google

string

class

<string>

String class

String objects are a special type of container, specifically designed to operate with sequences of characters.

Unlike traditional c-strings, which are mere sequences of characters in a memory array, C++ string objects belong to a class with many built-in features to operate with strings in a more intuitive way and with some additional useful features common to C++ containers.

The string class is an instantiation of the basic_string class template, defined in <string> as:

```
typedef basic_string<char> string;
```

Member functions

(constructor)	Construct string object (constructor member)
operator=	String assignment (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)

Capacity:

size	Return length of string (public member function)
length	Return length of string (public member function)
max_size	Return maximum size of string (public member function)
resize	Resize string (public member function)
capacity	Return size of allocated storage (public member function)
reserve	Request a change in capacity (public member function)

<http://www.cplusplus.com/reference/string/string/>

Klasa string

//Dołączamy plik nagłówkowy

```
#include <string>
```

//deklaracja i definicja obiektu klasy string
string imie;

//inicjalizacja

```
imie = "Malgosia";
```

łańcuch znaków (c-napis)



//deklaracja, definicja i inizjalizacja

```
string imie = "Malgosia";
```

```
string imie ("Malgosia");
```

Klasa string

Przykład:

```
string imie;  
cout<<"podaj imie"<<endl;  
cin>>imie;  
cout<<"Masz na imię " <<imie;
```

Klasa string

przydatne metody klasy string

```
imie.size();    // wielkość obiektu klasy string
```

np.:

```
string imie;  
cout<<"podaj imie"<<endl;  
cin>>imie;  
cout<<"Twoje imię ma "<<imie.size()<<"liter";
```

Klasa string

zmienna imie może być traktowana jak tablica znaków:

imie[0] //pierwszy znak tablicy imie

lub

imie.at(0); //pierwszy znak tablicy imie

np.:

```
string imie;
```

```
cout<<"podaj imie"<<endl;
```

```
cin>>imie;
```

```
cout<<"Twoje imię zaczyna się na literę „<<imie[0];
```

FUNKCJE

Funkcja jest podprogramem.

Funkcje

Deklaracja funkcji

nazwa funkcji

void komunikat();

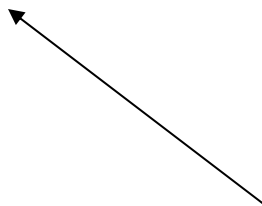
typ zwracanej wartości
(funkcja komunikat()
nie zwraca żadnej wartości)

argumenty (parametry) formalne

Funkcje

Definicja funkcji

```
void komunikat()  
{  
    cout<<„Uwaga WIRUS!!!”;  
}
```



ciało funkcji

Funkcje

```
#include<iostream>

using namespace std;

//deklaracja funkcji komunikat
void komunikat();

int main() {

    //wyświetlamy komunikat
    komunikat();

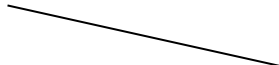
}

//definicja funkcji komunikat
void komunikat() {

    cout << "Uwaga WIRUS!!!" << endl;

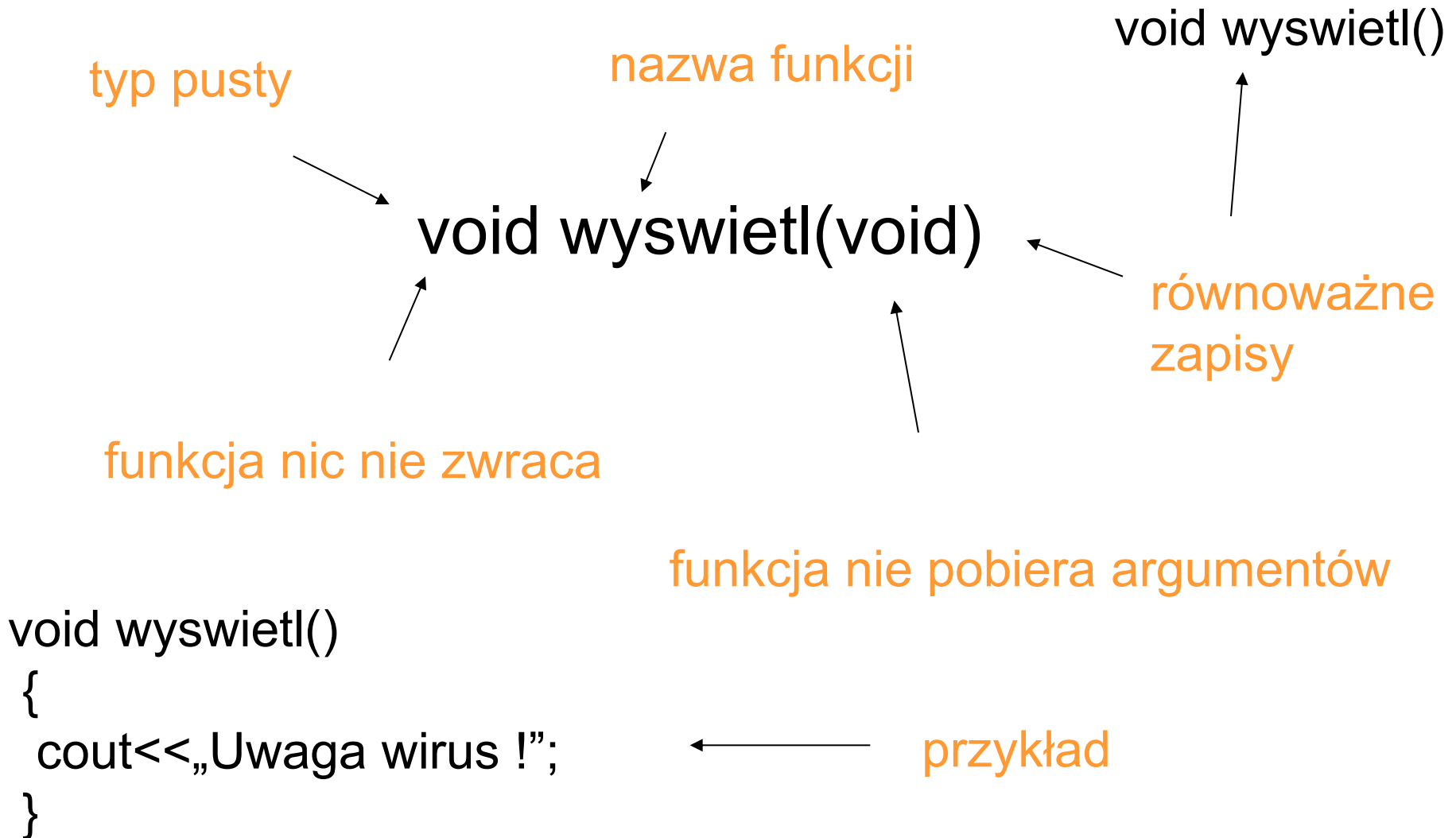
}
```

kompilator musi wiedzieć co oznacza komunikat()



Funkcje

Deklaracja funkcji, która nic nie zwraca ani nie pobiera



Funkcje

Deklaracja funkcji

wykładnik
 x^n
podstawa

nazwa funkcji

int potega(int x, int n)

typ zwracanej wartości

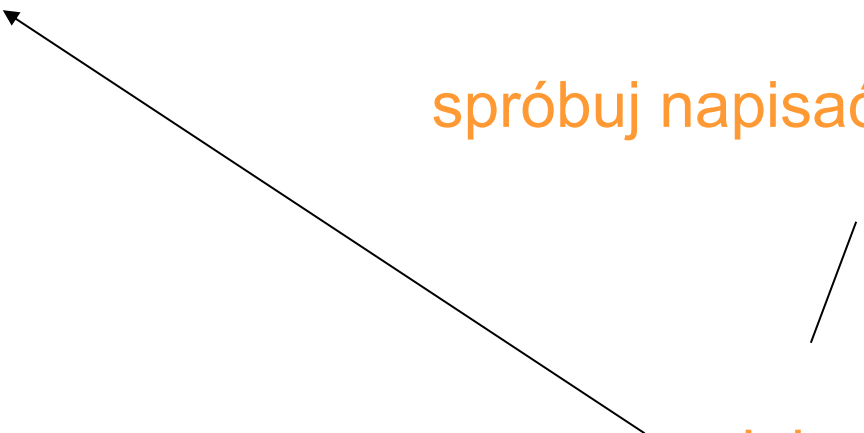
argumenty (parametry) formalne

Definicja funkcji

```
int potega (int x, int n)
{
    ...
}
```

spróbuj napisać samodzielnie

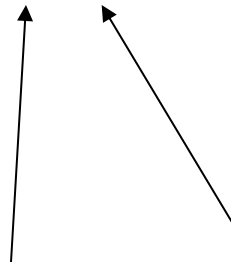
ciało funkcji

A diagram consisting of two orange text labels and a black arrow. The label 'spróbuj napisać samodzielnie' is positioned above the label 'ciało funkcji'. A black arrow originates from the 'ciało funkcji' label and points diagonally upwards and to the left, terminating at the ellipsis '...' within the function body of the code snippet.

Funkcje

Wywołanie funkcji

```
int k;  
k=potega(2,3);
```



argumenty aktualne

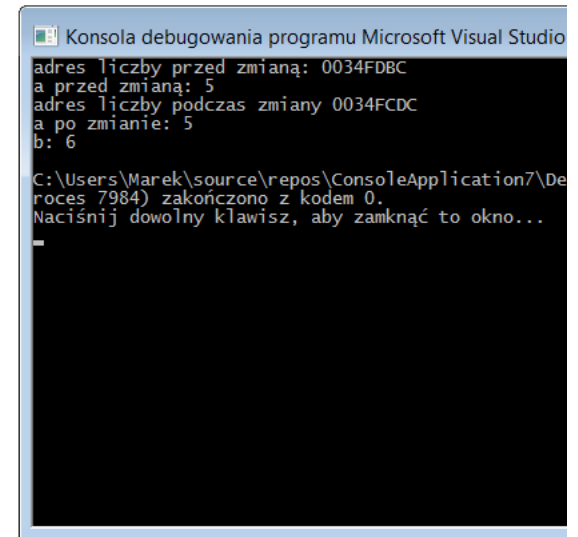
Argumenty funkcji przekazywane przez wartość

```
#include <iostream>

using namespace std;

int change(int a) {
    cout << "adres liczby podczas zmiany " << &a << endl;
    a++;
    return a;
}

int main()
{
    int a = 5;
    cout << "adres liczby przed zmianą: " << &a << endl;
    cout << "a przed zmianą: " << a << endl;
    int b = change(a);
    cout << "a po zmianie: " << a << endl;
    cout << "b: " << b << endl;
}
```



```
Konsola debugowania programu Microsoft Visual Studio
adres liczby przed zmianą: 0034FDBC
a przed zmianą: 5
adres liczby podczas zmiany 0034FCDC
a po zmianie: 5
b: 6

C:\Users\Marek\source\repos\ConsoleApplication7\De
roces 7984) zakończono z kodem 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

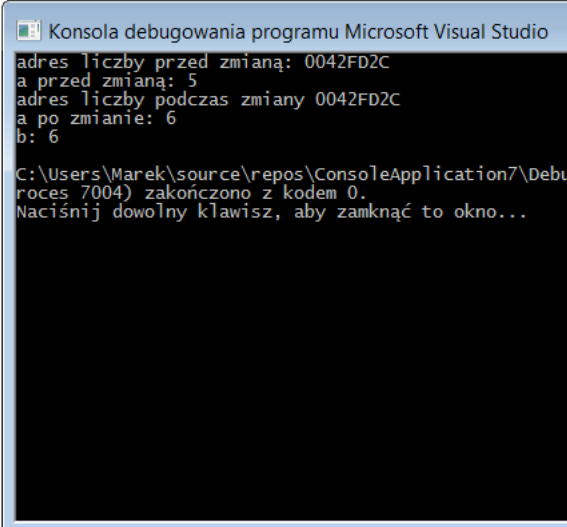

Argumenty funkcji przekazywane przez referencję

```
#include <iostream>

using namespace std;

int change(int& a) {
    cout << "adres liczby podczas zmiany " << &a << endl;
    a++;
    return a;
}

int main()
{
    int a = 5;
    cout << "adres liczby przed zmianą: " << &a << endl;
    cout << "a przed zmianą: " << a << endl;
    int b = change(a);
    cout << "a po zmianie: " << a << endl;
    cout << "b: " << b << endl;
}
```



```
Konsola debugowania programu Microsoft Visual Studio
adres liczby przed zmianą: 0042FD2C
a przed zmianą: 5
adres liczby podczas zmiany 0042FD2C
a po zmianie: 6
b: 6
C:\Users\Marek\source\repos\ConsoleApplication7\Debug
proces 7004) zakończono z kodem 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

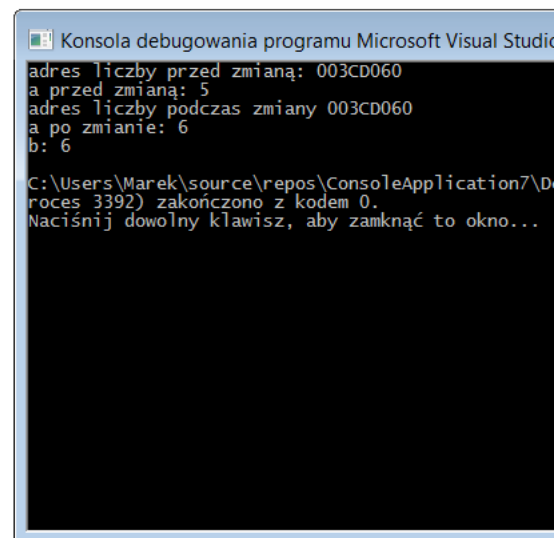
Argumenty funkcji przekazywane przez wskaźnik

```
#include <iostream>

using namespace std;

int change(int* a) {
    cout << "adres liczby podczas zmiany " << a << endl;
    (*a)++;
    return *a;
}

int main()
{
    int* a = new int(5);
    cout << "adres liczby przed zmianą: " << a << endl;
    cout << "a przed zmianą: " << *a << endl;
    int b = change(a);
    cout << "a po zmianie: " << *a << endl;
    cout << "b: " << b << endl;
    delete a;
    a = nullptr;
}
```

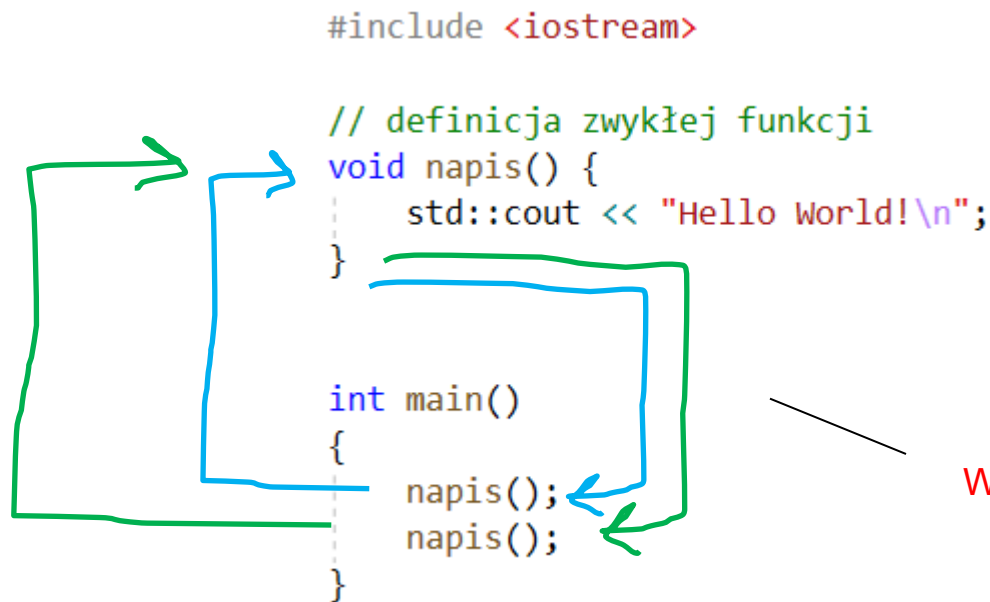


Konsola debugowania programu Microsoft Visual Studio

```
adres liczby przed zmianą: 003CD060
a przed zmianą: 5
adres liczby podczas zmiany 003CD060
a po zmianie: 6
b: 6

C:\Users\Marek\source\repos\ConsoleApplication7\Debug\
proces 3392) zakończono z kodem 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Funkcje - wywołanie zwykłej funkcji



Wywołanie zwykłej funkcji zajmuje czas

Jeśli wywołujemy zwykłą funkcję `napis()` program:

- zapamiętuje adres powrotu – adres kolejnej instrukcji za instrukcją skoku (trzeba będzie do programu wrócić)
- argumenty funkcji kopiuje na stos
- skacze pod adres funkcji `napis()` w pamięci
- wykonuje kod funkcji
- wartość, którą funkcja zwraca umieszcza w rejestrze (jeśli jest taka potrzeba)
- wykonuje skok powrotny pod zapamiętany adres powrotny

Dzieje się to w czasie wykonywania programu (runtime) i zajmuje czas.

Funkcje inline – przyspieszenie działania programu

```
#include <iostream>

// definicja funkcji inline
inline void napis() {
    std::cout << "Hello World!\n";
}
```

Jeśli funkcja jest inline kompilator zastępuje wywołanie funkcji jej kodem, dzieje się to w czasie kompilacji programu (compile time). Mówimy, że ich wywołania są **rozwijane** przez kompilator.

```
int main()
{
    napis();
    napis();
}
```



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

Wywołanie funkcji inline zajmuje mniej czasu, (nie ma skoku do adresu funkcji, zapamiętania argumentów itp.) ale w przypadku wielokrotnego wywołania zajmuje więcej pamięci (pojawiają się kopie kodu funkcji).

Aby osiągnąć zysk z zastosowania funkcji inline – czas jej wykonywania musi być krótki, W praktyce oznacza to, że jej ciało powinno mieścić się w jednej lub 2 liniach.

W praktyce kompilator nie musi dostosować się do wymogu funkcji inline – może uznać, że funkcja jest zbyt duża, zależy to od implementacji kompilatora.

Przeciążenie funkcji

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

polimorfizm funkcji – funkcja może przybierać wiele form

```
void suma(int, int);
void suma(double, double);
```

Funkcje przeciążone – mają taką samą nazwę lecz różnią się sygnaturą (do której zaliczamy ilość i typ parametrów).

```
int main()
```

```
{
    suma(5, 4);
    suma(5.0, 4.0);
}
```

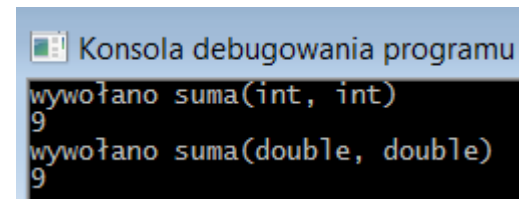
Którą funkcję wywoła kompilator?

```
void suma(int a, int b)
```

```
{
    cout << "wywołano suma(int, int)" << endl;
    cout << a + b << endl;
}
```

```
void suma(double a, double b)
```

```
{
    cout << "wywołano suma(double, double)" << endl;
    cout << a + b << endl;
}
```



```
Konsola debugowania programu
wywołano suma(int, int)
9
wywołano suma(double, double)
9
```

Kompilator dopasowuje argumenty wywołania funkcji z jej parametrami formalnymi w definicji. Wybiera tę funkcję, która najlepiej pasuje.

Przeciążenie funkcji

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

```
void suma(int, int);
int suma(int, int);
```

nie można przeciążyć funkcji w ten sposób

int suma(int, int)

+1 przeciążenie

[Wyszukaj w trybie online](#)

nie można przeciążyć funkcji różniących się tylko typem zwracanej wartości

[Wyszukaj w trybie online](#)

wartość zwracana nie należy do sygnatury funkcji

Przeciążenie funkcji

pobaw się we kompilator ☺



```
suma("suma liczb wynosi", 12.0F); // wywołanie 1
suma(12L, 23);                      // wywołanie 2
suma(1, 2, 3);                      // wywołanie 3
suma(1, 2.0, 3);                    // wywołanie 4
suma(123, 345);                     // wywołanie 5
```



dopasuj wywołanie funkcji do jej deklaracji

```
void suma(int a, int b);           // 1
void suma(long a, int b);          // 2
void suma(int a, int b, int c);     // 3
void suma(int a, double b, int c);  // 4
void suma(const char* napis, float b); // 5
```

Rekurencja

Przykładowe zadanie 21.

```
function suma($n)
{
    if($n < 1) return 0;
    return $n + suma($n - 1);
}
echo suma(5);
```

Która wartość zostanie wyświetlona w wyniku działania skryptu PHP?

- A. 0
- B. 10
- C. 14
- D. 15

Rekurencja

```
#include <iostream>
```

```
int suma(int a);
```

```
int main()
```

```
{
```

```
    std::cout << "Rekurencja\n";
```

```
    std::cout << suma(5);    //?
```

```
}
```

```
int suma(int a)
```

```
{
```

```
    if (a < 1) return 0;
```

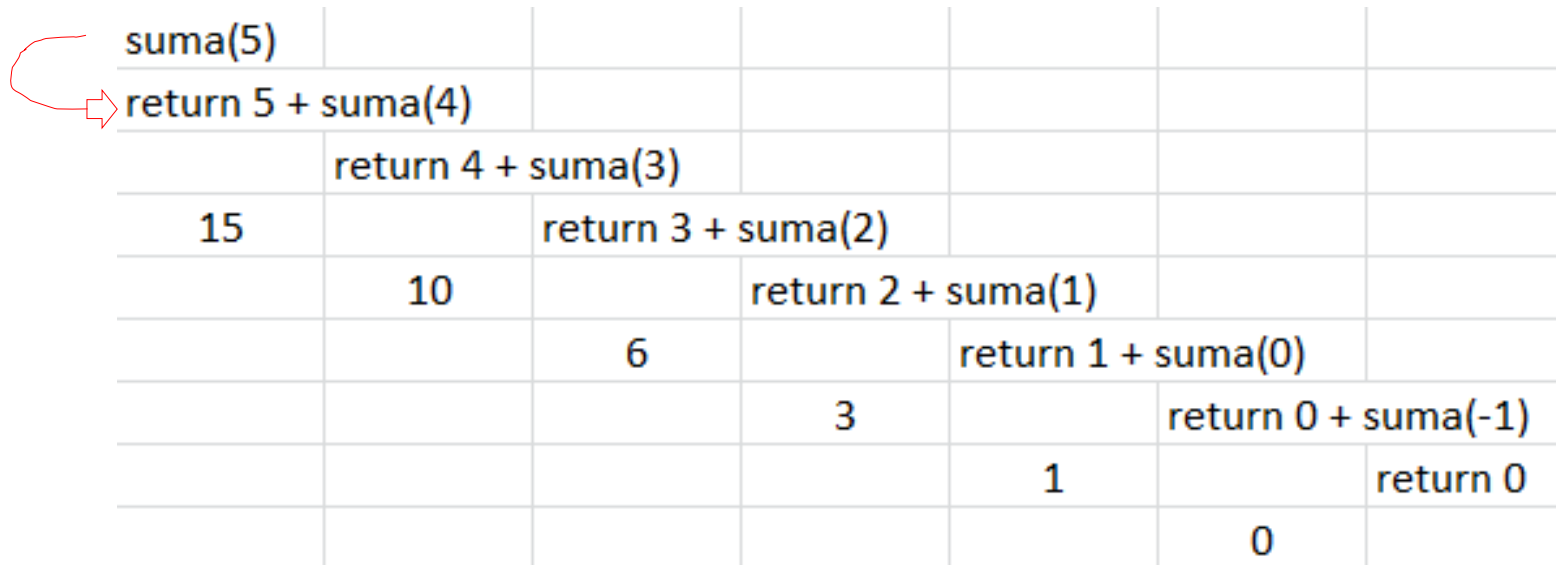
```
    return a + suma(a - 1);
```

```
}
```

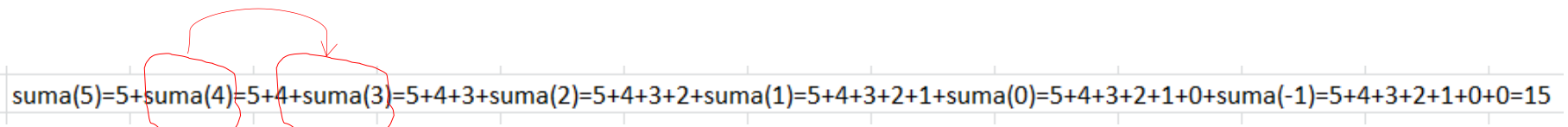
definicja wyjścia z wywołań rekurencyjnych

funkcja suma() wywołuje samą siebie

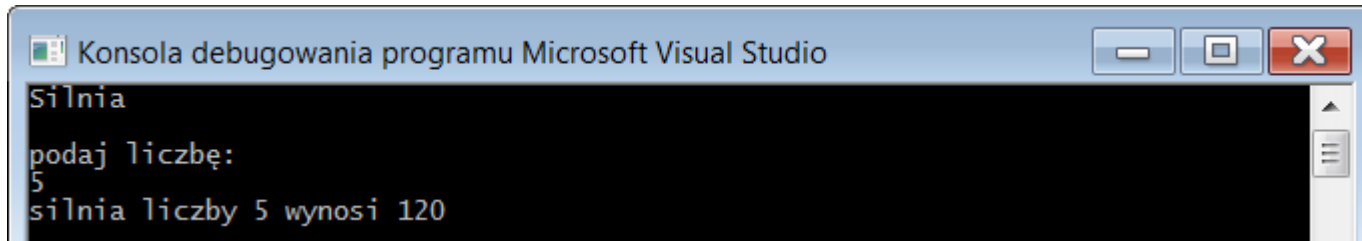
Rekurencja



suma(5)						
return 5 + suma(4)						
	return 4 + suma(3)					
15		return 3 + suma(2)				
	10		return 2 + suma(1)			
		6		return 1 + suma(0)		
			3		return 0 + suma(-1)	
				1		return 0
					0	



$\text{suma}(5) = 5 + \text{suma}(4) = 5 + 4 + \text{suma}(3) = 5 + 4 + 3 + \text{suma}(2) = 5 + 4 + 3 + 2 + \text{suma}(1) = 5 + 4 + 3 + 2 + 1 + \text{suma}(0) = 5 + 4 + 3 + 2 + 1 + 0 + \text{suma}(-1) = 5 + 4 + 3 + 2 + 1 + 0 + 0 = 15$



```
Konsola debugowania programu Microsoft Visual Studio
Silnia
podaj liczbę:
5
silnia liczby 5 wynosi 120
```

$5! = 1*2*3*4*5 = 120$

należy wykorzystać rekurencję

Podział programu na pliki

deklarację funkcji możemy przenieść do pliku nagłówkowego komunikat.h

funkcje.cpp

```
#include<iostream>

using namespace std;

//deklaracja funkcji komunikat
void komunikat();

int main() {

    //wyświetlamy komunikat
    komunikat();

}

//definicja funkcji komunikat
void komunikat() {

    cout << "Uwaga WIRUS!!!" << endl;

}
```

definicję funkcji możemy przenieść do pliku komunikat.cpp

komunikat.h

```
// zabezpieczenia pliku nagłówkowego
// przed wielokrotnym dołączaniem
// do tego samego projektu
// #pragma once

// lub

// zabezpieczenie dyrektywami preprocesora
#ifndef KOMUNIKAT_H
#define KOMUNIKAT_H

//deklaracja funkcji komunikat
void komunikat();

#endif /* KOMUNIKAT_H */
```

komunikat.cpp

```
#include<iostream>

//definicja funkcji komunikat
void komunikat() {

    std::cout << "Uwaga WIRUS!!!" << std::endl;

}
```

dołączamy plik nagłówkowy komunikat.h z deklaracją funkcji za pomocą dyrektywy preprocesora #include

funkcje.cpp

```
#include<iostream>
#include"komunikat.h"

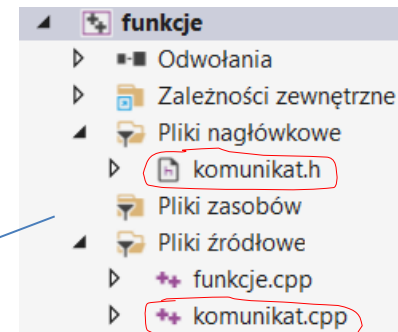
using namespace std;

int main() {

    //wyświetlamy komunikat
    komunikat();

}
```

Eksplorator rozwiązań z nowymi plikami



REFERENCJA

Referencja Reference

są często używane przy przekazywaniu argumentów do funkcji przez referencję (funkcja nie tworzy wtedy kopii argumentów tylko pracuje na oryginalnych wartościach)

inna nazwa (alias) zmiennej

```
std::cout << "Referencje!\n";
```

b staje się inną nazwą zmiennej a

```
short a = 10;
```

```
// tworzymy referencję (do typu short) zmiennej a  
short& b = a;
```

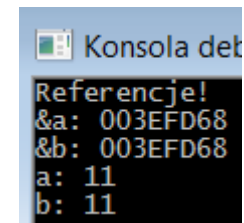
adres zmiennej a oraz b
jest taki sam w pamięci RAM

```
// adresy w pamięci zmiennej a oraz b  
std::cout << "&a: " << &a << std::endl;  
std::cout << "&b: " << &b << std::endl;
```

```
// inkrementujemy a  
a += 1;
```

```
std::cout << "a: " << a << std::endl; // 11  
std::cout << "b: " << b << std::endl; // 11
```

po inkrementacji zmiennej a
inkrementowała się również zmienna b

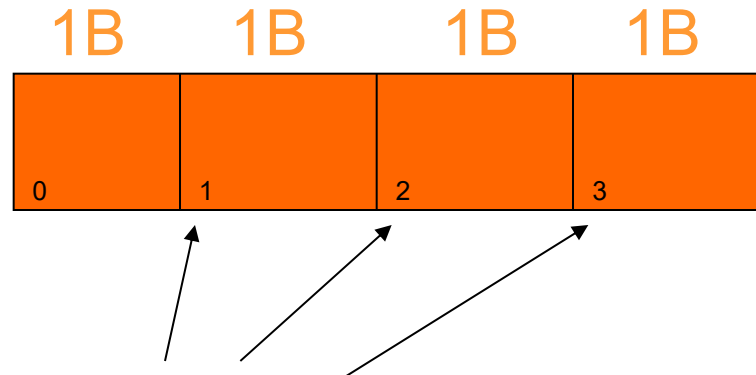


```
Konsola de  
Referencje!  
&a: 003EFD68  
&b: 003EFD68  
a: 11  
b: 11
```

WSKAŹNIKI

Wskaźnik Pointer

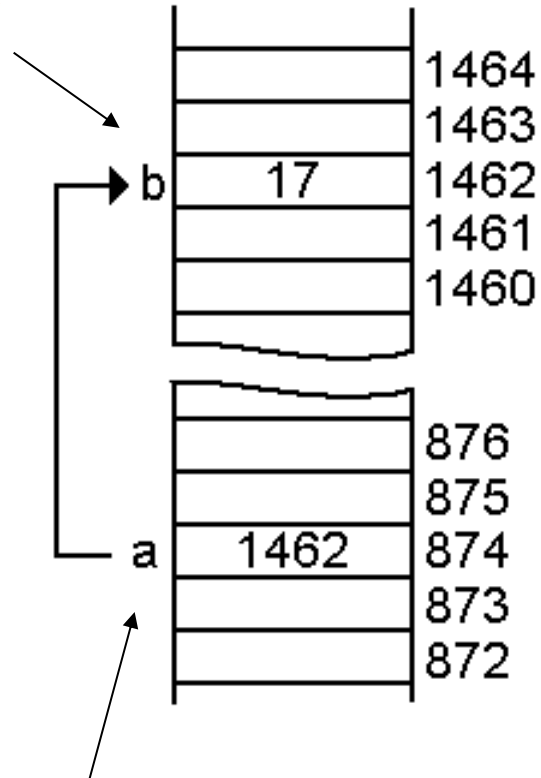
Pamięć w komputerach może być reprezentowana jako jednowymiarowa tablica bajtów (płaski model pamięci)



wskaźnik jest indeksem tej tablicy
(adresem danej komórki w pamięci)

Wskaźnik Pointer

zmienna

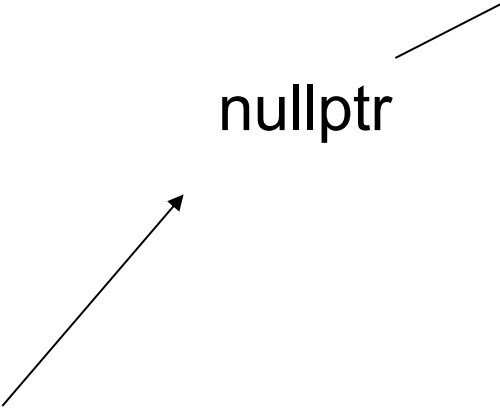


wskaźnik na zmienną

Wskaźnik pusty The null pointer

wskaźnik pusty (C++11)

nullptr



Wskaźnik taki nie wskazuje na nic (konkretnie: jego wartość liczbowa jako adresu jest równa 0), gdyż w nowoczesnych systemach operacyjnych żaden proces nie ma dostępu do komórki pamięci o adresie 0, stąd też jest ona wykorzystywana do oznaczenia *wskazania do niczego*.

Wskaźnik pusty NULL // c

nie preferowane

```
// NULL jako wskaźnik pusty  
int* a = NULL;    // C, C++98
```

```
cout << NULL;    // 0  
cout << a;        // 00000000
```

```
// NULL jako liczba  
int b = NULL;
```

```
cout << b;        // 0
```

▶ #define NULL 0

Jest rozwijane do:0

[Wyszukaj w trybie online](#)

zerowy adres 32 bitowy

preferowane

```
// nullptr oznacza tylko wskaźnik pusty  
int* c = nullptr; // C++11
```

```
cout << c;        // 00000000
```

```
int d = nullptr; // błąd
```

zerowy adres 32 bitowy

nie można użyć wartości typu "std::nullptr_t" do zainicjowania jednostki typu "int"

[Wyszukaj w trybie online](#)

Wskaźniki

Assume that `a` is located at address `0x8130` in memory and `money` at `0x8134`; also assume this is a 32-bit machine such that an `int` is 32-bits wide. The following is what would be in memory after the following code snippet is executed

```
int a = 5;  
int *money = NULL;
```

Address	Contents
0x8130	0x00000005
0x8134	0x00000000



The NULL pointer shown here is 0x00000000

Wskaźniki

By assigning the address of a to money

```
money = &a;
```

yields the following memory values

Address	Contents
0x8130	0x00000005
0x8134	0x00008130

Wskaźniki

Then by dereferencing money by doing

```
*money = 8;
```

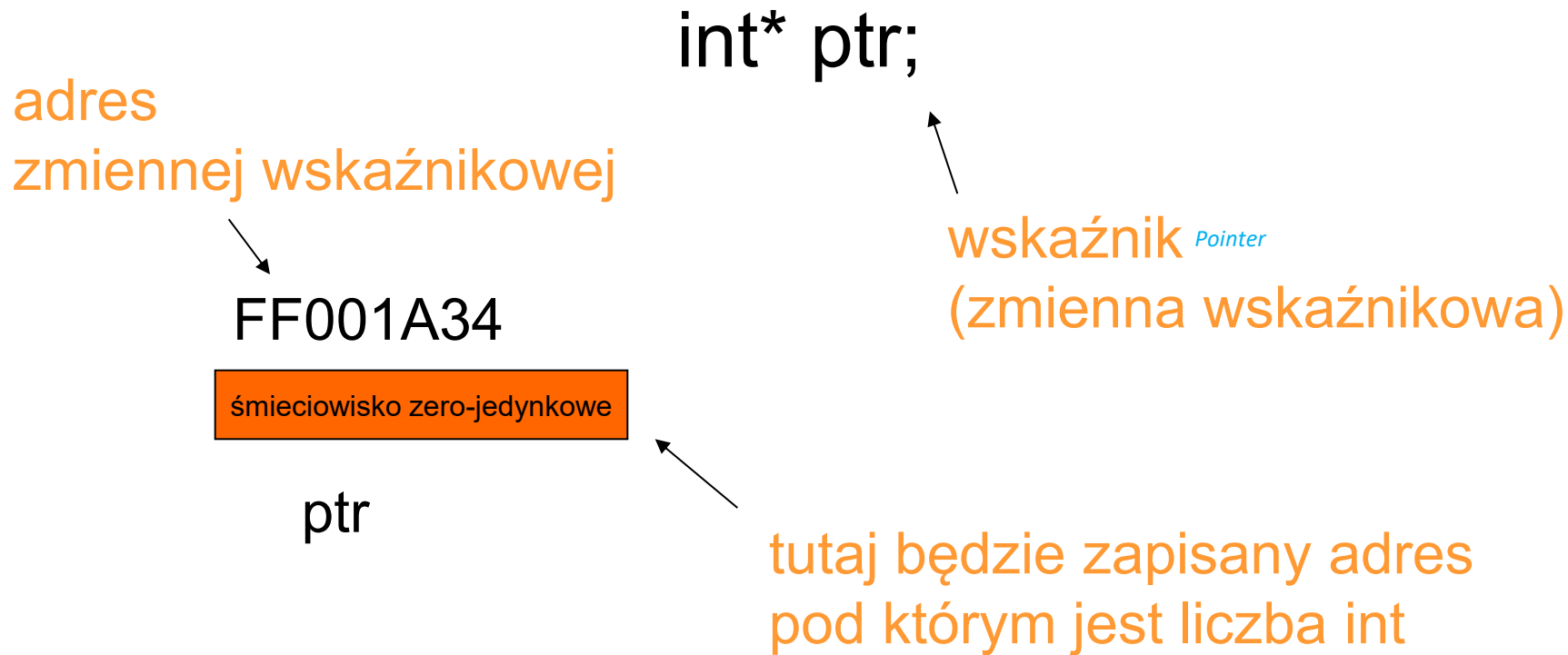
the computer will take the contents of money (which is 0x8130), go to that address, and assign 8 to that location yielding the following memory

Address	Contents
0x8130	0x00000008
0x8134	0x00008130

Wskaźniki

Deklaracja (i definicja) wskaźnika pokazującego na liczbę typu int

Deklaracja (i definicja) zmiennej wskaźnikowej pokazującej na liczbę typu int



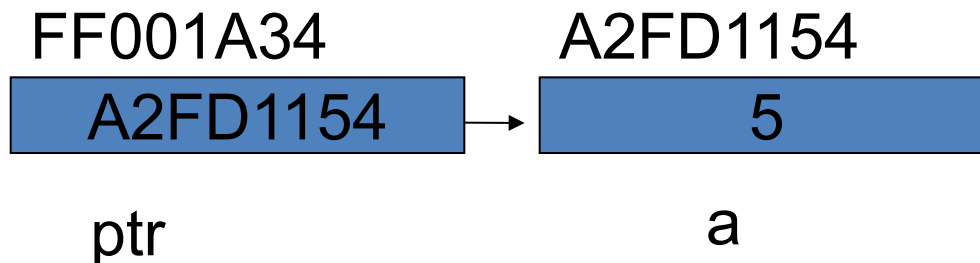
Wskaźniki

```
int* ptr;           //deklaracja i definicja wskaźnika  
                    //wskazującego na liczbę typu integer  
                    //(całkowitą) w pamięci zostaje utworzona  
                    //zmienna wskaźnikowa ptr
```

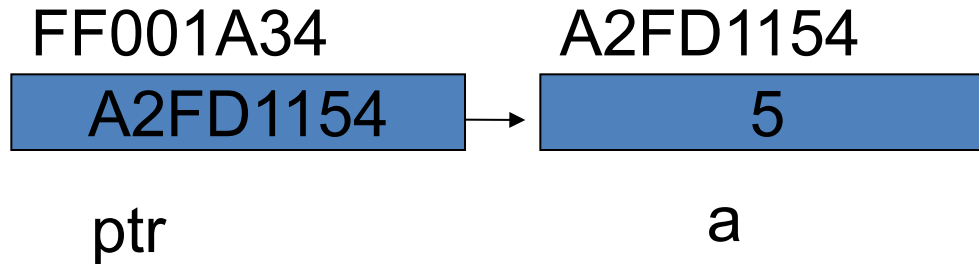
```
int a = 5;          //deklaracja, definicja i inicjalizacja  
                    //zmiennej a typu integer
```

```
ptr = &a            //przypisanie adresu zmiennej a  
                    //do wskaźnika ptr
```

operator adresu



Wskaźniki



//wypisanie na konsoli liczby 5

//tradycyjny sposób

```
cout<<"liczba a wynosi "<<a;
```

operator wyłuskania,
dereferencji
(gwiazdka)

//wypisanie na konsoli liczby 5

//za pomocą wskaźnika

```
cout<<"liczba a wynosi "<<* ptr;
```

//wypisanie na konsoli zawartości

//zmiennej wskaźnikowej p,

//czyli adresu zmiennej a

```
cout<<"adresem zmiennej a w pamięci RAM";
```

```
cout<<"jest liczba szesnastkowa "<< ptr;
```

Wskaźniki

wskaźnik na typ int

`int* ptr;`

FF001A

śmieciowisko zero-jedynkowe

ptr

stworzenie nowego obiektu typu int
adres obiektu przypisujemy do wskaźnika

`ptr = new int;`

heap
alokuje pamięć na stercie

FF001A

A2FD11

ptr

A2FD11

śmieciowisko zero-jedynkowe

W komórce pamięci pod adresem
FF001A zostało stworzone
miejsce przeznaczone na adres.
Na razie jest wypełnione śmieciami.

nowy obiekt typu int
Tu można
wrzucić liczbę
typu int

Wskaźniki

*ptr=25;



FF001A



ptr

A2FD11



komplecik!

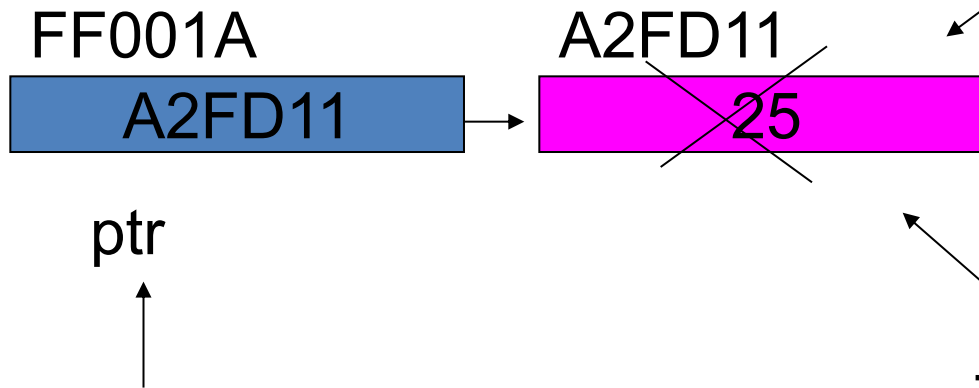
Wskaźniki

bez tego wyciek
pamięci w
programie

delete ptr;

usunięcie obiektu
typu int

Uwaga!
Nie jest to usunięcie
wskaźnika!

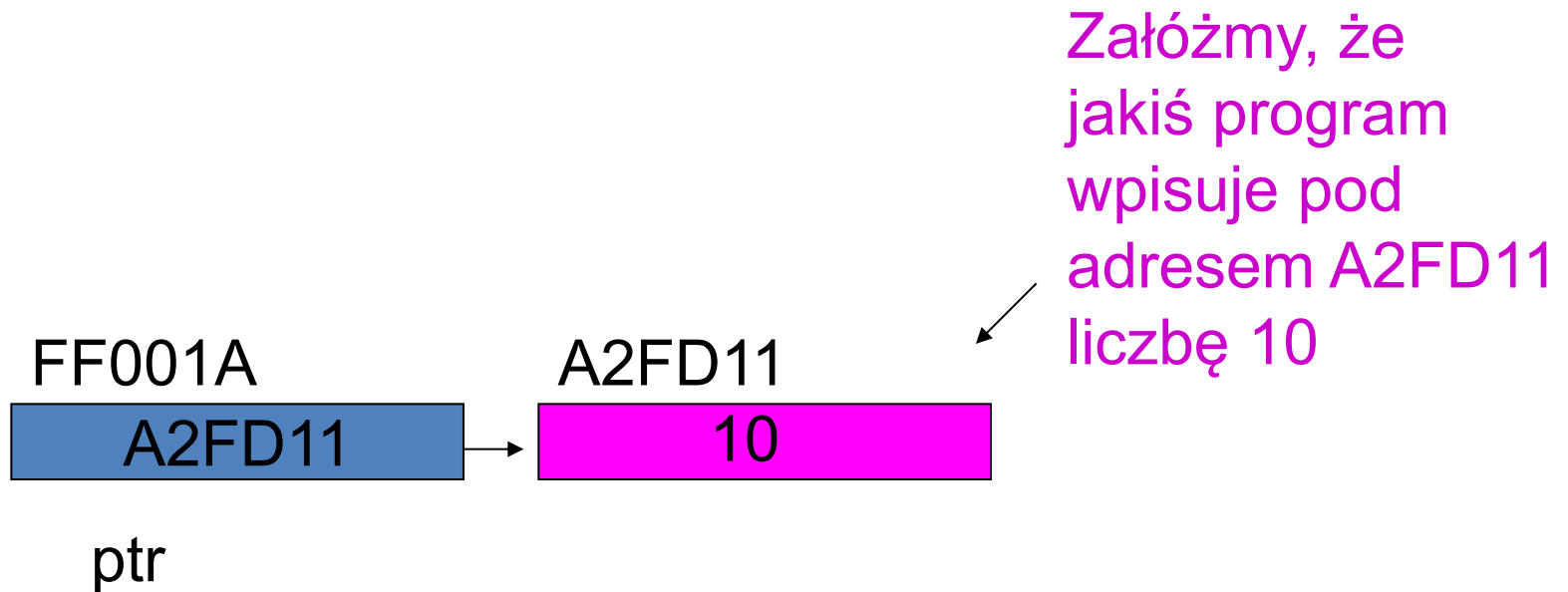


we wskaźniku nic się nie zmienia!

ptr nadal wskazuje na
ten sam obszar pamięci!
(to może zależeć od kompilatora)

Ten obszar pamięci
już nie należy do
naszego programu!
Został zwolniony,
mogą z niego korzystać
inne programy.

Wskaźniki



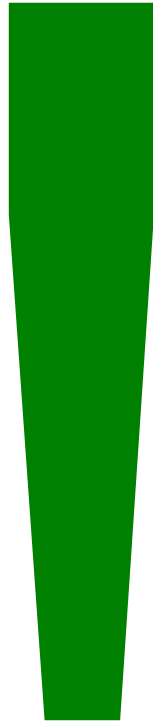
Jeśli po raz drugi wykonamy instrukcję:

`delete ptr;`

to ...

Wskaźniki

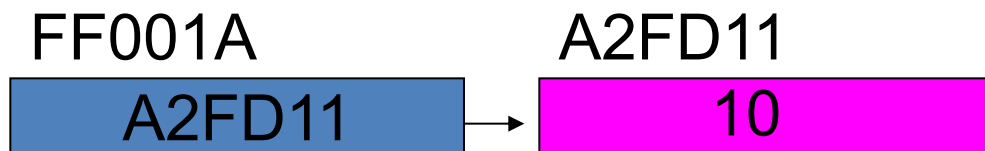
undefined behaviour



Wskaźniki

System może zgłosić *access violation* czyli **błąd ochrony pamięci**, może być runtime crash lub coś innego.

Ten obszar już nie należy do naszego programu

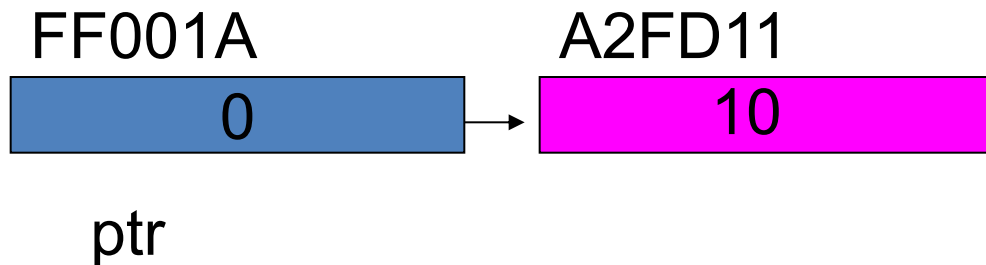


Możemy przypadkowo zniszczyć obiekty nie należące do naszego programu

Dlatego należy wyzerować wskaźnik:

```
ptr = nullptr;
```

Wskaźniki



Jeśli teraz wywołamy ponownie instrukcję:

delete ptr;

Ten wiersz zostanie zignorowany.

Wskaźnik wskazuje na adres zerowy, którego nie ma. Wskaźnik nie wskazuje na żadną konkretną komórkę pamięci. Nie ma czego usuwać!

Wskaźniki

example of CRASH

```
int* ptr = new int(2);
```

```
cout << *ptr << endl;
```

```
cout << ptr << endl;
```

```
delete ptr;
```

```
//delete ptr; // crash
```

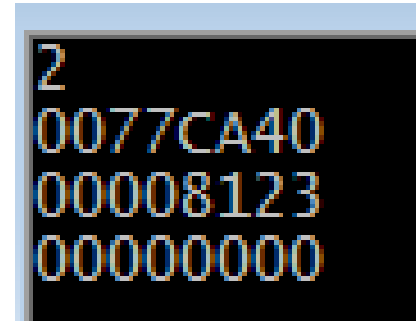
```
cout << ptr << endl;
```

don't do it !

🔍 (zmienna lokalna) `int *ptr`

[Wyszukaj w trybie online](#)

C6001: Używanie niezainicjowanej pamięci „ptr”.



```
//cout << *ptr << endl; //crash
```

```
ptr = nullptr;
```

```
cout << ptr << endl;
```

OK

don't do it !

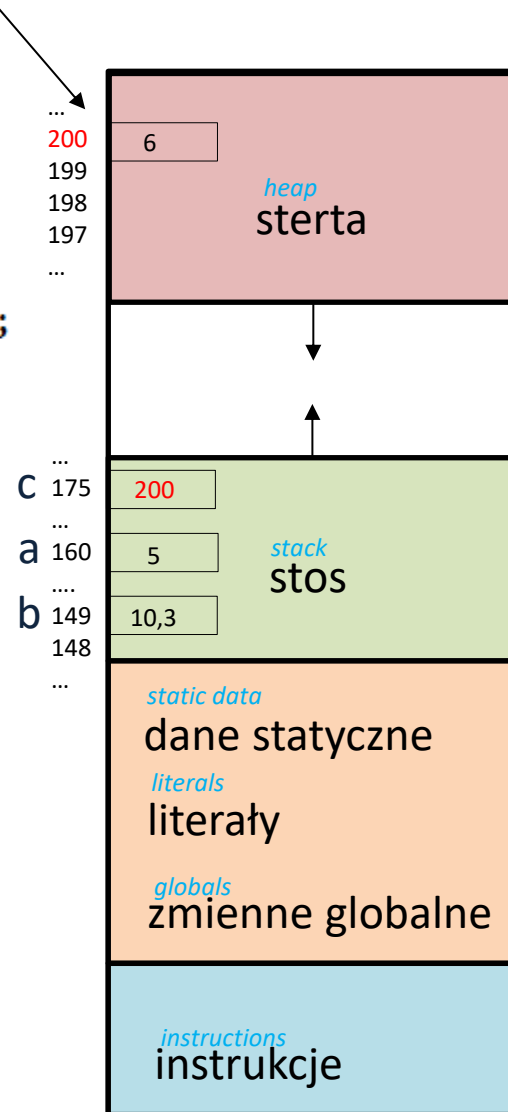
Pamięć programu

zmienne na sterpie są zarządzane przez programistę !

```
int main()
{
    int* c = new int(6);

    int a = 5;

    double b = 10.3;
}
```



runtime memory allocation

dynamic memory allocation

zmienne dla których pamięć jest przydzielana dynamicznie - alokowana w czasie działania programu (ich wielkość nie jest znana w czasie kompilacji),

static memory allocation

zmienne lokalne dla których pamięć alokowana jest w czasie kompilacji (ich wielkość jest znana w czasie kompilacji), zarządzana automatycznie przez kompilator,

RAM

Analiza wycieków pamięci przy użyciu debuggera

Dynamiczna alokacja pamięci na sterpie – wyciek pamięci

Przykład 1

```
void f() {  
    // alokujemy pamięć na sterpie w pętli nieskończonej  
    while(true)  
    {  
        long long* c = new long long(1000000);  
    }  
}  
  
int main()  
{  
    // funkcja alokująca pamięć na sterpie  
    f();  
  
    int d = 5;  
  
    double b = 10.3;  
}
```

Dynamiczna alokacja pamięci na stercie – wyciek pamięci

Przykład 1

The screenshot displays the Visual Studio IDE with the following components:

- Code Editor:** Shows `ConsoleApplication5.cpp` with the following code:

```
1 #include <iostream>
2 using namespace std;
3
4 void f() {
5     while (true) {
6         long long* c = new long long (100000);
7     }
8 }
9
10 int main()
11 {
12     f();
13     int a = 5;
14     double b = 10.3;
15 }
```

Arrows indicate that the infinite loop in `f()` is the source of the memory leak, leading to the high memory usage shown in the Task Manager and Diagnostic Tools windows.
- Windows Task Manager:** Opened to the 'Procesy' tab. It shows:
 - Procesy: 116
 - Procesor CPU: 37%
 - Pamięć fizyczna: 93%
- Narzędzia diagnostyczne (Diagnostic Tools):** Opened to the 'Użycie pamięci' (Memory Usage) tab. It shows a graph of memory usage over time, which is increasing steadily, confirming the memory leak.

pętla nieskończona alokująca pamięć, która wycieka – tracimy dostęp do zaalokowanych adresów – nadpisujemy wskaźnik `long long* c`

Dynamiczna alokacja pamięci na stercie – wyciek pamięci

Przykład 1

The screenshot displays the Visual Studio IDE with the following components:

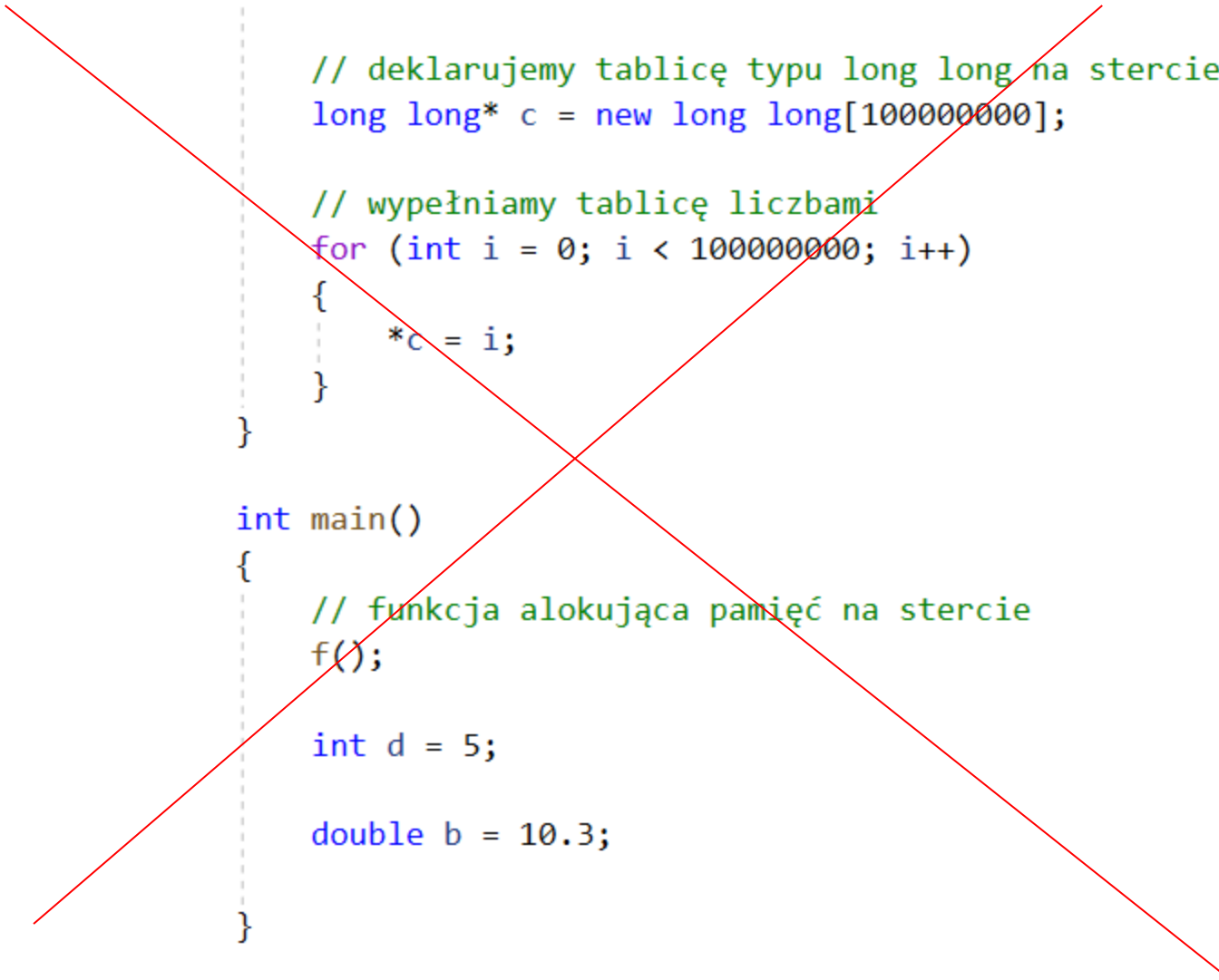
- Code Editor:** Shows the source file `ConsoleApplication5.cpp`. The code includes a function `__srt_throw_std_bad_alloc()` that calls `TerminateProcess(GetCurrentProcess(), 3);` and throws `std::bad_alloc`.
- Debugger:** The `Proces` dropdown shows `[5744] ConsoleApplication5.exe`. The `Wątek` dropdown shows `[6952] Wątek główny`. The `Ramka stosu` dropdown shows `__srt_throw_std_bad_alloc`.
- Exception Dialog:** A window titled "Nieobsługiwany wyjątek" (Unhandled exception) displays the message: "Nieobsłużony wyjątek w lokalizacji 0x7605C5AF w ConsoleApplication5.exe: wyjątek języka Microsoft C++: std::bad_alloc w lokalizacji pamięci 0x001AF838."
- Windows Task Manager:** A window titled "Menedżer zadań Windows" (Windows Task Manager) shows the `Użycie CPU` (CPU Usage) and `Historia użycia procesora` (Processor Usage History) graphs. The `Użycie CPU` graph shows 0% usage. The `Historia użycia procesora` graph shows a peak usage of 100%.
- Visual Studio Diagnostic Tools:** A window titled "Narzędzia diagnostyczne" (Diagnostic Tools) shows the `Sesja diagnostyki: 2:53 min (wybrano 2:53 min)` (Diagnostic session: 2:53 min (selected 2:53 min)). The `Użycie pamięci` (Memory Usage) graph shows a peak usage of 2.2 GB. The `Procesor CPU (% wszystkich procesorów)` (CPU Processor (% of all processors)) graph shows a peak usage of 100%.

Po wyczerpaniu dostępnej pamięci - wyjątek

Dynamiczna alokacja pamięci na sterpie – wyciek pamięci

Przykład 2

```
void f() {  
  
    // deklarujemy tablicę typu long long na sterpie  
    long long* c = new long long[100000000];  
  
    // wypełniamy tablicę liczbami  
    for (int i = 0; i < 100000000; i++)  
    {  
        *c = i;  
    }  
}  
  
int main()  
{  
    // funkcja alokująca pamięć na sterpie  
    f();  
  
    int d = 5;  
  
    double b = 10.3;  
}
```



Dynamiczna alokacja pamięci na stercie – wyciek pamięci

Przykład 2

The screenshot shows a Visual Studio IDE with a C++ project named 'ConsoleApplication5'. The code in 'ConsoleApplication5.cpp' is as follows:

```
4 void f() {  
5  
6     long long* c = new long long[100000000] ;  
7  
8     for (int i = 0; i < 100000000; i++)  
9     {  
10         *c = i;  
11     }  
12 }  
13  
14  
15  
16 int main()  
17 {  
18  
19     f();  
20  
21     int d = 5;  
22  
23     double b = 10.3;  
24 }  
25  
26  
27
```

A blue arrow points to line 19, with the text 'tu jesteśmy' (we are here) next to it. A red arrow points from the 'Pamięć' (Memory) section of the Windows Task Manager to the same line in the code. The Task Manager shows that the process is using 2.75 GB of memory.

Chcemy zaalokować 842 MB pamięci na stercie za pomocą funkcji f()

Dynamiczna alokacja pamięci na stercie – wyciek pamięci

Przykład 2

The screenshot illustrates a memory leak in a C++ application. The code in `ConsoleApplication5.cpp` shows a function `f()` that allocates a large array on the stack:

```
void f() {  
    long long* c = new long long[100000000];  
    for (int i = 0; i < 100000000; i++)  
    {  
        *c = i;  
    }  
}
```

The `main()` function calls `f()` and then declares some variables:

```
int main()  
{  
    f();  
    int d = 5;  
    double b = 10.3;  
}
```

The Windows Task Manager window shows the system's memory usage, with a graph indicating that 3.52 GB of physical memory is being used. The Visual Studio Diagnostic Tools window shows the process's memory usage, with a graph indicating that 842 MB of process memory is being used. A blue arrow points to the allocation line in the code, and a red arrow points to the 'zaalokowana pamięć' (allocated memory) in the Task Manager memory usage graph.

Procesy: 117 Procesor CPU: 1% Pamięć fizyczna: 45%

Stos Wywołania Punkty przerwania Ustawienia wyjątków Okno polecenia Okno bezpośrednie Dane wyjściowe

Zaalokowaliśmy 842 MB pamięci na stercie

Dynamiczna alokacja pamięci na stercie – wyciek pamięci

Przykład 2

Code Snippet:

```
4 void f() {  
5     long long* c = new long long[100000000];  
6  
7     for (int i = 0; i < 100000000; i++)  
8     {  
9         *c = i;  
10    }  
11  
12 }  
13  
14  
15  
16 int main()  
17 {  
18     f();  
19  
20     int d = 5;  
21  
22     double b = 10.3;  
23  
24     // ...  
25 }  
26  
27
```

Handwritten Notes:

- Po wyjściu z funkcji wskaźnik c jest usuwany ze stosu (After exiting the function, pointer c is removed from the stack)
- tu jesteśmy (przy końcu programu, program jest nadal uruchomiony) (Here we are (at the end of the program, the program is still running))
- brak wskaźnika c (Missing pointer c)

Windows Task Manager:

- Użycie CPU: 1%
- Pamięć: 3.52 GB
- Historia użycia procesora
- Historia użycia pamięci fizycznej
- Wyciek pamięci (Memory leak)

Visual Studio Diagnostic Tools:

- Sesja diagnostyki: 0 s (wybrano 659 ms)
- Zdarzenia
- Pamięć procesu (MB): 842
- Procesor CPU (% wszystkich procesorów): 100
- Podsumowanie
- Zdarzenia
- Użycie pamięci
- Utwórz migawkę
- Wyświetl sterę
- Usuń
- Profilowanie sterę
- Identyfikator
- Godzina
- Alokacje (różnica)
- Rozmiar sterę

System Information:

Pamięć fizyczna (MB)		System	
Razem	7934	Dojścia	40467
Buforowana	3438	Wątki	1630
Dostępna	4325	Procesy	115
Wolna	944	Czas pracy	0:04:00:02
		Zadeklarowane (GB)	6 / 15

Process Information:

Procesy	Procesor CPU	Pamięć fizyczna
115	1%	45%

Po zaalokowaniu 842 MB pamięci na stercie straciliśmy z nią kontakt (wskaźnik c jest lokalną zmienną funkcji, po wyjściu programu z funkcji jest usuwany przez kompilator)

Dynamiczna alokacja pamięci na sterce – wyciek pamięci

Przykład 2

The screenshot shows the Visual Studio IDE with the following components:

- Code Editor:** Displays `ConsoleApplication5.cpp`. The code defines a function `f()` that allocates a large array `long long* c = new long long[100000000];` and a `main()` function that calls `f();` and then returns. A red 'X' is drawn over the code, indicating a memory leak. A blue arrow points to the end of the program with the text "tu jesteśmy (program się zakończył)".
- Windows Task Manager:** Opened over the code editor. It shows the process `ConsoleApplication5.exe` with a memory usage of 2.69 GB. The text "Pamięć została zwolniona" (Memory has been released) is overlaid on the Task Manager window.
- Output Window:** Shows the debug output, indicating that the program has finished execution with a return code of 0.

Code Snippet:

```
void f() {  
    long long* c = new long long[100000000];  
    for (int i = 0; i < 100000000; i++)  
    {  
        *c = i;  
    }  
}  
  
int main()  
{  
    f();  
    int d = 5;  
    double b = 10.3;  
}
```

Task Manager Data:

Pamięć fizyczna (MB)		System	
Razem	7934	Dojścia	40096
Buforowana	3469	Wątki	1623
Dostępna	5171	Procesy	113
Wolna	1769	Czas pracy	0:04:00:32
		Zadeklarowane (GB)	5 / 15

Output Window:

```
„ConsoleApplication5.exe” (Win32): załadowano „C:\Windows\System32\api-ms-win-core-file-l2-1-0.dll”. Symbole zostały załadowane.  
„ConsoleApplication5.exe” (Win32): załadowano „C:\Windows\System32\api-ms-win-core-synch-l1-2-0.dll”. Symbole zostały załadowane.  
Wątek 0x10e4 zakończył działanie z kodem 0 (0x0).  
Program „[7852] ConsoleApplication5.exe” zakończył działanie z kodem 0 (0x0).
```

Zaalkokowana pamięć na sterce jest zwalniana do puli dostępnej pamięci dopiero po zakończeniu programu

Dynamiczna alokacja pamięci na sterpie – wyciek pamięci

Przykład 2

```
void f() {  
  
    // deklarujemy tablicę typu long long na sterpie  
    long long* c = new long long[1000000000] ;  
  
    // wypełniamy tablicę liczbami  
    for (int i = 0; i < 1000000000; i++)  
    {  
        *c = i;  
    }  
  
    // usuwamy wskaźnik na tablicę  
    delete[] c;  
  
    // przypisujemy do wskaźnika nullptr  
    c = nullptr;  
}
```

zwolnienie
Pamięci oraz
zabezpieczenie
wskaźnika



```
int main()  
{  
    // funkcja alokująca pamięć na sterpie  
    f();  
  
    int d = 5;  
  
    double b = 10.3;  
}
```

Dynamiczna alokacja pamięci na sterce – wyciek pamięci

Przykład 2

The screenshot shows the Visual Studio IDE with the following components:

- Code Editor:** Displays `ConsoleApplication5.cpp`. The code defines a function `f()` that allocates a large array `c` and a `main()` function that calls `f()` and then declares variables `d` and `b`. The allocation line `long long* c = new long long[100000000];` is highlighted with a green circle, and the deallocation line `delete[] c;` is circled in green. A blue arrow points to the deallocation line with the text "zwolnienie pamięci oraz zabezpieczenie wskaźnika".
- Debugger Console:** Shows the program is running. The status bar indicates "Nie znaleziono żadnych problemów".
- Windows Task Manager:** Opened over the code editor. The "Procesy" tab shows the process "ConsoleApplication5.exe". The "Historia użycia procesora" and "Historia użycia pamięci fizycznej" graphs show memory usage. A large text overlay on the memory graph says "Pamięć została zwolniona".
- Diagnostic Tools:** Opened on the right. The "Zdarzenia" pane shows a memory leak event. The "Podsumowanie" pane shows the memory usage statistics.

At the bottom of the code editor, there is a green text overlay: "tu jesteśmy (przy końcu programu, program jest nadal uruchomiony)".

Zaallokowaną pamięć na sterce należy zawsze zwolnić gdy nie jest już potrzebna

Paradygmat obiektowy – cechy programowania obiektowego

abstrakcja

przedstawienie obiektów świata realnego jako uproszczonych bytów abstrakcyjnych, które staną się typem własnym użytkownika (klasą) z ukrytymi przed użytkownikiem szczegółami implementacji.

enkapsulacja(hermetyzacja)

- ukrywanie danych przed światem zewnętrznym
- powiązanie danych oraz operacji na nich wykonywanych w ramach klasy,

dziedziczenie

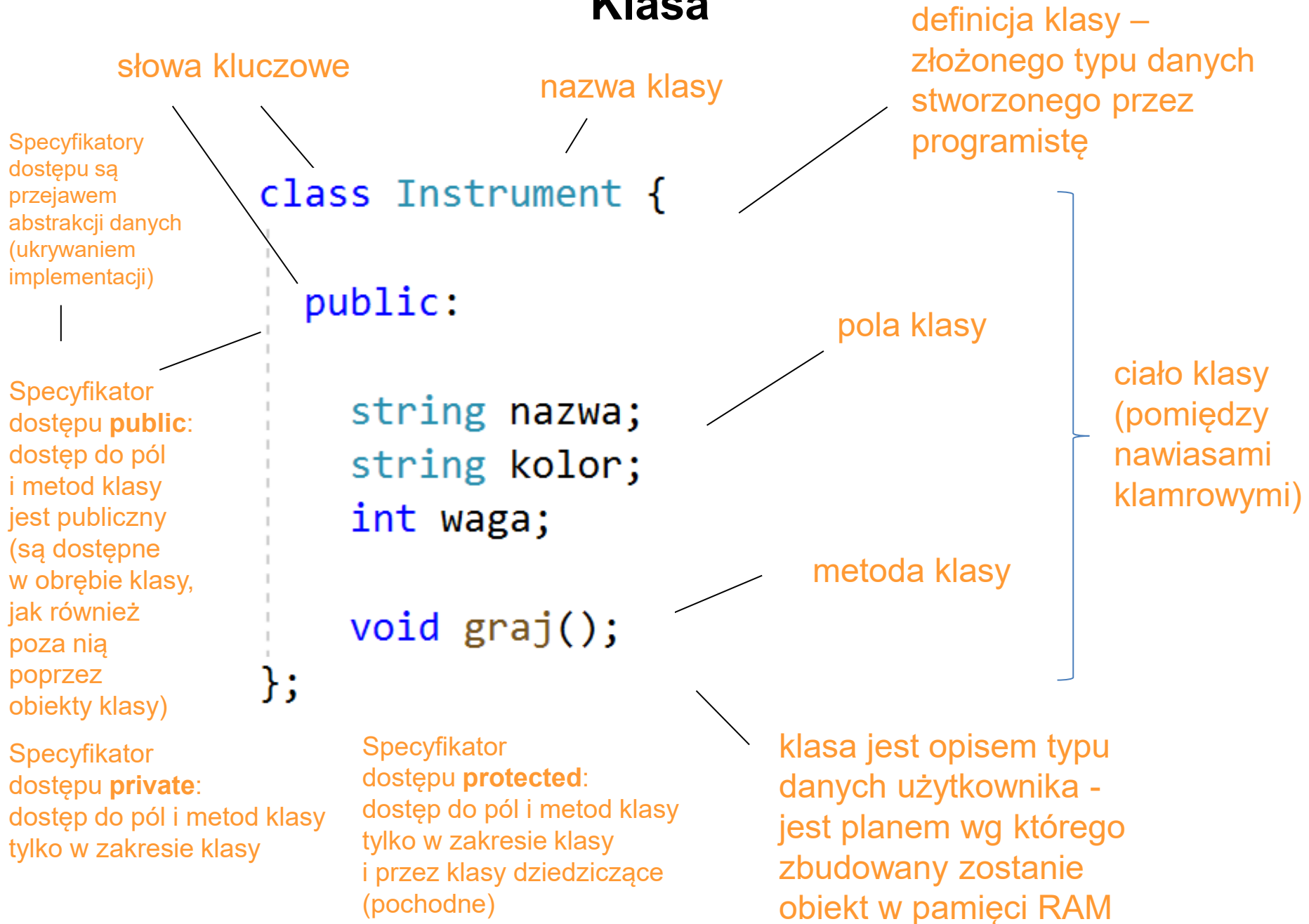
definiowanie klas dziedziczących (pochodnych) na podstawie klasy bazowej (klasy matki): klasy pochodne dziedziczą po klasie bazowej pola i metody

wielopostaciowość:
przeciążanie metod,
przesłanianie metod.
metody wirtualne,
metody czysto wirtualne

polimorfizm

Klasy

Klasa



Klasa – metoda inline

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

```
class Instrumnet {
public:
    string nazwa;
    string kolor;
    int waga;

    void graj() {
        cout << "Pięknie gram" << endl;
    };
};
```

Metoda jest zdefiniowana wewnątrz klasy, dlatego domyślnie przyjmuje się dla niej modyfikator **inline** (to samo tyczy się metod statycznych, konstruktorów i destruktorów). Metody takie powinny być krótkie (jedna linijka kodu ciała metody)

Tworzymy obiekt gitara klasy Instrument

```
int main()
{
    Instrumnet gitara;
    gitara.nazwa = "hawajska";
    gitara.graj();
}
```

Wywołanie metody `graj()` na rzecz obiektu `gitara`



```
int main()
{
    Instrumnet gitara;
    gitara.nazwa = "hawajska";
    cout << "Pięknie gram" << endl;
}
```

Kompilator rozwija metodę `graj()` w miejscu wywołania



Konsola debugowania
Pięknie gram

Klasa

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

```
class A {
private:
    int x;
protected:
    int y;
public:
    int z;
    int setX(int k) { x = k; };
    int setY(int m) { y = m; };
};
```

dostęp do prywatnego pola x
wewnątrz klasy A 😊 OK

dostęp do chronionego pola y
wewnątrz klasy A 😊 OK

```
int main() {
    A a;
    a.x = 5;
    a.setX(5);
    a.y = 3;
    a.setY(3);
    a.z = 2;
}
```

dostęp do
publicznych
pól i metod
poza klasą A
😊 OK

dostęp do prywatnego pola x
poza klasą A 😞 NOK

dostęp do chronionego pola y
poza klasą A 😞 NOK

Klasa Instrument z konstruktorem domyślnym i destruktor

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

// definicja klasy instrumentu muzycznego
class Instrument {
public:
    Instrument(); // konstruktor bezparametrowy
    ~Instrument(); // destruktor
    void graj(); // metoda graj()

    string nazwa ; // pole nazwa
    string kolor; // pole kolor
    int waga;      // pole waga
};
```

konstruktor bezparametrowy
(domyślny) - buduje obiekt
w pamięci

nazwa konstruktora jest zawsze
taka sama jak nazwa klasy

destruktor - sprząta pamięć, gdy obiekt
nie jest już potrzebny

```
// definicja metody graj()
void Instrument::graj() {
    cout << "pięknie gram" << endl;
}
```

nazwa destruktora zaczyna się od tyldy
i jest zawsze taka sama jak nazwa klasy

```
// definicja konstruktora bezparametrowego
Instrument::Instrument() {
    cout << "konstruktor bezparametrowy - buduję obiekt instrument w pamięci:" << endl;
    waga = 0;
    nazwa = "";
    kolor = "";
    cout << "waga: " << waga << "\nnazwa: " << nazwa << "\nkolor: " << kolor << endl;
}
```

nazwa
kwalifikowa

```
// definicja destruktora
Instrument::~Instrument(){
    cout << "\ndestruktor niszczy obiekt instrument o nazwie " << nazwa << " w pamięci\n" << endl;
}
```

Klasa Instrument z konstruktorem domyślnym i destruktor

obiekt gitara jest
budowany w pamięci
przez konstruktora
domyślnego

```
int main()
{
    cout << "tworzymy obiekt klasy Instrument" << endl;
    Instrument gitara;

    cout << "\nnadajemy nazwę instrumentowi: gitara" << endl;
    gitara.nazwa = "gitara";

    cout << "nazwa instrumentu: " << gitara.nazwa << endl;
}
```

gitara jest obiektem
(instancją) klasy
Instrument – czyli
zajmuje miejsce w
pamięci RAM



Konsola debugowania programu Microsoft Visual Studio

```
tworzymy obiekt klasy Instrument
konstruktor bezparametrowy - buduję obiekt instrument w pamięci:
waga: 0
nazwa:
kolor:

nadajemy nazwę instrumentowi: gitara
nazwa instrumentu: gitara

destruktor niszczy obiekt instrument o nazwie gitara w pamięci
```

Klasa Instrument z konstruktorem parametrowym

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

// definicja klasy instrumentu muzycznego
class Instrument {
public:
    Instrument(string naz = "", string kol = "", int wag = 0): nazwa(naz), kolor(kol), waga(wag) {
        cout << "konstruktor - buduję obiekt instrument w pamięci:" << endl;
        cout << "waga: " << waga << "\nnazwa: " << nazwa << "\nkolor: " << kolor << endl;
    };
    //~Instrument(); // destruktor zostanie i tak utworzony
    void graj();      // metoda graj()

    string nazwa;      // pole nazwa
    string kolor;      // pole kolor
    int waga;          // pole waga
};

// definicja metody graj()
void Instrument::graj() {
    cout << "pięknie gram" << endl;
}

int main()
{
    cout << "tworzymy obiekt klasy Instrument" << endl;
    Instrument instrument; // zostanie wywołany konstruktor
                           // wieloparametrowy z argumentami domyślnymi
    instrument.graj();

    cout << "tworzymy obiekt klasy Instrument o nazwie gitara" << endl;
    Instrument gitara("gitara", "czerwony"); //podajemy argumenty w konstruktorze
    gitara.graj();

    cout << "tworzymy kolejny obiekt klasy Instrument o nazwie perkusja" << endl;
    Instrument perkusja("perkusja");
    perkusja.graj();
}
```

konstruktor wieloparametrowy
z argumentami domyślnymi oraz
listą inicjalizacyjną

Konsola debugowania programu Microsoft Visual Studio

```
tworzymy obiekt klasy Instrument
konstruktor - buduję obiekt instrument w pamięci:
waga: 0
nazwa:
kolor:
pięknie gram
tworzymy obiekt klasy Instrument o nazwie gitara
konstruktor - buduję obiekt instrument w pamięci:
waga: 0
nazwa: gitara
kolor: czerwony
pięknie gram
tworzymy kolejny obiekt klasy Instrument o nazwie perkusja
konstruktor - buduję obiekt instrument w pamięci:
waga: 0
nazwa: perkusja
kolor:
pięknie gram
```

Klasa Instrument z konstruktorem parametrowym

konstruktor wieloparametrowy
z argumentami domyślnymi oraz
listą inicjalizacyjną

argumenty domyślne

lista inicjalizacyjna

```
Instrument(string naz = "", string kol = "", int wag = 0): nazwa(naz), kolor(kol), waga(wag) {  
    cout << "konstruktor - buduję obiekt instrument w pamięci:" << endl;  
    cout << "waga: " << wag << "\nnazwa: " << nazwa << "\nkolor: " << kolor << endl;  
};
```

ciało konstruktora (pomiędzy nawiasami klamrowymi)

Instrument(string naz = "",
gdy nie podamy pierwszego argumentu konstruktora,
zostanie wybrany argument domyślny, czyli pusty ciąg
tekstowy, podobnie z kolejnymi argumentami kol
oraz wag

nazwa(naz), kolor(kol),

zapis ten oznacza: przypisz zmienną naz (pierwszy
argument konstruktora) do pola klasy nazwa,
Podobnie z polami kolor oraz waga

Klasa Instrument - hermetyzacja (enkapsulacja)

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

ukrywanie danych

```
class Instrument {
private:
```

prywatne pola klasy dostępne w zasięgu klasy

```
    string nazwa;
    string kolor;
    int waga;
```

interfejs publiczny klasy
dostępny w klasie i poza nią

```
public:
```

```
    void graj();
    string getNazwa();
    void setNazwa(string naz);
    string getKolor();
    void setKolor(string col);
    int getWaga();
    void setWaga(int wag);
```

getterzy
oraz setterzy
są przejawem
abstrakcji
danych
(ukrywaniem
implementacji)

```
};
```



interfejs telewizora
(public)

telewizor ma części
ukryte przed użytkownikiem
(private)

Klasa Instrument - hermetyzacja (enkapsulacja)

brak bezpośredniego dostępu do prywatnego pola klasy

```
int main()
{
    Instrument gitara;
    gitara.kolor = "zielony";
}
```

 (pole) std::string Instrument::kolor

[Wyszukaj w trybie online](#)

element składowa "Instrument::kolor" (zadeklarowane w wierszu 9) jest niedostępny

[Wyszukaj w trybie online](#)

```
int main()
{
    Instrument gitara;
    gitara.setKolor("zielony");
    cout << gitara.getKolor();
}
```

do prywatnego pola klasy mamy dostęp za pomocą specjalnych metod dostępowych : gettera i settera

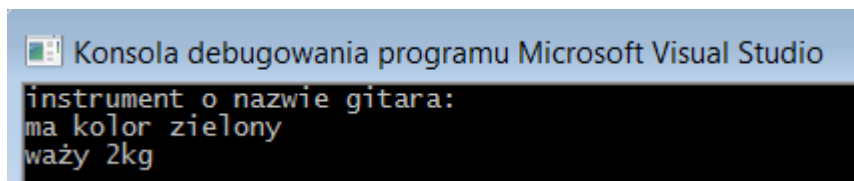
zadanie Klasa Instrument - hermetyzacja (enkapsulacja)

```
void Instrument::graj()
{
    cout << "instrument " << nazwa << " pięknie gra" << endl;
}
```

```
string Instrument::getNazwa()
{
    return nazwa;
}
```

```
void Instrument::setNazwa(string naz)
{
    nazwa = naz;
}
```

dokończ implementację klasy Instrument wykorzystującej hermetyzację



```
Konsola debugowania programu Microsoft Visual Studio
instrument o nazwie gitara:
ma kolor zielony
waży 2kg
```

napisz program
korzystając z
enkapsulacji

Klasa Komputer

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

// klasa reprezentacja komputer
class Komputer {
    // pola prywatne
    string nazwa;
    string systemOperacyjny;
    unsigned speed;
public:
    // konstruktor domyślny
    Komputer() { nazwa = "default"; systemOperacyjny = "default"; speed = 0; };
    // konstruktor wieloparametrowy
    Komputer(string naz, string sys, unsigned sp): nazwa(naz), systemOperacyjny(sys), speed(sp) {};
    void wypiszParametry(); // wypisuje na konsoli parametry komputera
};

void Komputer::wypiszParametry() {
    cout << "Parametry komputera o nazwie " << nazwa << " :\\nsystem: "
         << systemOperacyjny << "\\nszybkość: " << speed << "MHz" << endl;
}
```

Klasa Sala

```
// klasa reprezentująca salę
class Sala {
    // pola prywatne
    string nazwa;           // nazwa sali
    vector<Komputer> komputery; // lista komputerów w sali
public:
    Sala(string naz) : nazwa(naz){}; // konstruktor
    void getParametryKomputerow();    // wypisuje na konsoli parametry komputerów
    void addKomputerDoListy(Komputer komp); // dodaje komputer do listy komputerów
    void podajInformacje();           // wypisuje informacje o sali na konsoli
};

void Sala::podajInformacje() {
    cout << "Informacje dotyczące sali o nazwie " << nazwa << ":\nkomputery:\n";
    getParametryKomputerow();
}

void Sala::addKomputerDoListy(Komputer komp) {
    komputery.push_back(komp);
}

void Sala::getParametryKomputerow() {
    for (Komputer komp : komputery) {
        komp.wypiszParametry();
    }
}
```

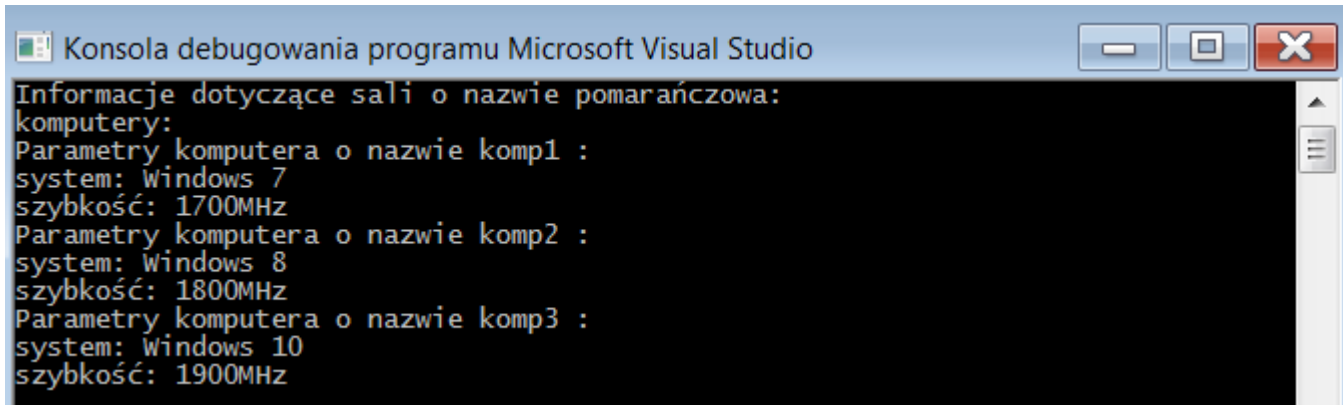
Obiekt pomaranczowa klasy Sala

```
int main()
{
    // tworzymy obiekty klasy komputer
    Komputer komp1("komp1", "Windows 7", 1700);
    Komputer komp2("komp2", "Windows 8", 1800);
    Komputer komp3("komp3", "Windows 10", 1900);

    // tworzymy obiekt sali pomarańczowej
    Sala pomaranczowa("pomarańczowa");

    // dodajemy komputery do sali
    pomaranczowa.addKomputerDoListy(komp1);
    pomaranczowa.addKomputerDoListy(komp2);
    pomaranczowa.addKomputerDoListy(komp3);

    // wypisujemy informacje o sali
    pomaranczowa.podajInformacje();
}
```



Konsola debugowania programu Microsoft Visual Studio

```
Informacje dotyczące sali o nazwie pomarańczowa:
komputery:
Parametry komputera o nazwie komp1 :
system: Windows 7
szybkość: 1700MHz
Parametry komputera o nazwie komp2 :
system: Windows 8
szybkość: 1800MHz
Parametry komputera o nazwie komp3 :
system: Windows 10
szybkość: 1900MHz
```

Konstruktor kopiujący

Jeśli programista go nie zdefiniuje zostanie wygenerowany automatycznie (konstruktor kopiuje wtedy klasę składnik po składniku (kopiuje wskaźniki jako adresy, a nie to na co wskazują – dlatego wskaźniki w kopii i oryginale będą wskazywać na to samo, a to nie jest porządane) – powstaje dokładna kopia obiektu - Shallow Copy),

Lecz jeśli mamy pola w klasie jako wskaźniki, trzeba ten konstruktor napisać samemu (Deep Copy)

tworzy obiekt, który stanowi kopię istniejącego obiektu

konstruktor kopiujący

Postać:

```
Uczen(const Uczen& u){}
```

argumentem jest referencja
do obiektu tej samej klasy
const gwarantuje, że konstruktor
nie zmieni przekazywanego obiektu

Wywoływanie (nie jest gwarantowane, zależy od kompilatora):

- w przypadku tworzenia nowego obiektu na podstawie innego obiektu:
Uczen uczen2 = uczen1;
- kiedy obiekt klasy jest zwracany przez wartość
- kiedy obiekt klasy jest argumentem przekazywanym do funkcji przez wartość
- kiedy kompilator generuje obiekt tymczasowy

Konstruktor kopiujący wygenerowany automatycznie

```
class Uczeń {
private:
    string imie;  // typ wbudowany string
public:
    // konstruktor
    Uczeń(string nazwa) : imie(nazwa) {};
    string getImie() {
        return imie;
    }
    void setImie(string nazwa) {
        imie = nazwa;
    }
};

int main() {

    Uczeń uczen1("Marek");
    cout << uczen1.getImie() << endl;  // Marek

    // tworzymy nowego ucznia kopiując istniejącego ucznia
    // działa konstruktor kopiujący wygenerowany automatycznie
    Uczeń uczen2 = uczen1;

    cout << uczen2.getImie() << endl;  // Marek
    // zmieniamy imie 1 ucznia
    uczen1.setImie("Jarek");
    cout << uczen1.getImie() << endl;  // Jarek
    // imię 2 ucznia nie zmieniło się - brak problemu
    cout << uczen2.getImie() << endl;  // Marek

    return 0;
}
```

Shallow Copy



uczen1
imie: Marek

uczen2
imie: Marek

uczen1
imie: Jarek

uczen2
imie: Marek



Konstruktor kopiujący
wygenerowany automatycznie
wystarcza

Konstruktor kopiujący wygenerowany automatycznie

```
class Ucen {
private:
    string* imie;
public:
    // konstruktor
    Ucen(string nazwa) {
        imie = new string(nazwa);
    };
    // destruktor
    ~Ucen() {
        delete imie;
    }
    string getImie() {
        return *imie;
    }
    void setImie(string nazwa) {
        *imie = nazwa;
    }
};
```

wskaźnik typu string

wartość wskaźnika imie w obiektach uczen1 oraz uczen2 jest taka sama (wskaźniki pokazują na to samo)

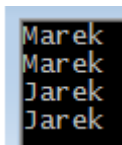
Shallow Copy

uczen1		uczen2
imie ->	Marek	<- imie
uczen1		uczen2
imie ->	Jarek	<- imie

```
int main() {
    Ucen uczen1("Marek");
    cout << uczen1.getImie() << endl; // Marek

    // działa konstruktor kopiujący wygenerowany automatycznie
    Ucen uczen2 = uczen1;
    cout << uczen2.getImie() << endl; // Marek
    uczen1.setImie("Jarek");
    cout << uczen1.getImie() << endl; // Jarek
    // imię 2 ucznia zmieniło się - mamy problem
    cout << uczen2.getImie() << endl; // Jarek!

    return 0;
}
```



Konstruktor kopiujący wygenerowany automatycznie nie wystarcza, trzeba napisać własny konstruktor kopiujący

uczen1	{imie=0x003ac968 "Jarek" }
uczen2	{imie=0x003ac968 "Jarek" }

Konstruktor kopiujący wygenerowany automatycznie

```
class Uczeń {  
private:  
    string* imie;  
public:  
    // konstruktor  
    Uczeń(string nazwa) {  
        imie = new string(nazwa);  
    };  
    // destruktor  
    ~Uczeń() {  
        delete imie;  
    }  
    string getImie() {  
        return *imie;  
    }  
    void setImie(string nazwa) {  
        *imie = nazwa;  
    }  
};
```

wskaznik typu string



```
= _Pnext->_Mynextiter) { // TRANSITION, VSO-1269037
```

Zgłoszony wyjątek

Zgłoszono wyjątek: naruszenie dostępu do odczytu.
_Pnext było 0xDDDDDE1.

Destruktor usuwa 2x ten sam wskaźnik

```
int main() {  
  
    Uczeń uczen1("Marek");  
    cout << uczen1.getImie() << endl; // Marek  
  
    // działa konstruktor kopiujący wygenerowany automatycznie  
    Uczeń uczen2 = uczen1;  
    cout << uczen2.getImie() << endl; // Marek  
    uczen1.setImie("Jarek");  
    cout << uczen1.getImie() << endl; // Jarek  
    // imię 2 ucznia zmieniło się - mamy problem  
    cout << uczen2.getImie() << endl; // Jarek!  
  
    return 0;  
}
```

uczen1	{imie=0x003ac968 "Jarek" }
uczen2	{imie=0x003ac968 "Jarek" }

Konstruktor kopiujący własny

Deep Copy



```
class Uczeń {  
private:  
    string* imie;  
public:  
    // konstruktor  
    Uczeń(string nazwa) {  
        imie = new string(nazwa);  
    };  
    // konstruktor kopiujący  
    Uczeń(const Uczeń& u) {  
        imie = new string(*(u.imie));  
    }  
    // destruktor  
    ~Uczeń() {  
        delete imie;  
    }  
  
    string getImie() {  
        return *imie;  
    }  
    void setImie(string nazwa) {  
        *imie = nazwa;  
    }  
};
```

wskaźnik typu string

wartość wskaźnika imie w obiektach
uczen1 oraz uczen2 jest inna
(każdy wskaźnik pokazuje na swój obiekt typu string)

napisany
konstruktor
kopiujący

nie ma problemu
z destrukтором

uczen1

imie -> Marek

uczen2

imie -> Marek

uczen1

imie -> Jarek

uczen2

imie -> Marek



```
int main() {  
  
    Uczeń uczen1("Marek");  
    cout << uczen1.getImie() << endl; // Marek  
  
    // działa konstruktor kopiujący napisany przez programistę (Deep Copy)  
    Uczeń uczen2 = uczen1;  
    cout << uczen2.getImie() << endl; // Marek  
    uczen1.setImie("Jarek");  
    cout << uczen1.getImie() << endl; // Jarek  
    // imię 2 ucznia nie zmieniło się  
    cout << uczen2.getImie() << endl; // Marek  
  
    return 0;  
}
```

W przypadku, gdy pola klasy są wskaźnikami
konstruktor kopiujący musi zostać
napisany przez programistę

Pola i metody statyczne klasy

definicja klasy Cat

```
class Cat {  
private:  
    // jeśli pole statyczne jest stałą typu całkowitego, znakowego, wyliczeniowego lub logicznego  
    // można je zainicjalizować w ciele definicji klasy  
    const static int max = 20;  
public:  
    // instance variables - zmienna instancyjna, wymaga istnienia obiektu klasy  
    string name;  
    // pole statyczne klasy, wspólne dla wszystkich obiektów klasy Cat  
    // (class variables - zmienna klasy, nie wymaga istnienia obiektu klasy)  
    // inicjalizacja musi być przeprowadzona poza definicją klasy  
    // można się do niego odwoływać poprzez obiekt lub klasę  
    static int counter;  
    // metoda statyczna może być zainicjowana wewnątrz definicji klasy,  
    // metody statyczne nie mają dostępu do niestatycznych metod i zmiennych,  
    // można się do niej odwoływać poprzez obiekt lub klasę  
    static int getMax() { return max; };  
    // konstruktor, w którym inkrementujemy counter  
    // w ten sposób będziemy zliczać powstające obiekty klasy Cat  
    Cat(string i = "") :name(i) {  
        counter++;  
    };  
};
```

stałe pole statyczne max

pole statyczne
counter:

- wspólne dla wszystkich obiektów
- odwołujemy się najczęściej poprzez nazwę klasy

Jeśli metoda byłaby niestatkyczna,
nie mogłaby korzystać
ze zmiennych statycznych

```
// inicjalizacja pola statycznego musi się odbyć na zewnątrz definicji klasy  
// w zakresie globalnym  
int Cat::counter = 0;
```

inicjalizacja pola statycznego klasy Cat

Pola i metody statyczne klasy

```
int main()
{
    std::cout << "My cats:\n\n";

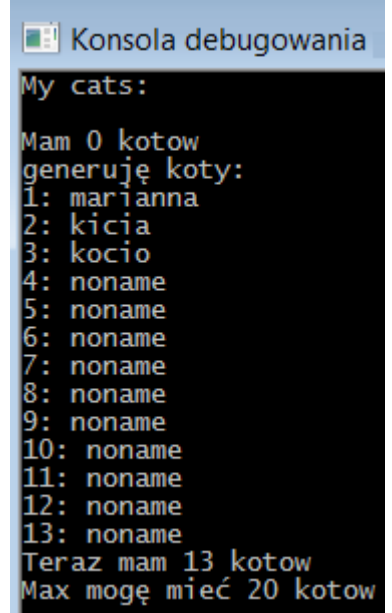
    // do pola statycznego możemy się dostać nawet wtedy,
    // gdy nie ma jeszcze utworzonych obiektów (poprzez nazwę klasy)
    cout << "Mam " << Cat::counter << " kotow" << endl;

    cout << "generuję koty: " << endl;
    // tworzymy obiekty klasy Cat
    Cat cat1("marianna");
    // do pola statycznego możemy się również dostać poprzez obiekt klasy
    cout << cat1.counter << ": " << cat1.name << endl;
    Cat cat2("kicia");
    cout << Cat::counter << ": " << cat2.name << endl;
    Cat cat3("kocio");
    cout << cat3.counter << ": " << cat3.name << endl;

    for (int i = 0; i < 10; i++)
    {
        Cat cat("noname");
        cout << cat.counter << ": " << cat.name << endl;
    }

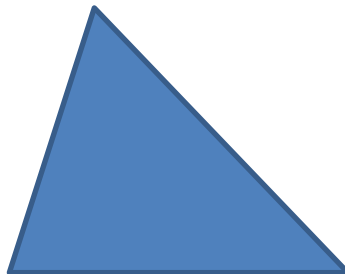
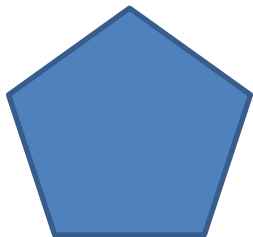
    cout << "Teraz mam " << Cat::counter << " kotow" << endl;
    // odwołanie do metody statycznej
    cout << "Max mogę mieć " << Cat::getMax() << " kotow" << endl;
}
```

zliczamy ilość obiektów klasy Cat
za pomocą pola statycznego counter

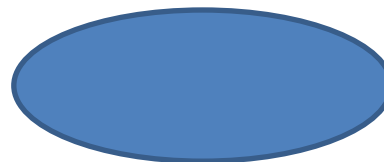
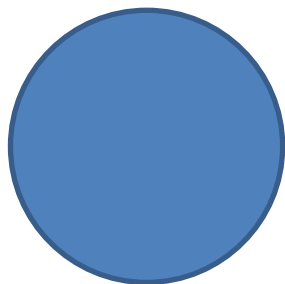


```
Konsola debugowania
My cats:
Mam 0 kotow
generuję koty:
1: marianna
2: kicia
3: kocio
4: noname
5: noname
6: noname
7: noname
8: noname
9: noname
10: noname
11: noname
12: noname
13: noname
Teraz mam 13 kotow
Max mogę mieć 20 kotow
```

Dziedziczenie



Jakie są wspólne cechy figur?



Dziedziczenie

Klasa bazowa Figura

wspólne cechy figur

```
#include <iostream>
#include<corecrt_math_defines.h>    //M_PI
using namespace std;

class Figura {
public:
    string nazwa;
    double pole;
    double obwod;
    // konstruktor z wartościami domyślnymi oraz listą inicjalizacyjną
    Figura(string n, double p, double o) : nazwa(n), pole(p), obwod(o) {};
    void toString() { cout << "figura:\t"<< nazwa << endl << " pole:\t" << pole << endl << " obwod:\t" << obwod << endl; };
};
```

wspólna metoda toString()

Dziedziczenie

Klasa pochodna Kwadrat

```
class Kwadrat : public Figura {
public:
    double bok;
    // konstruktor z wartościami domyslnymi oraz listą inicjalizacyjną,
    // tworzymy tu również obiekt klasy Figura (wywołujemy konstruktor klasy Figura)
    Kwadrat(string n, double b, double p = 0.0, double o = 0.0) : Figura(n, p, o), bok(b) {
        pole = bok * bok;
        obwod = 4 * bok;
    };
};
```

Klasa pochodna Kolo

```
class Kolo : public Figura {
public:
    double promien;
    // konstruktor z wartościami domyslnymi oraz listą inicjalizacyjną,
    // tworzymy tu również obiekt klasy Figura (wywołujemy konstruktor klasy Figura)
    Kolo(string n, double pr, double p = 0.0, double o = 0.0) : Figura(n, p, o), promien(pr) {
        pole = M_PI * promien * promien;
        obwod = 2 * M_PI * promien;
    };
};
```

Dziedziczenie

```
int main()
{
    Kwadrat kwadrat("kwadrat",5);
    kwadrat.toString();
    Kolo kolo("kolo", 1);
    kolo.toString();
}
```



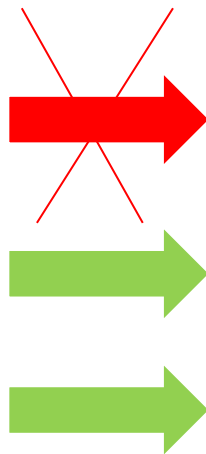
```
Konsola debugowania programu Microsoft Visual Studio
figura: kwadrat
pole: 25
obwod: 20
figura: kolo
pole: 3.14159
obwod: 6.28319
```

Dziedziczenie public

Dziedziczenie **public** :

- składowe prywatne z klasy bazowej nie są widoczne w klasie pochodnej;
- składowe protected (chronione) stają się protected w klasie dziedziczącej
- składowe publiczne w klasie bazowej stają się publiczne w klasie pochodnej

```
class A {  
private:  
    int x;  
protected:  
    int y;  
public:  
    int z;  
};
```



```
class B : public A {  
private:  
    int k;  
protected:  
    int m;  
public:  
    int n;  
};
```

konstruktory i destruktor nie są dziedziczone

Dziedziczenie public

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

```
class A {
private:
    int x;
protected:
    int y;
public:
    int z;
};
```



```
class B : public A {
public:
    int getY() { return y; };
protected:
    int y;
public:
    int z;
};
```

😊 OK



```
class C : public B {
public:
    int getZ() { return z; };
protected:
    int y;
public:
    int z;
};
```

😊 OK

```
int main() {
    A a;
    B b;
    b.z = 5;
}
```

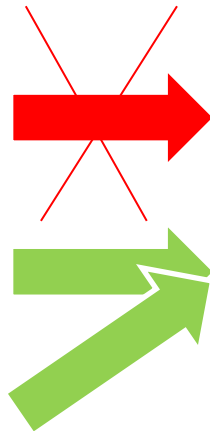
😊 OK

Dziedziczenie protected

Dziedziczenie **protected** :

- składowe prywatne z klasy bazowej nie są widoczne w klasie pochodnej;
- składowe protected (chronione) stają się protected w klasie dziedziczącej
- składowe publiczne w klasie bazowej stają się protected w klasie pochodnej

```
class A {  
private:  
    int x;  
protected:  
    int y;  
public:  
    int z;  
};
```



```
class B : protected A {  
private:  
    int k;  
protected:  
    int m;  
public:  
    int n;  
};
```

konstruktory i destruktor nie są dziedziczone

Dziedziczenie protected

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

```
class A {
private:
    int x;
protected:
    int y;
public:
    int z;
};
```



```
class B : protected A {
public:
    int getY() { return y; };
protected:
    int y;
protected:
    int z;
};
```

😊 OK



```
class C : public B {
public:
    int getZ() { return z; };
protected:
    int y;
protected:
    int z;
};
```

😊 OK

```
int main() {
    A a;
    B b;
    b.z = 5;
}
```

☹ NOK

🔍 (pole) `int A::z`

[Wyszukaj w trybie online](#)

element składowa "A::z" (zadeklarowane w wierszu 11) jest niedostępny

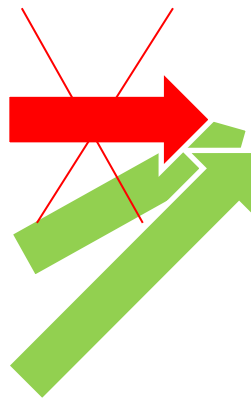
[Wyszukaj w trybie online](#)

Dziedziczenie private

Dziedziczenie **private** :

- składowe prywatne z klasy bazowej nie są widoczne w klasie pochodnej;
- składowe protected (chronione) stają się private w klasie dziedziczącej
- składowe publiczne w klasie bazowej stają się private klasie pochodnej

```
class A {  
private:  
    int x;  
protected:  
    int y;  
public:  
    int z;  
};
```



```
class B : private A {  
private:  
    int k;  
protected:  
    int m;  
public:  
    int n;  
};
```

konstruktory i destruktor nie są dziedziczone

Dziedziczenie private

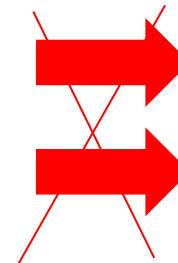
```
#include <iostream>
using namespace std;
```

```
class A {
private:
    int x;
protected:
    int y;
public:
    int z;
};
```



```
class B : private A {
public:
    int getY() { return y; };
private:
    int y;
private:
    int z;
};
```

😊 OK



```
class C : public B {
public:
    int getZ() { return z; };
};
```

☹ NOK

```
int main() {
    A a;
    B b;
    b.z = 5;
}
```

☹ NOK

🔍 (pole) `int A::z`

[Wyszukaj w trybie online](#)

element składowa "A::z" (zadeklarowane w wierszu 11) jest niedostępny

[Wyszukaj w trybie online](#)

Dziedziczenie public

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

class A {
private:
    int x;    // składowa z dostępem wewnątrz klasy
protected:
    int y;    // składowa z dostępem wewnątrz klasy
              // oraz z klas dziedziczących
public:
    // dostęp do składowych klasy z wnętrza klasy
    int getX() { return x; }; // OK
    // int getY() { return y; }; // OK
};

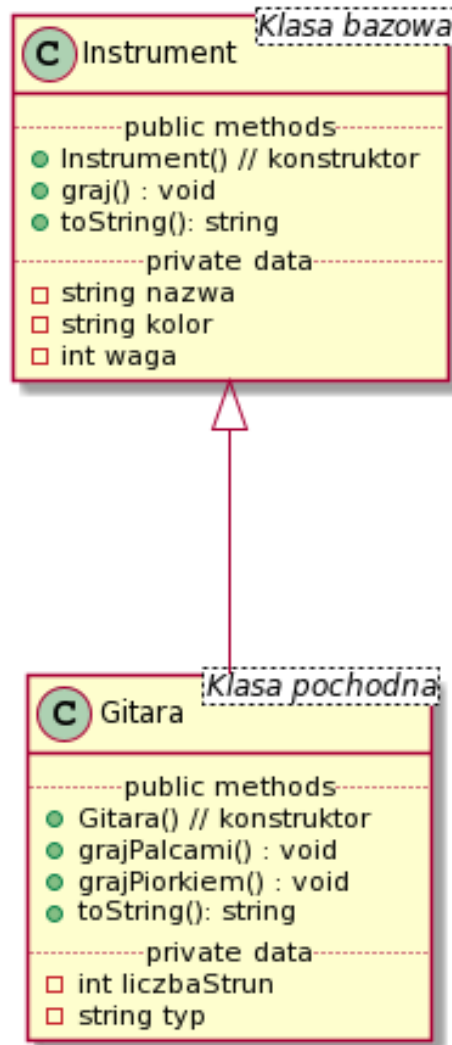
// klasa dziedzicząca publicznie po A
class B : public A {
public:
    // dostęp do składowych klasy z wnętrza klasy
    int getY() { return y; }; // OK
    int setX(int s) { x = s; } // NOK A::x jest niedostępne
};

class C {
public:
    int m;
};

int main()
{
    A a;
    B b;
    C c;
    // dostęp do składowych
    // klasy z zewnątrz klasy
    a.x = 5;           // NOK
    a.y = 10;          // NOK
    c.m = a.y;         // NOK
    int d = a.getX(); // OK
    b.getY();          // OK
}
```

Dziedziczenie

Diagram UML



class inheritance chain

łańcuch dziedziczenia

klasa Gitara dziedziczy
z klasy Instrument

zadanie

Dziedziczenie

dokończ implementację
klas Instrument oraz Gitara

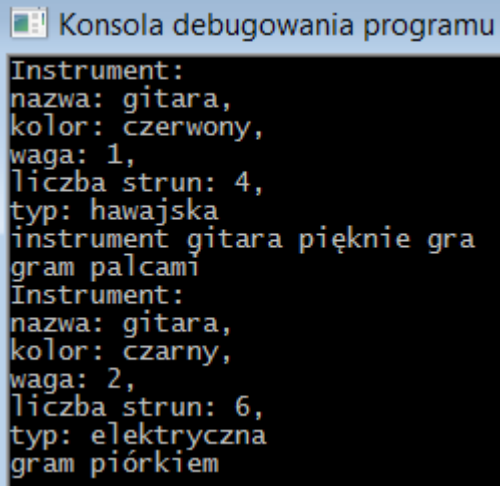
```
// klasa bazowa Instrument
class Instrument {
private:
    string nazwa;
    string kolor;
    int waga;
public:
    // konstruktor parametrowy z argumentami domyślnymi i listą inicjalizacyjną
    Instrument(string naz = "", //argumentem domyślnym naz jest pusty string
               string col = "",
               int wag = 0):
        nazwa(naz),
        kolor(col),
        waga(wag) {};

    // metoda opisująca prywatne pola instrumentu
    string toString();
    void graj();
};

// klasa pochodna Gitara dziedziczy publicznie z klasy bazowej Instrument
class Gitara: public Instrument{
// prywatne pola klasy pochodnej
private:
    int liczbaStrun;
    string typ;
public:
    // publiczne metody klasy pochodnej
    // konstruktor parametrowy z argumentami domyślnymi i listą inicjalizacyjną
    Gitara(string naz = "",
           string col = "",
           int wag = 0,
           int liczStr = 0,
           string ty = ""):
        Instrument(naz, col, wag), // wywołanie konstruktora klasy bazowej
        liczbaStrun(liczStr),
        typ(ty){};

    void grajPalcami();
    void grajPiorkiem();
    string toString();
};
```

napisz program



Konsola debugowania programu

```
Instrument:
nazwa: gitara,
kolor: czerwony,
waga: 1,
liczba strun: 4,
typ: hawajska
instrument gitara pięknie gra
gram palcami
Instrument:
nazwa: gitara,
kolor: czarny,
waga: 2,
liczba strun: 6,
typ: elektryczna
gram piórkiem
```


Polimorfizm *Polymorphism*

wielopostaciowość metod klasy, które mogą przybierać wiele form



Polimorfizm statyczny – przeciążanie metod

methods overloading

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

```
class Sumator {
public:
    void suma(int, int);
    void suma(double, double);
};
```

metody przeciążone mają taką samą nazwę
lecz różnią się sygnaturą (do której zaliczamy ilość
i typ parametrów).

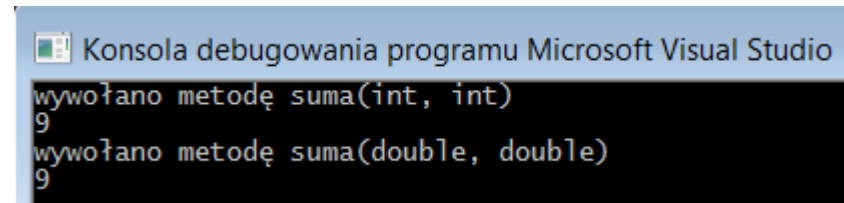
suma(int, int)

```
int main()
{
    Sumator sumator;
    sumator.suma(5, 4);
    sumator.suma(5.0, 4.0);
}
```

Którą metodę wywoła kompilator?

```
void Sumator::suma(int a, int b)
{
    cout << "wywołano metodę suma(int, int)" << endl;
    cout << a + b << endl;
}

void Sumator::suma(double a, double b)
{
    cout << "wywołano metodę suma(double, double)" << endl;
    cout << a + b << endl;
}
```



Konsola debugowania programu Microsoft Visual Studio

```
wywołano metodę suma(int, int)
9
wywołano metodę suma(double, double)
9
```

Kompilator dopasowuje
argumenty wywołania
metody z jej parametrami
formalnymi w definicji.
Wybiera tę metodę, która
najlepiej pasuje.
Dzieje się to w trakcie
kompilacji programu
(wczesne wiązanie).

Polimorfizm statyczny – przeciążanie metod methods overloading

Te metody nie są przeciążone

```
class Sumator {  
public:  
    void suma(int, int);  
    int suma(int, int);  
};
```

❖ `int Sumator::suma(int, int)`

[Wyszukaj w trybie online](#)

nie można przeciążyć funkcji różniących się tylko typem zwracanej wartości

[Wyszukaj w trybie online](#)

wartość zwracana nie należy do
sygnatury funkcji

Polimorfizm statyczny – przeciążanie metod methods overloading

pobaw się we kompilator ☺

```
class Sumator {
public:
    void suma(int a, double b);           // 1
    void suma(long a, int b);             // 2
    void suma(char a, int b, int c);       // 3
    void suma(int a, string b, int c);     // 4
    void suma(const char* s, bool b, int c); // 5
};

int main()
{
    Sumator s;
    s.suma("suma liczb wynosi", true, 23); // wywołanie 1
    s.suma(12L, 23);                        // wywołanie 2
    s.suma('1', 2, 3);                      // wywołanie 3
    s.suma(1, "suma", 3);                   // wywołanie 4
    s.suma(123, 345.98);                   // wywołanie 5
}
```

dopasuj wywołanie metody do jej deklaracji

Polimorfizm statyczny – przesłanianie metod

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

```
class Tata {
public:
    void przedstawSie() {
        cout << "Cześć, jestem tata :)" << endl;
    }
};
```

metoda przedstawSie() w klasie bazowej jest przesłonięta przez metodę o tej samej nazwie zdefiniowaną w klasie pochodnej.

```
class Syn : public Tata {
public:
    void przedstawSie() {
        cout << "Cześć, jestem synek :)" << endl;
    }
};
```

klasa Syn dziedziczy publicznie z klasy Tata

metoda przedstawSie() w klasie pochodnej przesłania metodę o tej samej nazwie zdefiniowaną w klasie bazowej.

```
int main()
{
    Tata tata;
    tata.przedstawSie();

    Syn syn;
    syn.przedstawSie();
}
```

kompilator wybiera metodę z klasy bazowej



```
Konsola debugowania pro
Cześć, jestem tata :)
Cześć, jestem synek :)
```

kompilator wybiera metodę z klasy pochodnej

metoda przedstawSie() w klasie pochodnej ma taki sam typ zwracany oraz sygnaturę jak metoda o tej samej nazwie zdefiniowana w klasie bazowej

Dopasowanie wywołania metody do kodu, który ma być wykonany realizowane jest w trakcie kompilacji programu (wczesne wiązanie).

Polimorfizm statyczny – przesłanianie metod

metoda `przedstawSie()`
nie jest zadeklarowana
w klasie bazowej jako
wirtualna

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

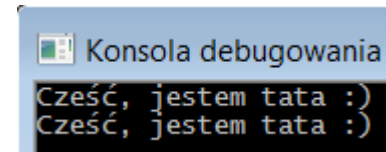
class Tata {
public:
    void przedstawSie() {
        cout << "Cześć, jestem tata :)" << endl;
    }
};

class Syn : public Tata {
public:
    void przedstawSie() {
        cout << "Cześć, jestem synek :)" << endl;
    }
};

int main()
{
    Syn syn;
    Tata tata;
    Tata* tata2 = &tata;
    tata2->przedstawSie();
    tata2 = &syn;
    tata2->przedstawSie();
}
```

dostęp do metody za pomocą wskaźnika

wskaźnik `tata2` może być
adresem syna, jednak kompilator
przy wywołaniu metody
`przedstawSie()` uwzględnia
typ wskaźnika `Tata*`
(a nie obiektu, na jaki wskazuje)
– wiązanie statyczne



Konsola debugowania

```
Cześć, jestem tata :)
Cześć, jestem tata :)
```

Polimorfizm dynamiczny

metoda `przedstawSie()`
jest zadeklarowana
w klasie bazowej jako
wirtualna

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

class Tata {
public:
    virtual void przedstawSie() {
        cout << "Cześć, jestem tata :)" << endl;
    }
};

class Syn : public Tata {
public:
    void przedstawSie() {
        cout << "Cześć, jestem synek :)" << endl;
    }
};

int main()
{
    Syn syn;
    Tata tata;
    Tata* tata2 = &tata;
    tata2->przedstawSie();
    tata2 = &syn;
    tata2->przedstawSie();
}
```

wskaźnik `tata2` jest
adresem syna, ponieważ
metoda `przedstawSie()` jest
zadeklarowana w klasie bazowej
jako wirtualna, kompilator
przy wywołaniu metody
`przedstawSie()` uwzględnia
typ obiektu, na jaki wskazuje
wskaźnik – w tym przypadku
na syna (a nie typ wskaźnika)
– wiązanie dynamiczne



Konsola debugowania

```
Cześć, jestem tata :)
Cześć, jestem synek :)
```

Polimorfizm dynamiczny

Aby skorzystać z polimorfizmu dynamicznego należy:

- W klasie bazowej zadeklarować metodę ze słowem kluczowym `virtual`, (definiujemy jako wirtualne metody, które wymagają innej definicji w klasach pochodnych).
Metoda będzie wirtualna w całym drzewie dziedziczenia klasy bazowej
- W klasach pochodnych należy zdefiniować metody o takiej samej nazwie, sygnaturze i typie zwracanym co metoda wirtualna.
- Metoda wirtualna wywołana za pomocą wskaźnika lub referencji uruchamia kod właściwy dla typu wskazywanego obiektu (a nie dla typu referencji lub wskaźnika).

Abstrakcja

abstrakcja

ukrywanie przed użytkownikiem
szczegółów implementacji klas,
udostępnienie użytkownikom funkcjonalności

getter, setter.
specyfikatory dostępu
public, private, protected,
a także podział programu na
pliki nagłówkowe i pliki z
kodem źródłowym są
przejawem abstrakcji
danych(ukrywaniem
implementacji)

Java, C# ,C++

interface
interfejs

- zawiera m.in. tylko deklaracje metod abstrakcyjnych
 - nie zawiera zwykłych metod i pól
- W C++ można go zaimplementować jako klasę abstrakcyjną zawierającą tylko metody czysto wirtualne (virtual ... = 0).
Dzięki wielodziedziczeniu dana klasa może dziedziczyć również taki „interfejs”

W Javie i C# dana klasa może dziedziczyć po wielu interfejsach i tylko jednej klasie bazowej, w C++ dana klasa może dziedziczyć po wielu klasach (wielodziedziczenie).

abstract class

klasa abstrakcyjna

- klasa bazowa, która posiada deklarację co najmniej jednej metody abstrakcyjnej (metody czysto wirtualnej virtual ... = 0), metoda abstrakcyjna posiada definicję w klasie pochodnej
- może posiadać zwykłe pola i metody niewirtualne
- nie można utworzyć obiektu klasy abstrakcyjnej (klasa abstrakcyjna jest bytem abstrakcyjnym, bez „ciała”)

abstract method

Klasa abstrakcyjna

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

```
// klasa abstrakcyjna
```

```
class Zagadka {
public:
```

```
    // metoda czysto wirtualna (pure virtual)
    virtual void zgadnij() = 0;
};
```

```
// klasa abstrakcyjna
```

```
// brak definicji metody wirtualnej
```

```
class ZagadkaDlaDzieci : public Zagadka {
};
```

```
class ZagadkaDlaDzieciMatematyczna : public ZagadkaDlaDzieci {
public:
    // definicja metody wirtualnej
    void zgadnij() { cout << "2*2 = ?" << endl; };
};
```

metoda abstrakcyjna - czysto wirtualna (pure virtual)
jest to deklaracja metody, bez definicji,
definicja powinna być określona w klasie pochodnej
(jeśli nie jest - wtedy klasa pochodna też jest
abstrakcyjna)

klasy pochodne

klasa abstrakcyjna
(zawiera co najmniej jedną
metodę abstrakcyjną –
czysto wirtualną),
może zawierać również
zwykłe pola i metody
niewirtualne

nie można tworzyć obiektu klasy
abstrakcyjnej

```
int main()
{
    Zagadka z;
}
```

(zmienna lokalna) Zagadka z

[Wyszukaj w trybie online](#)

obiekt typu klasy abstrakcyjnej "Zagadka" jest niedozwolony:
funkcja "Zagadka::zgadnij" to czysta funkcja wirtualna

[Wyszukaj w trybie online](#)

```
int main()
{
    ZagadkaDlaDzieci z;
}
```

(zmienna lokalna) ZagadkaDlaDzieci z

[Wyszukaj w trybie online](#)

obiekt typu klasy abstrakcyjnej "ZagadkaDlaDzieci" jest niedozwolony:
czysty element wirtualny funkcja "Zagadka::zgadnij" nie ma elementu przesłaniającego

[Wyszukaj w trybie online](#)

Klasa abstrakcyjna

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

// klasa abstrakcyjna
class Zagadka {
public:
    // metoda czysto wirtualna (pure virtual)
    virtual void zgadnij() = 0;
};

// klasa abstrakcyjna
// brak definicji metody wirtualnej
class ZagadkaDlaDzieci : public Zagadka {
};

class ZagadkaDlaDzieciMatematyczna : public ZagadkaDlaDzieci {
public:
    // definicja metody wirtualnej
    void zgadnij() { cout << "2*2 = ?" << endl; };
};
```

korzystamy z polimorfizmu dynamicznego: ponieważ metoda `zgadnij()` jest zadeklarowana w klasie bazowej jako wirtualna, kompilator przy wywołaniu metody `zgadnij()` uwzględnia typ obiektu, na jaki wskazuje wskaźnik – w tym przypadku na obiekt klasy `ZagadkaDlaDzieciMatematyczna` (a nie typ wskaźnika) – wiązanie dynamiczne

```
int main()
{
    ZagadkaDlaDzieciMatematyczna zm;
    zm.zgadnij();
}
```

```
int main()
{
    ZagadkaDlaDzieciMatematyczna zm;
    Zagadka* z;
    z = &zm;
    z->zgadnij();
}
```



Konsola
2*2 = ?

wywołanie metody `zgadnij()` na rzecz obiektu `zm` klasy `ZagadkaDlaDzieciMatematyczna`

Klasa abstrakcyjna - interfejs

implementujemy interfejs

```
#include <iostream>
#include <vector>

using namespace std;

// klasa bazowa z metodą czysto wirtualną
// (interfejs - są tylko metody czysto wirtualne, brak pól,
// mogą występować komponenty statyczne)
class Zwierze {
public:
    virtual void przedstawSie() = 0; //metoda czysto wirtualna (pure virtual)
};

// klasa z polami
class Nazwa {
public:
    string imie;
    Nazwa(string im) :imie(im) {}; // konstruktor parametrowy
};

// klasa pochodna z własną definicją metody wirtualnej
class Lew : public Zwierze, public Nazwa {
public:
    Lew(string im) :Nazwa(im) {}; // konstruktor parametrowy
    // definicja metody wirtualnej
    void przedstawSie() { cout << "Jestem lwem o imieniu " << imie << endl; };
};

// klasa pochodna z własną definicją metody wirtualnej
class Chomik : public Zwierze, public Nazwa {
public:
    Chomik(string im) :Nazwa(im) {};
    void przedstawSie() { cout << "Jestem chomikiem o imieniu " << imie << endl; };
};
```

dzięki hierarchii dziedziczenia
oraz słowu kluczowemu virtual
uzyskujemy właściwość polimorfizmu
dynamicznego (późne wiązanie)

wielodziedziczenie – klasa Lew
dziedziczy z interfejsu metodę
przedstawSie(), a z klasy Nazwa pole
imie.

wielodziedziczenie – klasa Chomik
dziedziczy z interfejsu metodę
przedstawSie(), a z klasy Nazwa pole
imie.

Klasa abstrakcyjna - interfejs

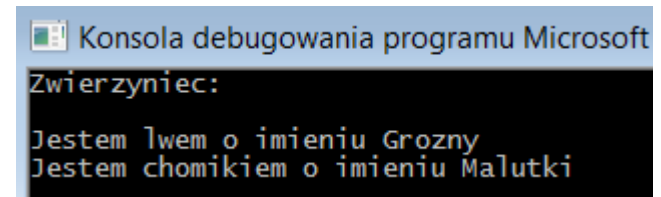
```
int main()
{
    cout << "Zwierzyniec:\n\n";
    // tworzymy instancje zwierząt na stercie
    Zwierze* lew = new Lew("Grozny");
    Zwierze* chomik = new Chomik("Malutki");

    // tablica ze wskaźnikami na kolejne zwierzęta
    vector<Zwierze*> zoo;
    zoo.push_back(lew);
    zoo.push_back(chomik);

    // petla po wszystkich wskaźnikach na zwierzęta
    for (auto z : zoo) {
        z->przedstawSie();
    }

    // sprzątamy pamięć
    delete lew;
    delete chomik;
    lew = nullptr;
    chomik = nullptr;
}
```

dzięki hierarchii dziedziczenia
oraz słowu kluczowemu virtual
uzyskujemy właściwość polimorfizmu
dynamicznego (późne wiązanie)



Konsola debugowania programu Microsoft

Zwierzyniec:

Jestem lwem o imieniu Grozny
Jestem chomikiem o imieniu Malutki

Funkcja zaprzyjaźniona z klasą

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

class Portfel {
private:
    int kasa;
public:
    // konstruktor domyślny
    Portfel():kasa(1000) {};
    // deklaracja funkcji zaprzyjaźnionej z klasą Portfel
    friend void przyjaciel(Portfel& portfel);
    void pokazKase() { cout << kasa; };
};
```

słowo
kluczowe
friend

funkcja zaprzyjaźniona jest
zwykłą funkcją

```
int main()
{
    cout << "Pożyczka dla przyjaciela\n\n";
    Portfel portfel;
    przyjaciel(portfel);
    cout << "w portfelu zostało: ";
    portfel.pokazKase();
}
```

```
void przyjaciel(Portfel& p)
{
    cout << "masz w portfelu : " << p.kasa << endl;
    cout << "pożyczam 100 zł" << endl;
    p.kasa -= 100;
    cout << "dzięki!" << endl;
}
```

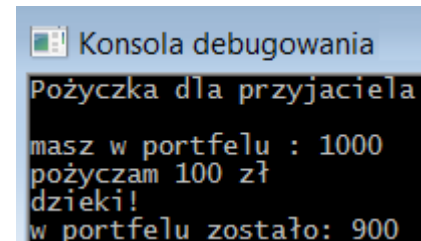
funkcja zaprzyjaźniona operuje na
prywatnym polu klasy Portfel

funkcja zaprzyjaźniona z klasą Portfel:

- ma dostęp do prywatnych i chronionych członków klasy,
- nie jest członkiem klasy (nie wywołuje się jej na rzecz obiektu)
- deklaracja może być umieszczona w dowolnym miejscu klasy (nawet private),
- parametrem jest obiekt klasy Portfel przekazywany do funkcji przez referencję (wewnątrz funkcji zmieniamy stan obiektu),
- funkcja zaprzyjaźniona może być przyjacielem kilku klas jednocześnie
- rozszerza mechanizm hermetyzacji (ukrywania danych)

wywołanie funkcji zaprzyjaźnionej

definicja funkcji zaprzyjaźnionej z klasą Portfel



Konsola debugowania

```
Pożyczka dla przyjaciela
masz w portfelu : 1000
pożyczam 100 zł
dzięki!
w portfelu zostało: 900
```

Funkcja (metoda) zaprzyjaźniona z klasą

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

```
// deklaracja klasy Portfel
class Portfel;
```

```
class Przyjaciel {
public:
    void pozyczKase(Portfel& portfel);
};
```

```
class Portfel {
private:
    int kasa;
public:
    // konstruktor domyślny
    Portfel():kasa(1000) {};
    // deklaracja funkcji zaprzyjaźnionej z klasą Portfel
    // (to nie jest metoda klasy Portfel)
    friend void Przyjaciel::pozyczKase(Portfel& portfel);
    void pokazKase() { cout << kasa; };
};
```

ważna jest kolejność klas!

funkcja zaprzyjaźniona jest metodą klasy Przyjaciel, ma dostęp do prywatnych i chronionych członków klasy Portfel

```
int main()
{
    cout << "Pożyczka dla przyjaciela\n\n";
    Portfel portfel;
    Przyjaciel przyjaciel;
    przyjaciel.pozyczKase(portfel);
    cout << "w portfelu zostało: ";
    portfel.pokazKase();
}
```

wywołanie funkcji zaprzyjaźnionej

```
void Przyjaciel::pozyczKase(Portfel& p)
{
    cout << "masz w portfelu : " << p.kasa << endl;
    cout << "pożyczam 100 zł" << endl;
    p.kasa -= 100;
    cout << "dzięki!" << endl;
}
```

funkcja zaprzyjaźniona jest zdefiniowana w klasie Przyjaciel

```
Konsola debugowania
Pożyczka dla przyjaciela
masz w portfelu : 1000
pożyczam 100 zł
dzięki!
w portfelu zostało: 900
```

Klasy zaprzyjaźnione

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

// deklaracja klasy Portfel
class Portfel;

// klasa zaprzyjaźniona z klasą Portfel
class Przyjaciel {
public:
    void pozyczKase(Portfel& portfel);
    void dodajKase(Portfel& portfel);
};

class Portfel {
private:
    int kasa;
public:
    // konstruktory domyślny
    Portfel():kasa(1000) {};
    // deklaracja klasy Przyjaciel zaprzyjaźnionej z klasą Portfel
    // klasa Przyjaciel ma dostęp do prywatnych składników klasy Portfel
    friend class Przyjaciel;
    void pokazKase() { cout << kasa << endl; };
};
```

deklaracja klasy zaprzyjaźnionej
Przyjaciel, której obiekty mają dostęp
do prywatnych i chronionych
członków klasy Portfel

```
int main()
{
    cout << "Pożyczka dla przyjaciela\n\n";
    Portfel portfel;
    Przyjaciel przyjaciel;
    // wywołania funkcji zaprzyjaźnionych
    przyjaciel.pozyczKase(portfel);
    przyjaciel.dodajKase(portfel);
    cout << "w portfelu zostało: ";
    portfel.pokazKase();
}

void Przyjaciel::pozyczKase(Portfel& p)
{
    cout << "masz w portfelu : " << p.kasa << endl;
    cout << "pożyczam 100 zł" << endl;
    p.kasa -= 100;
    cout << "dzięki!" << endl;
}

void Przyjaciel::dodajKase(Portfel& p)
{
    cout << "masz w portfelu : " << p.kasa << endl;
    cout << "dokładam 1 zł" << endl;
    p.kasa += 1;
}
```



```
Konsola debugowania
Pożyczka dla przyjaciela
masz w portfelu : 1000
pożyczam 100 zł
dzięki!
masz w portfelu : 900
dokładam 1 zł
w portfelu zostało: 901
```


Klasy zaprzyjaźnione

Dana klasa może mieć kilka klas zaprzyjaźnionych

Dana klasa może być przyjacielem wielu klas

klasy zaprzyjaźnione

mechanizm przyjaźni nie jest dziedziczony

Przyjaźń między klasami nie jest wzajemna, jeśli deklarujemy, że klasa A jest przyjacielem klasy B, nie znaczy to, że automatycznie klasa B jest przyjacielem klasy A.

Przyjaźń nie jest przechodnia. - gdy klasa A jest przyjacielem klasy B, a klasa B jest przyjacielem klasy C, to nie oznacza to, że klasa A jest przyjacielem klasy C

Szablony funkcji

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

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

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

```
string dodaj(string a, string b) {
    return a + b;
}
```

```
int main()
{
    cout << dodaj(2,4) << endl;
    cout << dodaj(2.0, 4.0) << endl;
    string a = "duzy ";
    string b = "piesek";
    cout << dodaj(a,b) << endl;
}
```

funkcje mające różne typy parametrów
i tę samą funkcjonalność – dodają swoje
argumenty



czy nie można byłoby
zastąpić tych funkcji
jedną, uogólnioną?



```
Konsola
6
6
duzy piesek
```

Szablony funkcji

model, wzorzec funkcji operujący na typach uogólnionych

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

szablon funkcji dodaj()



```
template <typename T>  
T dodaj(T a, T b) {  
    return a + b;  
}
```

szablony funkcji realizują
tę samą funkcjonalność
dla różnych typów danych

Szablony funkcji

parametry szablone
oddzielone przecinkami w
nawiasach kątowych

słowo kluczowe typename (lub class)
określające parametr szablonowy(uogólniony)
reprezentujący uogólniony typ danych *generic data type*

słowo kluczowe
template

identyfikator
uogólnionego typu
danych

```
template <typename T>
```

funkcja zwraca
uogólniony typ
danych

```
T dodaj(T a, T b) {  
    return a + b;  
}
```

funkcja przyjmuje
parametry typu
uogólnionego

te same szablony
funkcji

można stosować
również słowo
kluczowe class

```
template <class T>  
T dodaj(T a, T b) {  
    return a + b;  
}
```

Szablony funkcji - wywołanie

przy wywołaniu funkcji szablonej `dodaj()` kompilator tworzy konkretną wersję funkcji dla konkretnych typów danych - jest to konkretyzacja szablonu

```
template <typename T>
T dodaj(T a, T b) {
    return a + b;
}
```

kod funkcji szablonej jest taki sam dla każdego jej wywołania, różni się skonkretyzowanym typem

tutaj nastąpi niejawna konwersja z typu podanego argumentu `double` na typ skonkretyzowany szablonu `long`

```
int main()
{
    cout << dodaj(2,4) << endl;
    cout << dodaj(2.0, 4.0) << endl;
    string a = "duzy ";
    string b = "piesek";
    cout << dodaj(a,b) << endl;
}
```

niejawne określenie parametrów szablonych – kompilator dedukuje typy na podstawie dostarczonych argumentów.

```
int main()
{
    cout << dodaj<int>(2,4) << endl;
    cout << dodaj<long>(2.0, 4.0) << endl;
    string a = "duzy ";
    string b = "piesek";
    cout << dodaj<string>(a,b) << endl;
}
```

jawne określenie parametrów szablonych (skonkretyzowanie typów) – szablon jest skonkretyzowany i dostarczone typy argumentów zostaną niejawnie skonwertowane na skonkretyzowany typ (jeśli będzie taka potrzeba)



```
Konsola
6
6
duzy piesek
```

Szablony funkcji - specjalizacja

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

```
template <class T>
T dodaj(T a, T b) {
    return a + b;
}
```

definicja szablonu
funkcji dodaj() –
wersja ogólna
(generyczna)

definicja
specjalizowanej wersji
szablonu funkcji dodaj()
dla typu int, jej kod różni
się od wersji ogólnej
szablonu

```
template<>
int dodaj(int a, int b) {
    cout << "suma liczb " << a << " oraz " << b << " wynosi ";
    return a + b;
}
```

```
int main()
{
    cout << dodaj(2,4) << endl;
    cout << dodaj(2.0, 4.0) << endl;
    cout << dodaj<char>('2','4') << endl;
}
```

w tym wywołaniu funkcji szablonej dodaj()
kompilator wybiera wersję specjalizowaną szablonu
(zostanie wykonana funkcja specjalizowana dla typu int)

przy wywołaniu funkcji szablonej
kompilator w pierwszej kolejności
szuka specjalizacji szablonu dla
podanego typu, gdy jej nie znajdzie
konkretyzuje szablon dla tego typu



```
Konsola debugowania programu
suma liczb 2 oraz 4 wynosi 6
6
f
```

dlaczego f ?

Szablon klasy

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

```
template <typename T>
class Dodawanie {
private:
    T a;
    T b;
public:
    // konstruktor
    Dodawanie(T aa, T bb) : a(aa), b(bb) {};
    T dodaj();
};
```

dzięki generycznemu szablónowi klasy można zdefiniować wiele klas w zależności od typów danych

definicja szablonu
klasy Dodawanie
wersja ogólna
(generyczna)

typy danych w
klasie zastępujemy
typem
generycznym
(uogólnionym) T

```
int main()
{
    Dodawanie <int> dwieLiczby(10, 20);
    cout << dwieLiczby.dodaj() << endl;

    Dodawanie <string> dwaNapisy("maly ", "kotek");
    cout << dwaNapisy.dodaj() << endl;

    Dodawanie <char> dwaZnaki('1', '0');
    cout << dwaZnaki.dodaj() << endl;
}
```

Kompilator generuje klasę na podstawie szablonu - instancję szablonu skonkretyzowaną dla podanego w nawiasach kwadratowych typu danych, na podstawie tak wygenerowanej klasy powstaje obiekt.

Szablon klasy jest parametryzowany parametrami szablonowymi – uogólnionymi typami danych

```
template<typename T>
T Dodawanie<T>::dodaj()
{
    return a + b;
}
```

definicja metody
dodaj()

Szablon klasy - specjalizacja

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

```
template <typename T>
class Dodawanie {
```

```
private:
```

```
    T a;
```

```
    T b;
```

```
public:
```

```
    // konstruktor
```

```
    Dodawanie(T aa, T bb) : a(aa), b(bb) {};
```

```
    T dodaj();
```

```
};
```

```
template <>
```

```
class Dodawanie <char> {
```

```
private:
```

```
    char a;
```

```
    char b;
```

```
public:
```

```
    // konstruktor
```

```
    Dodawanie(char aa, char bb) : a(aa), b(bb) {};
```

```
    char dodaj() {
```

```
        cout << "dodajemy kody ASCII znaków:\n" << a << "' - kod " << (int)a << endl << "' << b <<
            "' - kod " << (int)b << "\ni zwracamy znak, którego kodem jest otrzymana suma\n" <<
            "kod (suma) " << (int)a + (int)b << ": ";
```

```
        return a + b;
```

```
    };
```

```
};
```

definicja szablonu
klasy Dodawanie
wersja ogólna
(generyczna)

definicja szablonu
specjalizowanego
klasy Dodawanie,
parametrem
szablonu jest typ
char

definicja szablonu
specjalizowanego
klasy Dodawanie,
różni się od wersji
generycznej

Szablon klasy - specjalizacja

```
int main()
{
    Dodawanie <int> dwieLiczby(10, 20);
    cout << dwieLiczby.dodaj() << endl;

    Dodawanie <char> dwaZnaki('1', '0');
    cout << dwaZnaki.dodaj() << endl;
}

template<typename T>
T Dodawanie<T>::dodaj()
{
    return a + b;
}
```

przy tworzeniu obiektu na podstawie szablonu klasy kompilator w pierwszej kolejności szuka specjalizacji szablonu dla podanego typu, gdy jej nie znajdzie konkretyzuje szablon dla tego typu.

Gdy znajdzie specjalizację szablonu klasy dla podanego typu - tworzy obiekt korzystając ze znalezionej specjalizacji



Konsola debugowania programu Microsoft Visual Studio

```
30
dodajemy kody ASCII znaków:
'1' - kod 49
'0' - kod 48
i zwracamy znak, którego kodem jest otrzymana suma
kod (suma) 97: a
```

Szablony funkcji, funkcje przeciążone - polimorfizm

funkcje przeciążone
mogą różnić się
implementacją

```
void suma(int, int);  
void suma(int, int, int);  
void suma(double, double);
```

generyczny szablon funkcji ma
jednakową implementację dla
wszystkich funkcji

```
template <typename T>  
void suma(T a, T b) {  
    cout << a + b;  
}
```

specjalizowany szablon funkcji
ma odmienną implementację

```
template <>  
void suma(int a, int b) {  
    cout << "suma: " << a + b;  
}
```

przejawy polimorfizmu
(wielopostaciowości)

Obsługa błędów i wyjątków

komunikaty i kody błędów
zaimplementowane przez
programistę

obsługa błędów i wyjątków

mechanizm obsługi wyjątków wbudowany
w C++ (try, throw, catch)

wyjątki zaimplementowane
przez programistę - zmienne
wbudowane lub obiekty

wyjątki wbudowane w C++
(klasa exception)

Obsługa błędów i wyjątków

brak obsługi błędów



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

int main()
{
    double a, b;
    cout << "podaj 2 liczby" << endl;
    cin >> a >> b;
    cout << "iloraz podanych liczb: " << a / b ;
}
```



mianownik nie może być zerem

Obsługa błędów i wyjątków

komunikaty błędów
zaimplementowane
przez programistę

obsługa błędu
za pomocą instrukcji
warunkowej –
przechwycenie
i wyświetlenie
stosownego komunikatu

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

```
int main()
{
    double a, b;
    cout << "podaj 2 liczby: licznik i mianownik" << endl;
    cin >> a >> b;
    if (b != 0) {
        cout << "iloraz podanych liczb: " << a / b;
    }
    else {
        cout << "error, mianownik nie może być zerem" << endl;
    }
}
```

komunikat błędu



```
Konsola debugowania programu Microsoft Visual Studio
podaj 2 liczby: licznik i mianownik
1
0
error, mianownik nie może być zerem
```

Obsługa błędów i wyjątków

komunikaty błędów
zaimplementowane
przez programistę

obsługa błędu
za pomocą instrukcji
warunkowej
oraz funkcji
sprawdz()

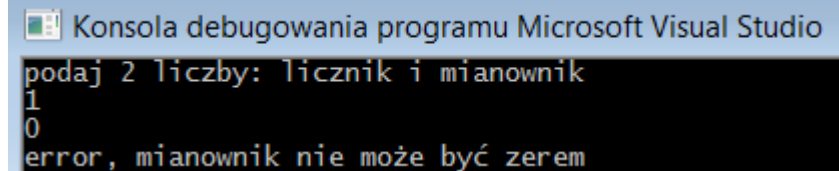
```
#include <iostream>
using namespace std;

int sprawdz(double mianownik, string& komunikat);

int main()
{
    double a, b;
    string komunikat = "";
    cout << "podaj 2 liczby: licznik i mianownik" << endl;
    cin >> a >> b;
    if (sprawdz(b, komunikat)) {
        cout << "iloraz podanych liczb: " << a / b;
    }
    else {
        cout << komunikat << endl;
    }
}

int sprawdz(double mianownik, string& komunikat)
{
    int kod = 1;
    if (mianownik == 0) {
        komunikat = "error, mianownik nie może być zerem";
        kod = 0;
        return kod;
    }
    komunikat = "";
    return kod;
}
```

funkcja weryfikująca błąd
i ustawiająca stosowny
komunikat



Konsola debugowania programu Microsoft Visual Studio

```
podaj 2 liczby: licznik i mianownik
1
0
error, mianownik nie może być zerem
```

Mechanizm obsługi wyjątków

konstrukcja try, throw, catch

mechanizm obsługi
wyjątków jest
wbudowany w C++

```
// wyjątkiem może być obiekt lub typ wbudowany(int, double, string itp)
Wyjatek wyjatek;

// w bloku try zamieszczamy kod, z którym mogą być problemy
try {
    // blok kodu, który może "rzucić" wyjątek
    throw wyjatek; // zgłoszenie wyjątku w przypadku problemu
}
// przechwycenie wyjątku - typ wyjątku musi się zgadzać z typem zgłoszonego wyjątku
catch (Wyjatek w) {
    // blok kodu obsługujący zgłoszony wyjątek
}
```

https://www.w3schools.com/cpp/cpp_exceptions.asp

<https://cpp0x.pl/kursy/Kurs-C++/Poziom-5/Wyjatki/596>

Mechanizm obsługi wyjątków

wyjątek typu string

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

```
int main()
{
    double a, b;
    string komunikat = "error, mianownik nie może być zerem";
    cout << "podaj 2 liczby: licznik i mianownik" << endl;
    cin >> a >> b;
    try {
        if (b == 0) {
            throw komunikat;
        }
        cout << "iloraz podanych liczb: " << a / b;
    }
    catch(string kom){
        cout << kom << endl;
    }
}
```

wyjątek - komunikat błędu typu string

W przypadku błędu „rzucamy” wyjątek

łapiemy wyjątek typu string w bloku catch i wypisujemy komunikat



```
Konsola debugowania programu Microsoft Visual Studio
podaj 2 liczby: licznik i mianownik
12
0
error, mianownik nie może być zerem
```


Mechanizm obsługi wyjątków

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

własne klasy wyjątków

```
// klasa abstrakcyjna wyjątek z metodą wczysto wirtualną - komunikatem
// korzystamy z poliformizmu dynamicznego
```

```
class Wyjatek{
public:
    virtual string komunikat() = 0;
};
```

definicje klas wyjątków
zaprojektowane przez
programistę
z abstrakcyjną klasą
Wyjatek na górze hierarchii
dziedziczenia,
w ten sposób skorzystamy
z polimorfizmu
dynamicznego

```
// klasa Mianownik - wyjątek dzielenia przez zero
```

```
class MianownikZero : public Wyjatek {
public:
    string komunikat() {
        string k = "error, mianownik nie może być zerem";
        return k;
    }
};
```

klasy wyjątków dziedziczą
publicznie z abstrakcyjnej
klasy bazowej

```
// klasa Liczba - wyjątek wielkości licznika
```

```
class LicznikZlaWielkosc : public Wyjatek {
public:
    string komunikat() {
        string k = "error, licznik musi być większy od 10";
        return k;
    }
};
```

Mechanizm obsługi wyjątków

własne
klasy
wyjątków

```
int main()
{
    double a, b;
    cout << "podaj 2 liczby: licznik (większy od 10) i mianownik" << endl;
    cin >> a >> b;
    try {
        // sprawdzamy licznik i gdy jest za mały rzucamy wyjątek w postaci
        // obiektu klasy LicznikZlaWielkosc
        if (a <= 10) {
            throw LicznikZlaWielkosc();
        }
        // sprawdzamy mianownik i gdy jest zerem rzucamy wyjątek
        // w postaci obiektu klasy MianownikZero
        if (b == 0) {
            throw MianownikZero();
        }
        cout << "iloraz podanych liczb: " << a / b;
    }
    // łapiemy wyjątki w postaci referencji obiektów klasy Wyjatek
    // w ten sposób korzystamy z mechanizmu polimorfizmu dynamicznego
    catch(Wyjatek& w){
        // wypisujemy stosowny komunikat w zależności od obiektu błędu
        cout << w.komunikat();
    }
}
```



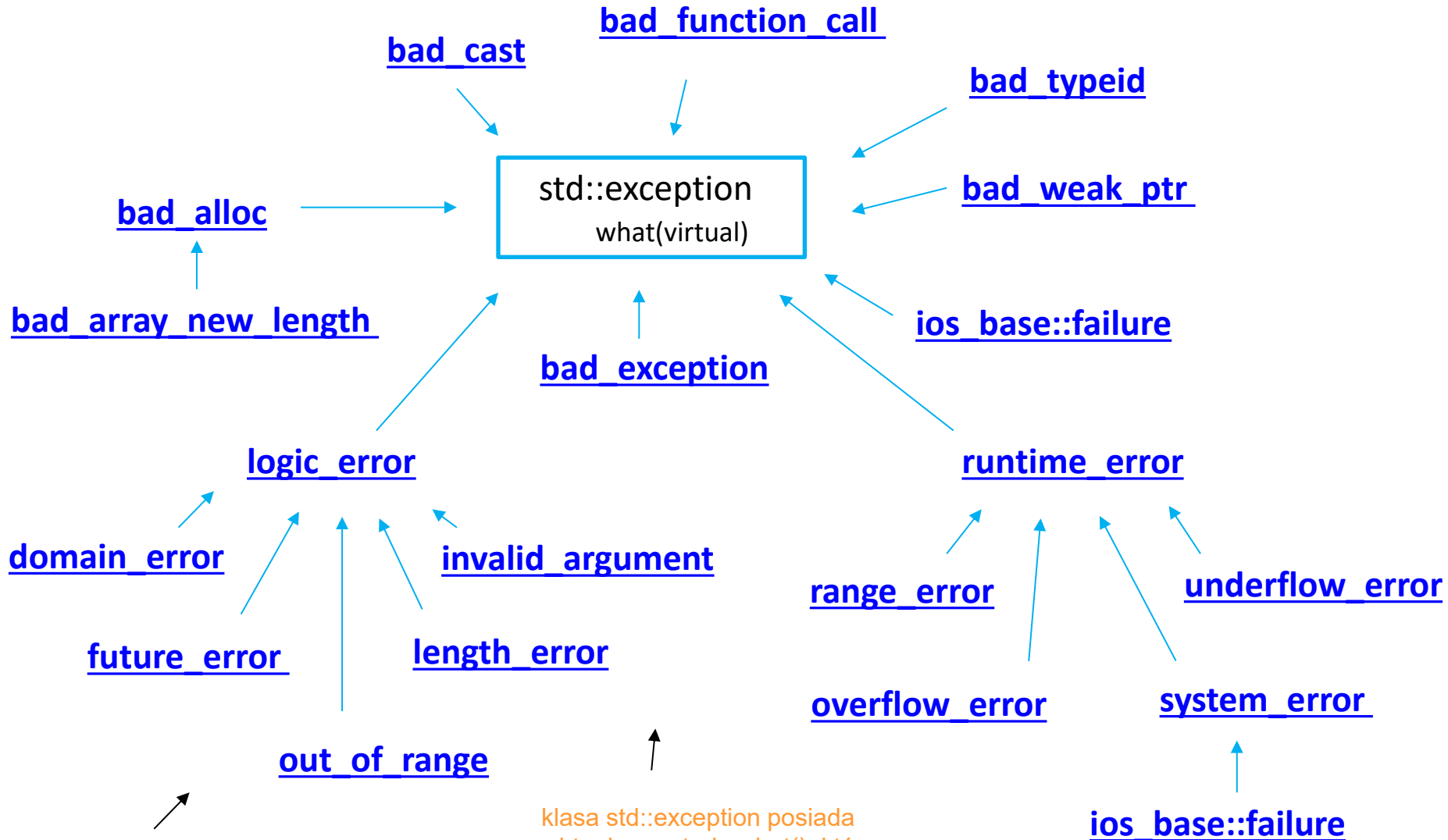
Konsola debugowania programu Microsoft Visual Studio

```
podaj 2 liczby: licznik (większy od 10) i mianownik
10
30
error, licznik musi być większy od 10
```

Konsola debugowania programu Microsoft Visual Studio

```
podaj 2 liczby: licznik (większy od 10) i mianownik
11
0
error, mianownik nie może być zerem
```

Mechanizm obsługi wyjątków



wbudowane klasy
wyjątków dziedziczące z
klasy `std::exception`

klasa `std::exception` posiada
wirtualną metodę `what()`, którą
można zdefiniować we
własnej klasie wyjątku (jest
opisem sytuacji wyjątkowej)

<https://www.cplusplus.com/reference/exception/exception/>

<https://en.cppreference.com/w/cpp/error/exception>

Mechanizm obsługi wyjątków

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

```
int main()
{
    while (true) {
        int* a = new int[1000000];
    }
}
```

alokujemy pamięć na stacku
aż do jej wyczerpania



Nieobsługiwany wyjątek

Nieobsłużony wyjątek w lokalizacji 0x764EC5AF w wyjatki.exe: wyjątek
języka Microsoft C++: std::bad_alloc w lokalizacji pamięci 0x0015F990.

wyjątek std::bad_alloc

Mechanizm obsługi wyjątków

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

int main()
{
    throw logic_error("błąd logiczny");
}
```

Nieobsługiwany wyjątek

Nieobsłużony wyjątek w lokalizacji 0x764EC5AF w wyjatki.exe: wyjątek języka Microsoft C++: std::logic_error w lokalizacji pamięci 0x0027FC2C.

[Kopiuuj szczegóły](#) | [Rozpocznij sesję rozszerzenia Live Share...](#)

► [Ustawienia wyjątków](#)

wyjątek std::logic_error nieobsłużony

opis wyjątku – dostęp przez e.what()

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

int main()
{
    try {
        throw logic_error("błąd logiczny");
    }
    catch (logic_error& e) {
        cout << e.what();
    }
}
```

Konsola debu
błąd logiczny

wyjątek std::logic_error obsługowany

Mechanizm obsługi wyjątków

własne klasy wyjątków
dziedziczące z
bibliotecznej klasy
exception

```
#include <iostream>
#include <exception>    // klasa exception
using namespace std;

// klasa Mianownik - wyjątek dzielenia przez zero
// dziedziczy z bibliotecznej klasy exception
// nadpisujemy wirtualna metodę what() zdefiniowana w tej klasie
// (metoda what() zostaje przesłonięta)
class MianownikZero : public exception {
public:
    const char* what() const throw() {
        const char* komunikat = "error, mianownik nie może być zerem";
        return komunikat;
    }
};

// klasa Liczba - wyjątek wielkości licznika
// dziedziczy z bibliotecznej klasy exception
// nadpisujemy wirtualna metodę what() zdefiniowana w tej klasie
class LicznikZlaWielkosc : public exception {
public:
    const char* what() const throw() {
        return "error, licznik musi być większy od 10";
    }
};
```

Mechanizm obsługi wyjątków

własne klasy wyjątków
dziedziczące z
bibliotecznej
klasy exception

```
int main()
{
    double a, b;
    cout << "podaj 2 liczby: licznik (większy od 10) i mianownik" << endl;
    cin >> a >> b;
    try {
        // sprawdzamy licznik i gdy jest za mały rzucamy wyjątek w postaci
        // obiektu klasy LicznikZlaWielkosc
        if (a <= 10) {
            throw LicznikZlaWielkosc();
        }
        // sprawdzamy mianownik i gdy jest zerem rzucamy wyjątek
        // w postaci obiektu klasy MianownikZero
        if (b == 0) {
            throw MianownikZero();
        }
        cout << "iloraz podanych liczb: " << a / b;
    }
    // łapiemy wyjątki w postaci referencji obiektów klasy exception
    // w ten sposób korzystamy z mechanizmu polimorfizmu dynamicznego
    catch(exception& e){
        // wypisujemy stosowny komunikat w zależności od obiektu błędu
        cout << e.what();
    }
}
```



Konsola debugowania programu Microsoft Visual Studio

```
podaj 2 liczby: licznik (większy od 10) i mianownik
10
30
error, licznik musi być większy od 10
```

Konsola debugowania programu Microsoft Visual Studio

```
podaj 2 liczby: licznik (większy od 10) i mianownik
11
0
error, mianownik nie może być zerem
```

Standard Template Library STL

standardowa biblioteka szablonów

biblioteka C++ zawierająca algorytmy, pojemniki, iteratory oraz inne konstrukcje w formie szablonów, gotowe do użycia w programach.

kontenery

Kontener to obiekt, który przechowuje kolekcję innych obiektów (jego elementów). Kontenery są zaimplementowane jako szablony klas, co pozwala na dużą elastyczność w typach obsługiwanych jako elementy.

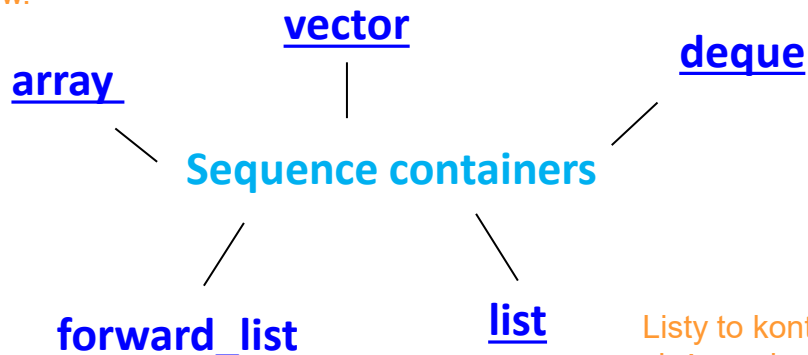
Sequence containers to kontenery sekwencyjne – elementy są przechowywane w kolejności, w której zostały wstawione, można je iterować, czyli przechodzić od elementu kontenera do elementu kontenera po kolei.

Tablice są kontenerami sekwencyjnymi o stałym rozmiarze, przechowują określoną liczbę uporządkowanych elementów.

Wektory to kontenery sekwencyjne reprezentujące tablice, których rozmiar może się zmieniać (tablice dynamiczne)

Kolejki o podwójnym końcu (double-ended queues) to kontenery sekwencyjne o dynamicznym rozmiarze, które można rozszerzać lub skracać na obu końcach (z przodu lub z tyłu).

W przeciwieństwie do wektorów, deque nie gwarantują przechowywania wszystkich swoich elementów w ciągłych lokalizacjach pamięci



Forward list to kontenery sekwencyjne, służące do przechowywania elementów w węzłach, które mają linki (wskaźniki) do węzła następnego (lista jednokierunkowa – można iterować listę w jednym kierunku).

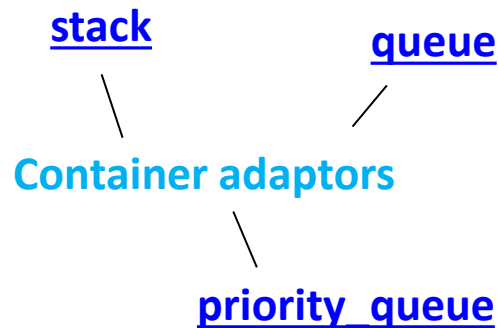
Listy to kontenery sekwencyjne, służące do przechowywania elementów w węzłach, które mają linki (wskaźniki) do węzła poprzedzającego i następnego (lista dwukierunkowa – można iterować listę w obu kierunkach).

kontenery

Container adapters to adaptery kontenerów, czyli klasy, które używają w swoim wnętrzu obiektu określonej klasy kontenera jako kontenera bazowego, zapewniając określony zestaw funkcji członkowskich w celu uzyskania dostępu do jego elementów.

LIFO – Last In First Out (np. stos talerzy do zmywania)

FIFO – First In First Out (np. kolejka po zakupy)



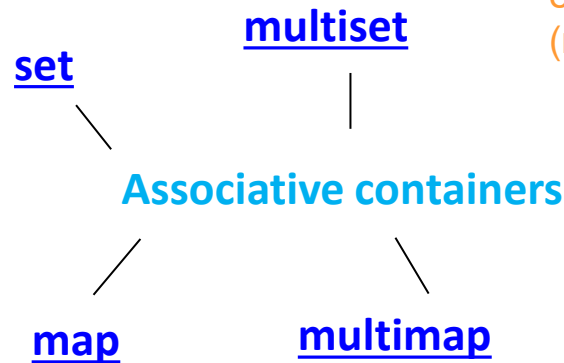
Kolejka, której elementy mają przypisany priorytet (na wyjściu pojawiają się dane, które mają największy priorytet np. kolejka po zakupy, w której stoją kobiety ciężarne lub z dzieckiem na ręku)

kontenery

Associative containers to kontenery asocjacyjne przechowujące klucze oraz odpowiadające im wartości.

Set (zbór) to kontener, który przechowuje unikalne elementy (wartości) w określonej kolejności. Wartość jest tutaj kluczem. Danego elementu nie można zmienić, można go wstawić lub usunąć.

Multiset to kontener podobny do Set, z tym, że jego elementy mogą się powtarzać (nie są unikalne)



Mapy to kontenery, które przechowują elementy utworzone przez kombinację wartości klucza i wartości zmapowanej w określonej kolejności. W mapie wartości klucza są zwykle używane do sortowania i jednoznacznej identyfikacji elementów (klucze są unikalne), podczas gdy zmapowane wartości przechowują zawartość powiązaną z tym kluczem.

Multimap to kontener podobny do Map, z tym, że jego klucze mogą się powtarzać (nie są unikalne)

kontenery

Unordered associative containers to kontenery asocjacyjne, które przechowują klucze oraz odpowiadające im wartości.

Unordered set to kontener podobny do Set tylko elementy (wartości) nie są ułożone w określonej kolejności.

unordered_set

unordered_multiset

Unordered multiset to kontener podobny do Multiset tylko elementy (wartości) nie są ułożone w określonej kolejności.

Unordered associative containers

unordered_map

unordered_multimap

Unordered map to kontener podobny do Map tylko elementy (wartości) nie są ułożone w określonej kolejności.

Unordered multimap to kontener podobny do Multimap elementy (wartości) nie są ułożone w określonej kolejności.

Vector

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

aby używać wektora trzeba
dodać nagłówek

```
int main()
```

```
{
```

```
    // deklaracja i inicjalizacja wektora
```

```
    vector<string> napisy = {"wiosna jest łagodna",
                             "lato jest ciepłe",
                             "jesień jest chłodna",
                             "zima jest mroźna"};
```

vector jest szablonem klasy

```
}
```

Wektor to tablica dynamiczna. Jest jedną ze kilku struktur danych zwanych kontenerem (obok list, map, set itd.) Jest zaimplementowany jako szablon klasy, czyli ogólny framework, który może być zdefiniowany jako np.: wektor wartości całkowitych, wektor łańcuchów, wektor obiektów klasy zdefiniowanej przez użytkownika itp.

Vector

```
#include <iostream>
#include <vector>
#include <string> // getline()
using namespace std;

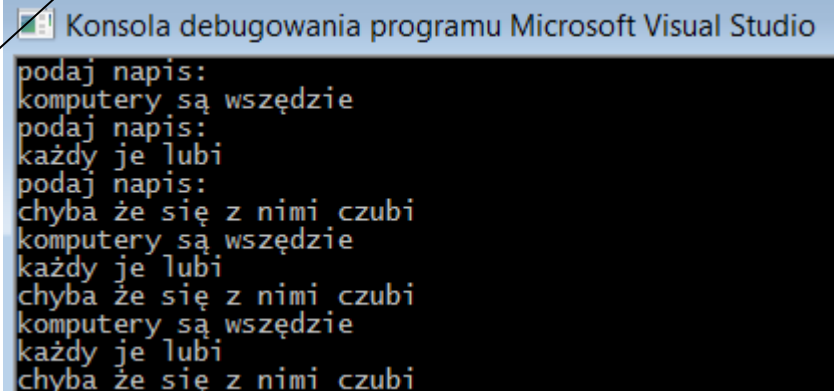
int main()
{
    // deklaracja trzelementowego wektora zainicjowanego pustymi łańcuchami tekstowymi
    vector<string> napisy(3);

    // użytkownik podaje napisy umieszczane w wektorze
    // element wektora czyli zmienna napis jest referencją typu string
    // dlatego operujemy w pętli na oryginalnym elemencie wektora
    // a nie na kopii
    for (string& napis : napisy)
    {
        cout << "podaj napis:" << endl;
        // pobieramy całą linijkę
        getline(cin, napis);
    }

    // wypisanie wartości wektora na konsoli - 1 sposób
    for (int i = 0; i < napisy.size(); i++)
    {
        cout << napisy[i] << endl;
    }

    // wypisanie wartości wektora na konsoli - 2 sposób
    for (string napis : napisy)
    {
        cout << napis << endl;
    }
}
```

pętla foreach (C++11)

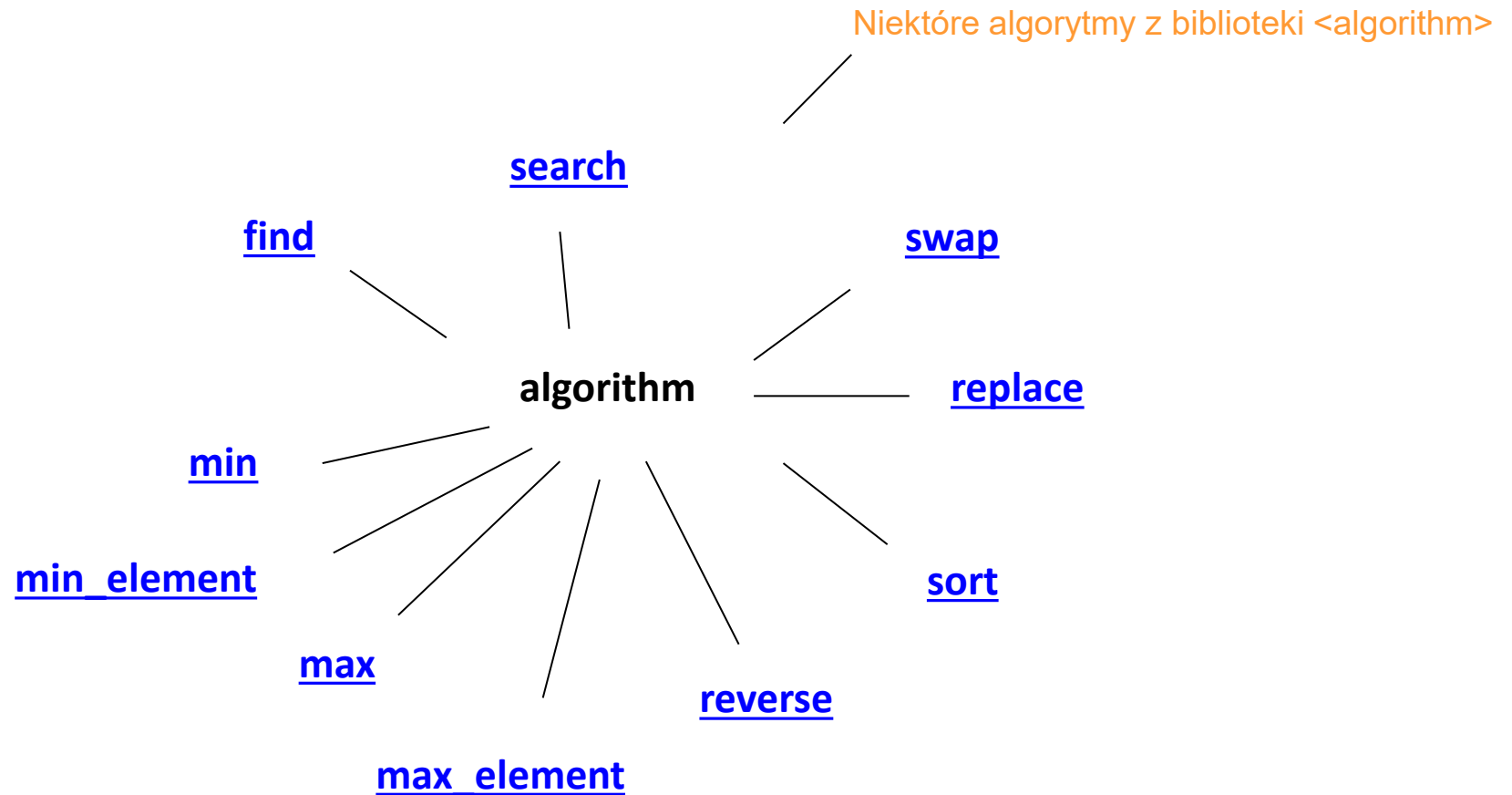


Konsola debugowania programu Microsoft Visual Studio

```
podaj napis:
komputery są wszędzie
podaj napis:
każdy je lubi
podaj napis:
chyba że się z nimi czubi
komputery są wszędzie
każdy je lubi
chyba że się z nimi czubi
komputery są wszędzie
każdy je lubi
chyba że się z nimi czubi
```

algorytmy

Biblioteka STL zawiera duży zbiór algorytmów, które rozwiązują podstawowe problemy, na jakie może się natknąć programista.



algorytmy sort

```
#include <iostream>
#include<algorithm> // sort()
using namespace std;
```

szablon funkcji sort() służy
do sortowania tablic

```
int main()
{
    // tablica do posortowania - posiada 9 elementów
    int tab[] = { 9,4,6,3,54,62,-67,2,0 };

    cout << "tablica nieposortowana" << endl;
    for (int item : tab) {
        cout << item << " ";
    }

    // sortujemy tablicę,
    // argumentami szablonu funkcji sort() są: pierwszy i ostatni element tablicy
    sort(tab, tab + 9);
    cout << "\ntablica posortowana" << endl;
    for (int item : tab) {
        cout << item << " ";
    }
}
```


algorytmy sort

```
#include <iostream>
#include<algorithm> // sort()
#include<vector>     // vector
using namespace std;
```

sortowanie wektora

```
int main()
{
    // wektor do posortowania
    vector<int> tab;

    // inicjalizacja wektora
    for (int i = 10; i >= 0; i--) tab.push_back(i);

    cout << "wektor nieposortowany" << endl;
    // wypisanie elementów wektora z pomocą iteratorów
    for (vector<int>::iterator it = tab.begin(); it != tab.end(); ++it) {
        cout << *it << " ";
    }

    // sortujemy tablicę,
    // argumentami szablonu funkcji sort() są: pierwszy i ostatni element wektora
    sort(tab.begin(), tab.end());
    cout << "\nwektor posortowany" << endl;
    for (int item : tab) {
        cout << item << " ";
    }
}
```

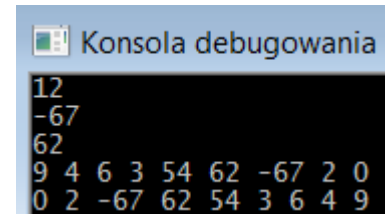
reference objects with similar properties to pointers

algorytmy min, max, reverse

```
#include <iostream>
#include<algorithm> // sort()
using namespace std;

int main()
{
    // tablica do posortowania - posiada 9 elementów
    int tab[] = { 9,4,6,3,54,62,-67,2,0 };

    // zwraca mniejszą wartość z dwóch podanych
    cout << min(12,45) << endl;
    // zwraca najmniejszą wartość z zakresu elementów (tablicy)
    cout << *std::min_element(tab, tab + 9) << endl;
    // zwraca największą wartość z zakresu elementów (tablicy)
    cout << *std::max_element(tab, tab + 9) << endl;
    for (int item : tab) {
        cout << item << " ";
    }
    cout << endl;
    // odwraca kolejność elementów w tablicy
    reverse(tab, tab + 9);
    for (int item : tab) {
        cout << item << " ";
    }
}
```



Konsola debugowania

```
12
-67
62
9 4 6 3 54 62 -67 2 0
0 2 -67 62 54 3 6 4 9
```

[Tworzenie kalkulatora konsoli w C++ w oparciu o klasy](#)

[Create a simple Universal Windows Platform \(UWP\) game with DirectX](#)

[C++ language documentation \(Microsoft\) - można pobrać pdf](#)

[Kurs C++](#)

ODPOWIEDZI

menu

```
using namespace std;

int main()
{
    cout << "Kelner do uslug!\n\n\n";

    cout << "Oto nasze menu\n\n\n";

    // nr pozycji na karcie
    int nr;

    cout << "Ktora pozycje z karty chca Panstwo zobaczyc?\nProsze wybrac 1-4 (q exit): ";
    // w przypadku podania litery wyrażenie cin>>nr zwraca false (litera nie jest typu int)
    while (cin >> nr) {
        switch (nr) {
            case 1:
                cout << endl << "Zurek z kielbasa" << endl << endl;
                break;
            case 2:
                cout << endl << "Rosol z makaronem" << endl << endl;
                break;
            case 3:
                cout << endl << "Schabowy z ziemniakami" << endl << endl;
                break;
            case 4:
                cout << endl << "Lazanie z frytkami" << endl << endl;
                break;
            default:
                cout << endl << "Takiej pozycji nie ma na karcie" << endl << endl;
        }
        cout << "Ktora pozycje z karty chca Panstwo zobaczyc?\nProsze wybrac 1-4 (q exit): ";
    }

    cout << "\n\n\nDziekujemy i zapraszamy ponownie";
}
```

zamowienie

```
using namespace std;

int main()
{
    cout << "Zamowienie\n\n\n";
    cout << "Witamy w naszej restauracji\n\n";
    cout << "Oto nsze menu:\n";

    cout << "1 - zurek z kielbasa\n";
    cout << "2 - rosol z makaronem\n";
    cout << "3 - schabowy z ziemniakami\n";
    cout << "4 - lody czekoladowe\n";

    int wybor;
    cout << "\nProsze wybrac potrawe z menu:";
    cin >> wybor;
    cout << "\nOto co Panstwo wybraliscie:\n\n";

    switch (wybor) {
    case 1:
        cout << "Wybraliscie Panstwo wspanaily zurek z kielbasa, z pewnoscia bedzie Panstwu smakowal.\n";
        break;
    case 2:
        cout << "Rosol z makaronem to dobry wybor. Smacznego!\n";
        break;
    case 3:
        cout << "Typowa polska potrawa: schabowy z ziemniakami. Na pewno sie Panstwo najecie.\n";
        break;
    case 4:
        cout << "Lody czekoladowe - niebo w giebie.\n";
    default:
        cout << "Takiej pozycji nie ma w menu.\n";
    }
}
```

Struktury

```
#include <iostream>

using namespace std;

int main()
{
    //definicja struktury
    struct osoba {
        string imie;
        string nazwisko;
        short wiek;
    };

    //tworzymy egzemplarz struktury
    osoba pierwsza;

    //inicjalizacja egzemplarza
    cout << "podaj imie" << endl;
    cin >> pierwsza.imie;

    cout << "podaj nazwisko" << endl;
    cin >> pierwsza.nazwisko;

    cout << "podaj wiek" << endl;
    cin >> pierwsza.wiek;

    //wypisujemy dane
    cout << "podałeś następujące dane:" << endl;
    cout << "imie: " << pierwsza.imie << endl;
    cout << "nazwisko: " << pierwsza.nazwisko << endl;
    cout << "wiek: " << pierwsza.wiek << endl;

    system("PAUSE");
    return 0;
}
```

Tablica jednowymiarowa (statyczna)

```
int tab[4];           //deklaracja tablicy
```

```
for(int i=0; i<=3;i++)    //pętla inicjalizująca tablicę  
    tab[i]=i;
```

```
for(int i=0; i<=3;i++)    //pętla wyświetlająca zawartość  
    cout<<tab[i]<<endl;  //tablicy na konsoli
```


Kolej

```
char slowo[6]="kolej";           //deklaracja tablicy
```

```
for(int i=0; i<5;i++)           //pętla wyświetlająca  
    cout<<slowo[i]<<endl;
```

Lustro

```
char tab[]={"marek"};    //deklaracja tablicy
```

```
for(int i=0; i<5; i++)    //pętla wypisująca imię  
    cout<<tab[i];
```

```
cout<<" | ";              //"lustro"
```

```
for(int i=4; i>=0; i--)    //pętla wypisująca imię od końca  
    cout<<tab[i];
```

Funkcje

Definicja funkcji

```
int potega (int x, int n)
{
    int i,p; //zmienne lokalne funkcji potega
    p=1;
    for (i=1; i<=n; i++)
        p=p*x;
    return p;
}
```

Funkcje

```
#include<iostream>

using namespace std;

//deklaracja funkcji potega
int potega(int x, int n);

int main() {

    //liczba
    int podstawa;

    //wykladnik potęgi
    int wykladnik;

    //potęga
    int a;

    cout << "podaj liczbe:" << endl;
    cin >> podstawa;

    cout << "podaj wykladnik:" << endl;
    cin >> wykladnik;

    //obliczamy potęgę
    //wywołanie funkcji potega
    a = potega(podstawa, wykladnik);

    cout << podstawa << " do potegi " << wykladnik << " = " << a << endl;

}

//definicja funkcji potega
int potega(int x, int n) {

    int i, p;
    p = 1;
    for (i = 1; i <= n; i++)
        p = p * x;
    return p;
}
```

Rekurencja

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

int silnia(int a);

int main()
{
    cout << "Silnia\n\n";
    cout << "podaj liczbę:" << endl;
    int liczba = 0;
    cin >> liczba;
    cout << "silnia liczby " << liczba << " wynosi " << silnia(liczba) << endl;
}

int silnia(int a)
{
    if (a == 0) {
        return 1;
    }
    else if (a == 1) {
        return 1;
    }
    else {
        return a *= silnia(a - 1);
    }
}
```

wywołanie rekurencyjne funkcji



Klasa Instrument - hermetyzacja (enkapsulacja)

```
using namespace std;

class Instrument {
private:
    string nazwa;
    string kolor;
    int waga;
public:
    void graj();
    string getNazwa();
    void setNazwa(string naz);
    string getKolor();
    void setKolor(string col);
    int getWaga();
    void setWaga(int wag);
};

void Instrument::graj()
{
    cout << "instrument " << nazwa << " pięknie gra" << endl;
}

string Instrument::getNazwa()
{
    return nazwa;
}

void Instrument::setNazwa(string naz)
{
    nazwa = naz;
}

string Instrument::getKolor()
{
    return kolor;
}

void Instrument::setKolor(string col)
{
    kolor = col;
}

int Instrument::getWaga()
{
    return waga;
}

void Instrument::setWaga(int wag)
{
    waga = wag;
}

int main()
{
    Instrument instrument;
    instrument.setKolor("zielony");
    instrument.setNazwa("gitara");
    instrument.setWaga(2);
    cout << "instrument o nazwie " << instrument.getNazwa()
        << ": \nma kolor " << instrument.getKolor() << " \nważy "
        << instrument.getWaga() << "kg \n";
}
```

```
// metoda zwraca ciąg tekstowy z opisem pól instrumentu
string Instrument::toString()
{
    return "Instrument:\nnazwa: " + nazwa + ",\nkolor: " + kolor + ",\nwaga: " + to_string(waga);
}

void Instrument::graj()
{
    cout << "instrument " << nazwa << " pięknie gra" << endl;
}

void Gitara::grajPalcami()
{
    cout << "gram palcami" << endl;
}

void Gitara::grajPiorkiem()
{
    cout << "gram piórkiem" << endl;
}

// metoda wywołuje toString() klasy bazowej i dopisuje opis prywatnych pól klasy pochodnej
string Gitara::toString()
{
    return Instrument::toString() + ",\nliczba strun: " + to_string(liczbaStrun) + ",\ntyp: " + typ + "\n";
}

int main()
{
    // tworzymy obiekt klasy Gitara
    Gitara hawajska("gitara", "czerwony", 1, 4, "hawajska");
    // opis obiektu hawajska
    cout << hawajska.toString();
    hawajska.graj();
    hawajska.grajPalcami();

    // tworzymy drugi obiekt klasy Gitara
    Gitara elektryczna("gitara", "czarny", 2, 6, "elektryczna");
    // opis obiektu elektryczna
    cout << elektryczna.toString();
    elektryczna.grajPiorkiem();
}
```