C++ wciągnie Was

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)

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,
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)

Od kodu źródłowego do pliku wykonywalnego

From source code to exe

dyrektywa preprocesora

preprocessor directive

main.cpp

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

Kod źródłowy

Source code

wkleja zawartość biblioteki (pliku) iostream do kodu źródłowego

replaces #include directives with the content of the respective files, handles other directives too

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 jedynkowy) , wynikiem jest tzw. plik obiektowy*

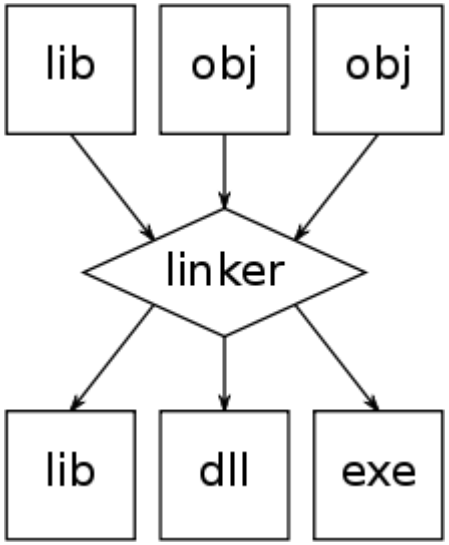
Assembler assembles that code into machine code producing binary file (object file)

linkowanie

Linking

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

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

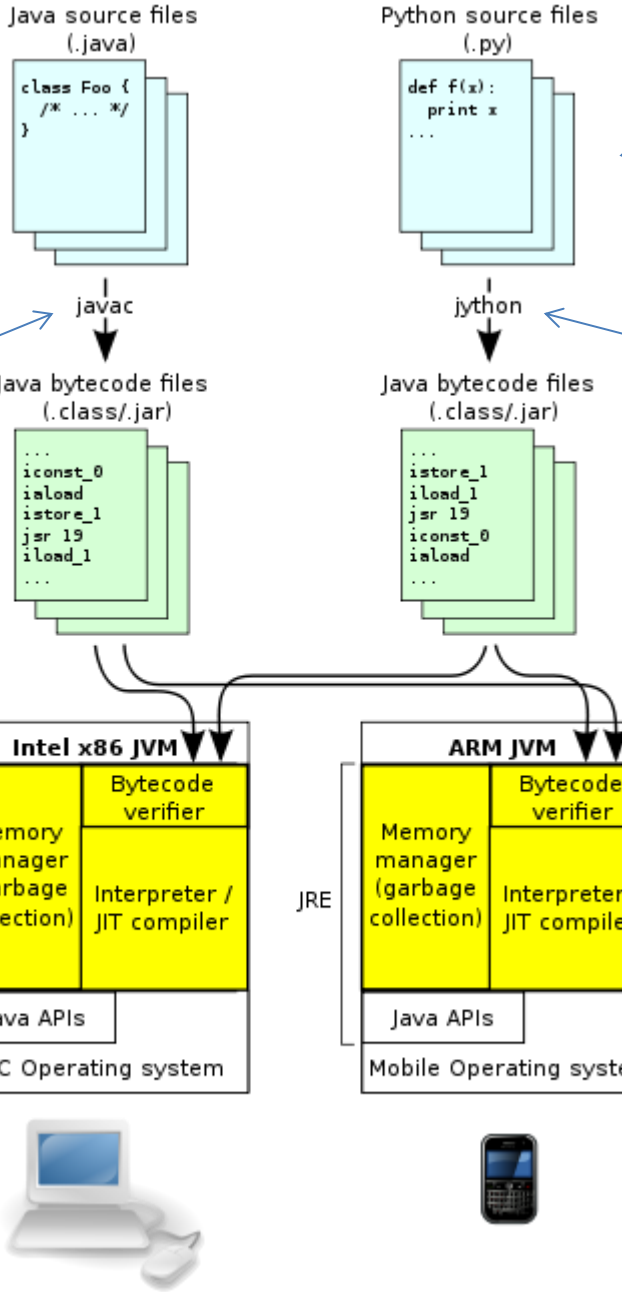


An illustration of the linking process. Object files and static libraries are assembled into a new library or executable

https://en.wikipedia.org/wiki/Interpreter_(computing)

Java

język kompilowany do kodu bajtowego uruchamianego przez JVM



Kod źródłowy
Source code

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

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

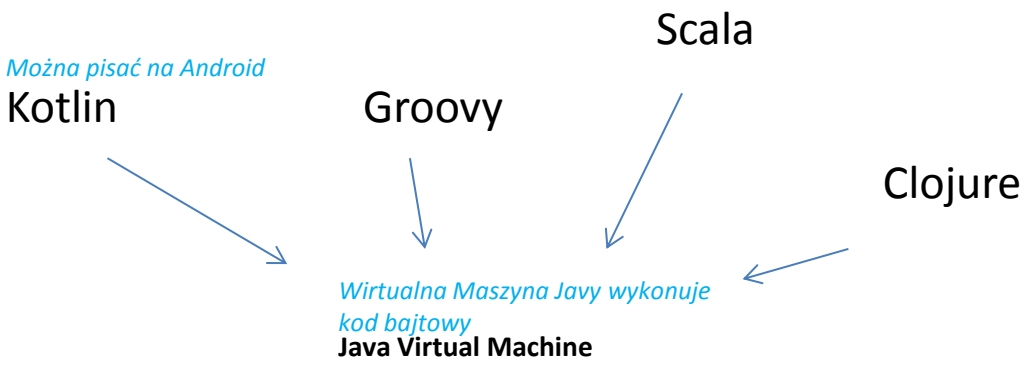
Wirtualna Maszyna Javy wykonuje kod bajtowy
Java Virtual Machine

Java Virtual Machine

Środowisko uruchomieniowe
Java Runtime Environment

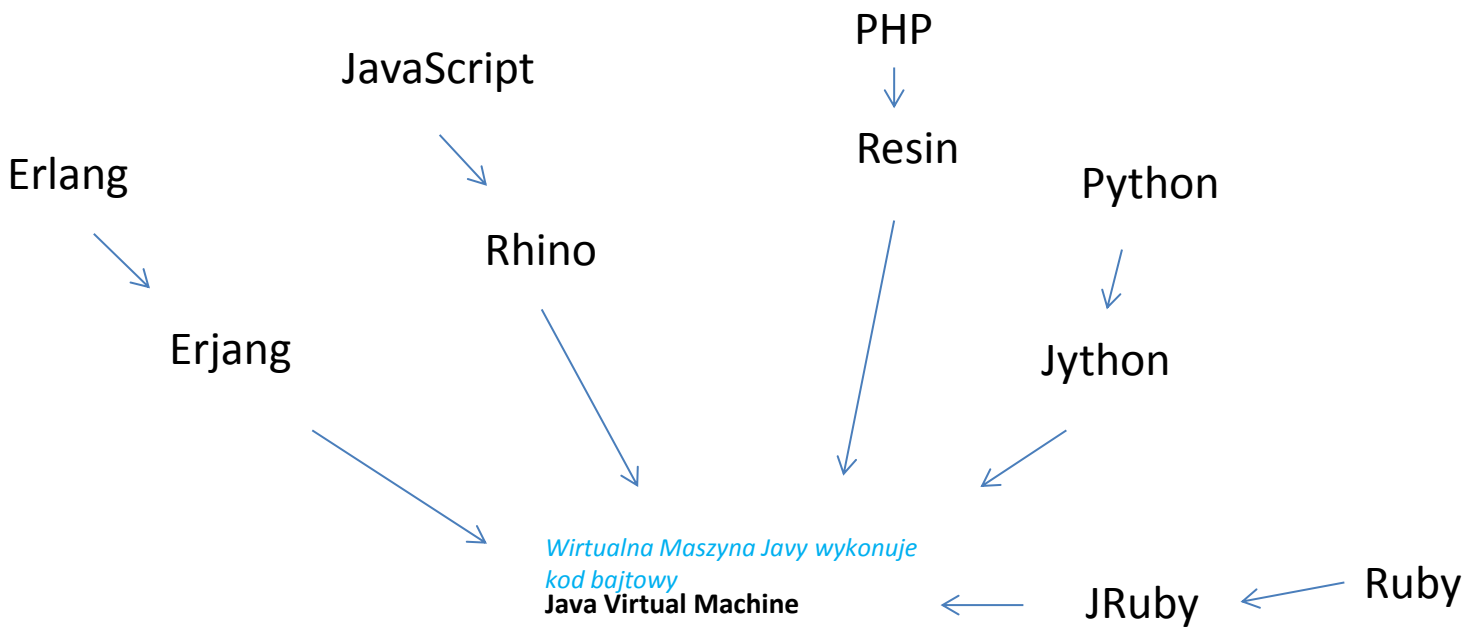
źródło: https://pl.wikipedia.org/wiki/Wirtualna_maszyna_Javy

Języki (nowe) wykonywane na JVM



źródło: https://pl.wikipedia.org/wiki/Wirtualna_maszyna_Javy

Implementacje istniejących języków na JVM



źródło: https://pl.wikipedia.org/wiki/Wirtualna_maszyna_Javy

Python język interpretowany

```
def helloWorld():
  print("Hello World!")

helloWorld()
```

Kod źródłowy
Source code

Brak kompilacji

Interpreter analizuje kod źródłowy programu i przeanalizowane fragmenty wykonuje

Interpreter

Efekt programu na konsoli

```
Hello World!
```

[Godzina kodowania Python Silient Teacher]

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

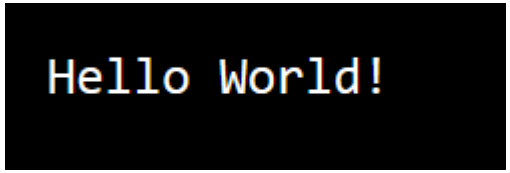
kompilacja
Compilation

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

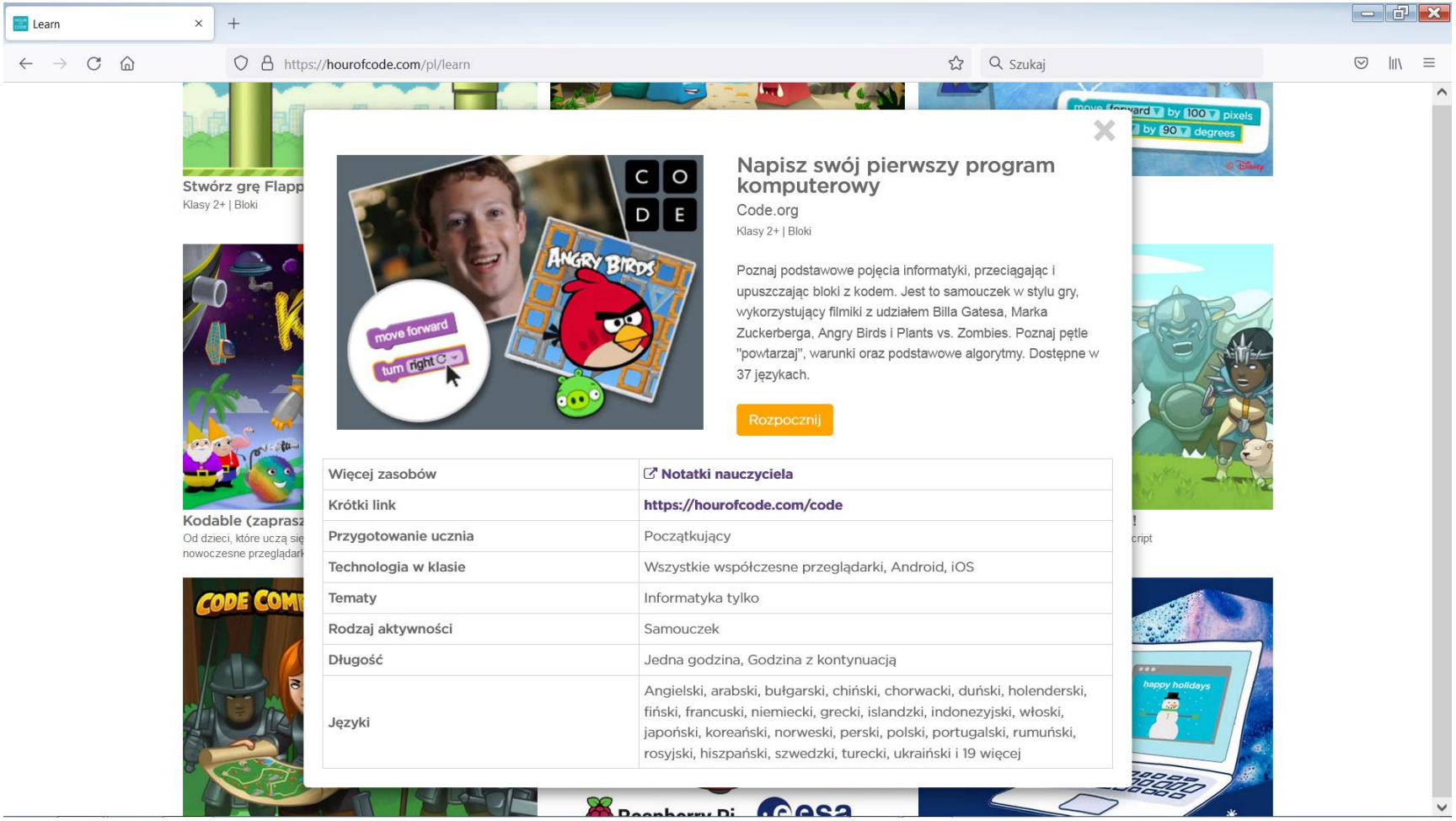
.NET Framework

środowisko uruchomieniowe (jest ich więcej) zainstalowane na komputerze

Efekt programu na konsoli

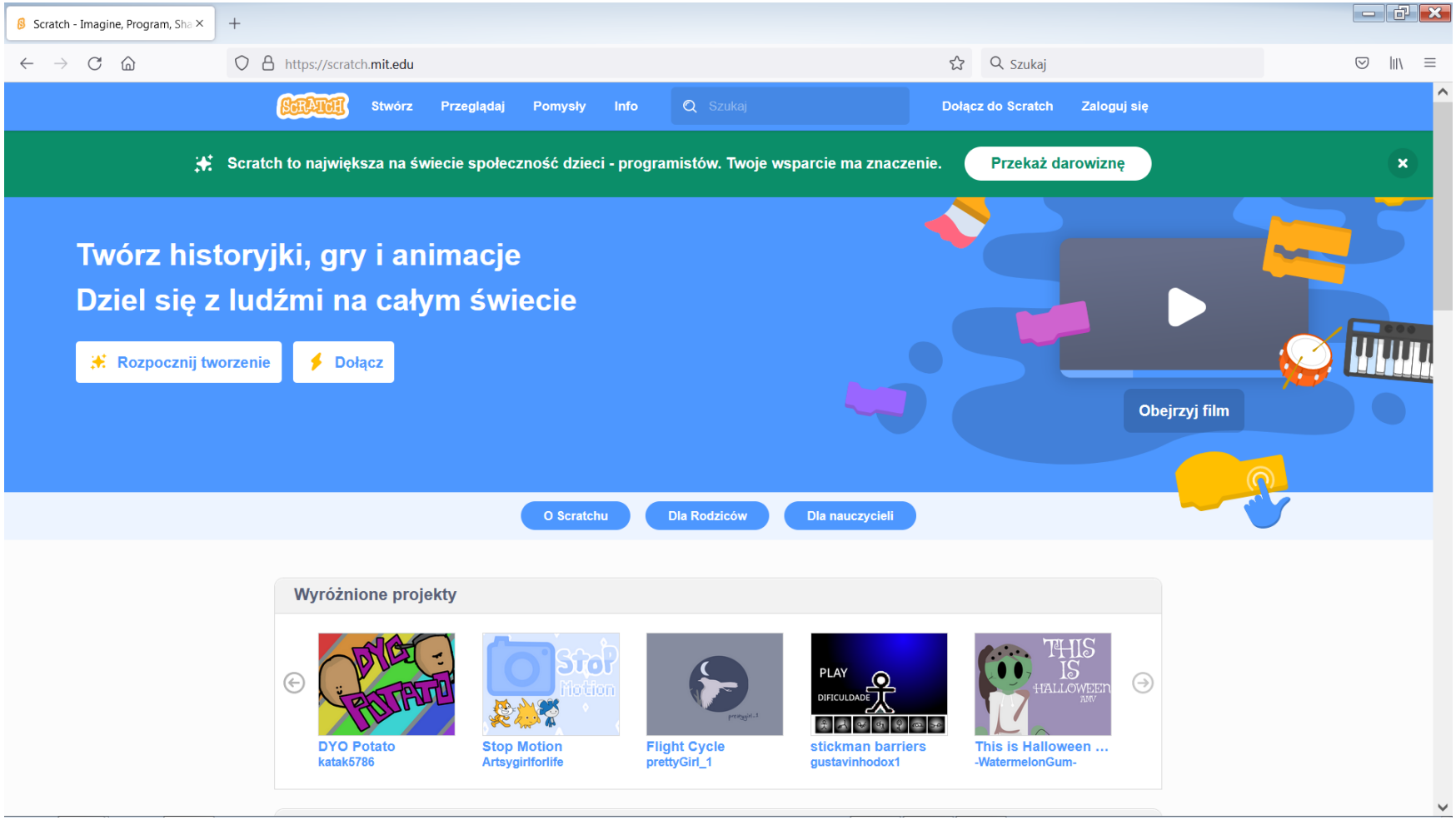


Hour of code



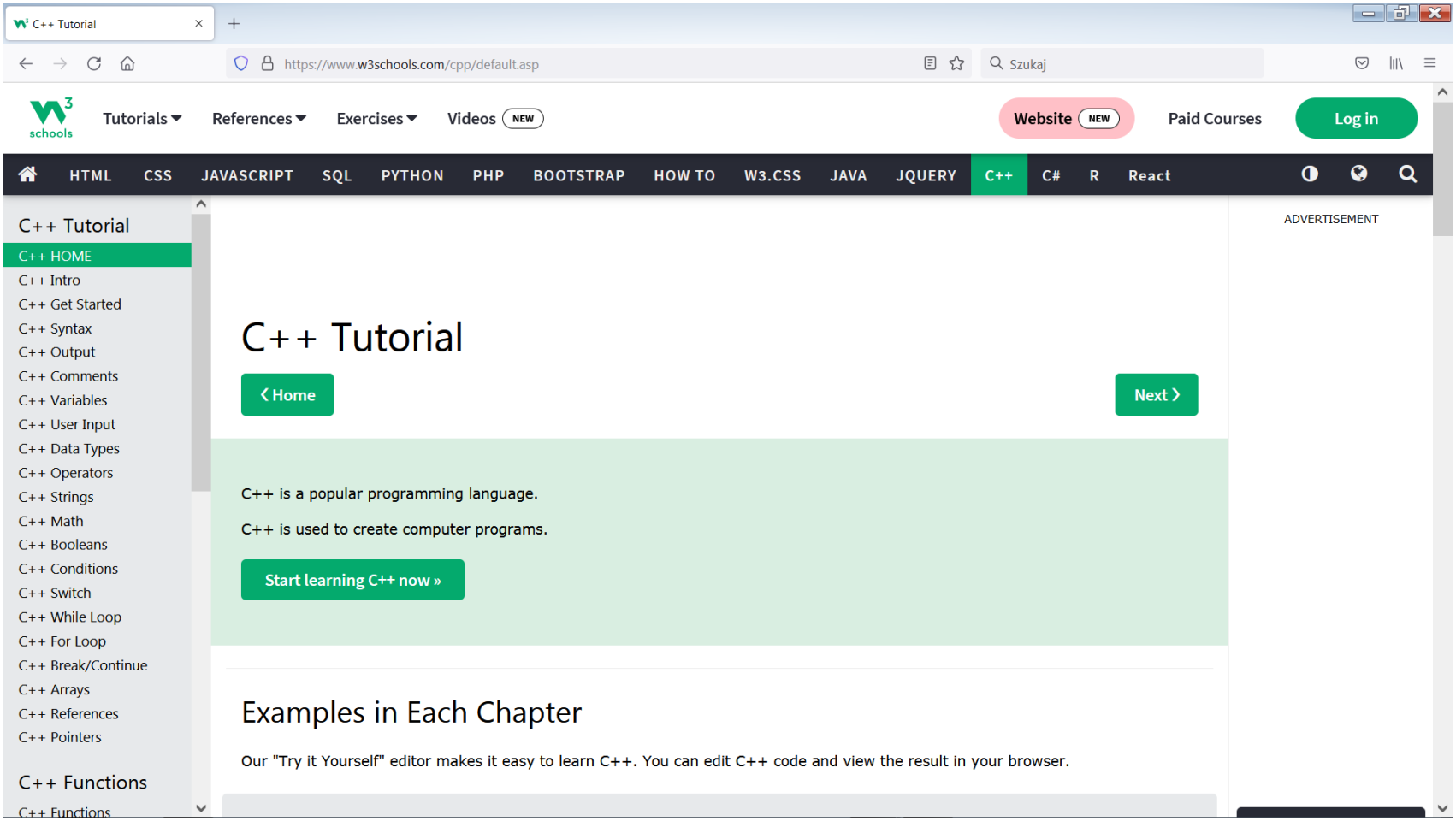
https://studio.code.org/hoc/1

Scratch



https://scratch.mit.edu/

www.w3schools.com

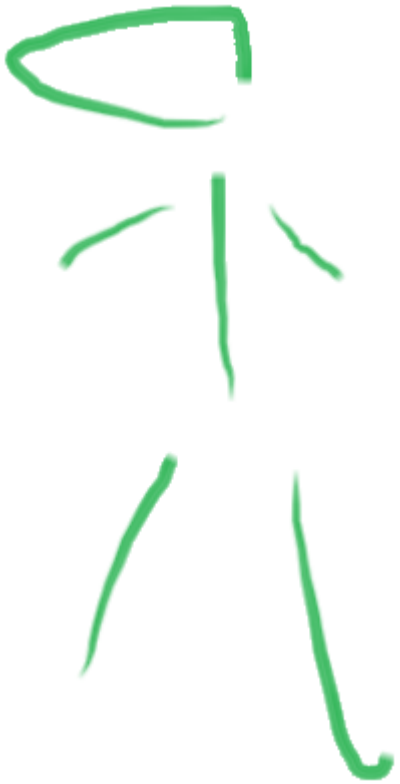


www.w3schools.com

Zadanie

System operacyjny

program



Żona mówi do męża:
„Przygotuj śniadanie dziecku”

Mąż:
„dobrze”

Zmienne

dane

Mąż poszedł na zakupy

zakupy

zmienne zainicjowane

półka warzywa=„pomidory”

półka chleb=„chleb_wiejski”

półka dżem=„dzem_sliwkowy”

Lodówka pełna

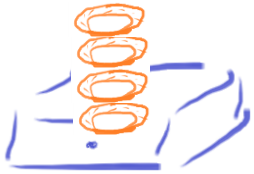
zmienne

Lodówka pusta

Program

Książka Kucharska

Szuflada z naczyniami



Szuflada ze sztućcami

dodatkowe narzędzia

Algorytmy

program w komputerze

Lodówka pełna

półka warzywa=„pomidory”

półka chleb=„chleb_wiejski”

półka dżem=„dzem_sliwkowy”

Zmienne w pamięci komputera

wynik programu

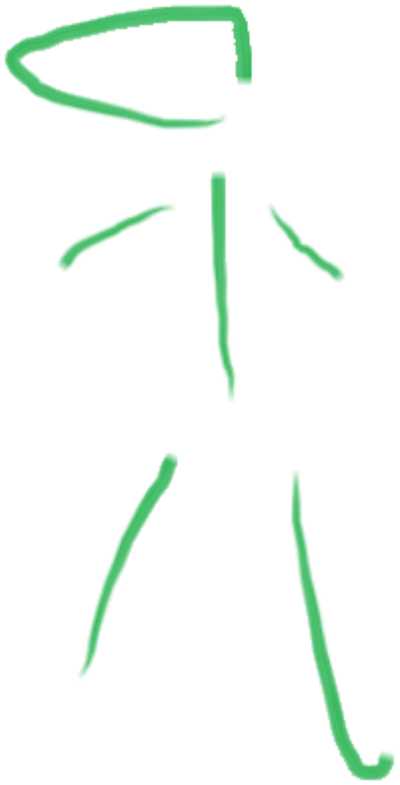
Tata przygotowuje posiłek wg. Książki Kucharskiej korzystając z talerza i noża

kanapka dla dziecka

Zadanie wykonane

System operacyjny

program



return 0

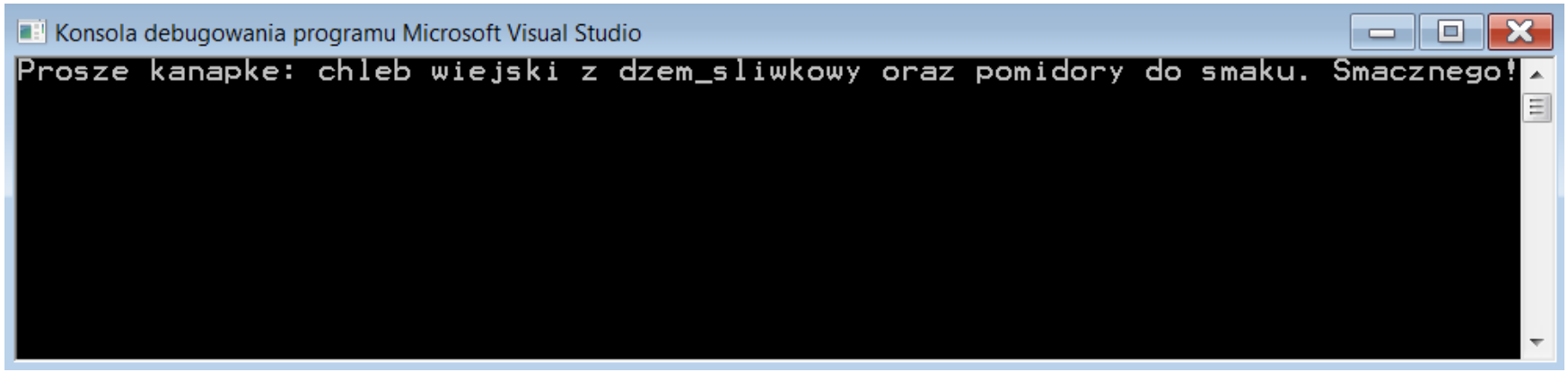
Mąż:
„Synek zjadł kanapkę i nie zwymiotował”

Żona mówi do męża:
„super”

Program kanapka

```cpp
#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 << "Prosze kanapke: " << kanapka << " oraz " << warzywa << " do smaku. Smacznego!" << endl;

    // dziecko po zjedzeniu kanapki nie zwymiotowało
    return 0;
}
```

Program kanapka



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

https://www.stroustrup.com/

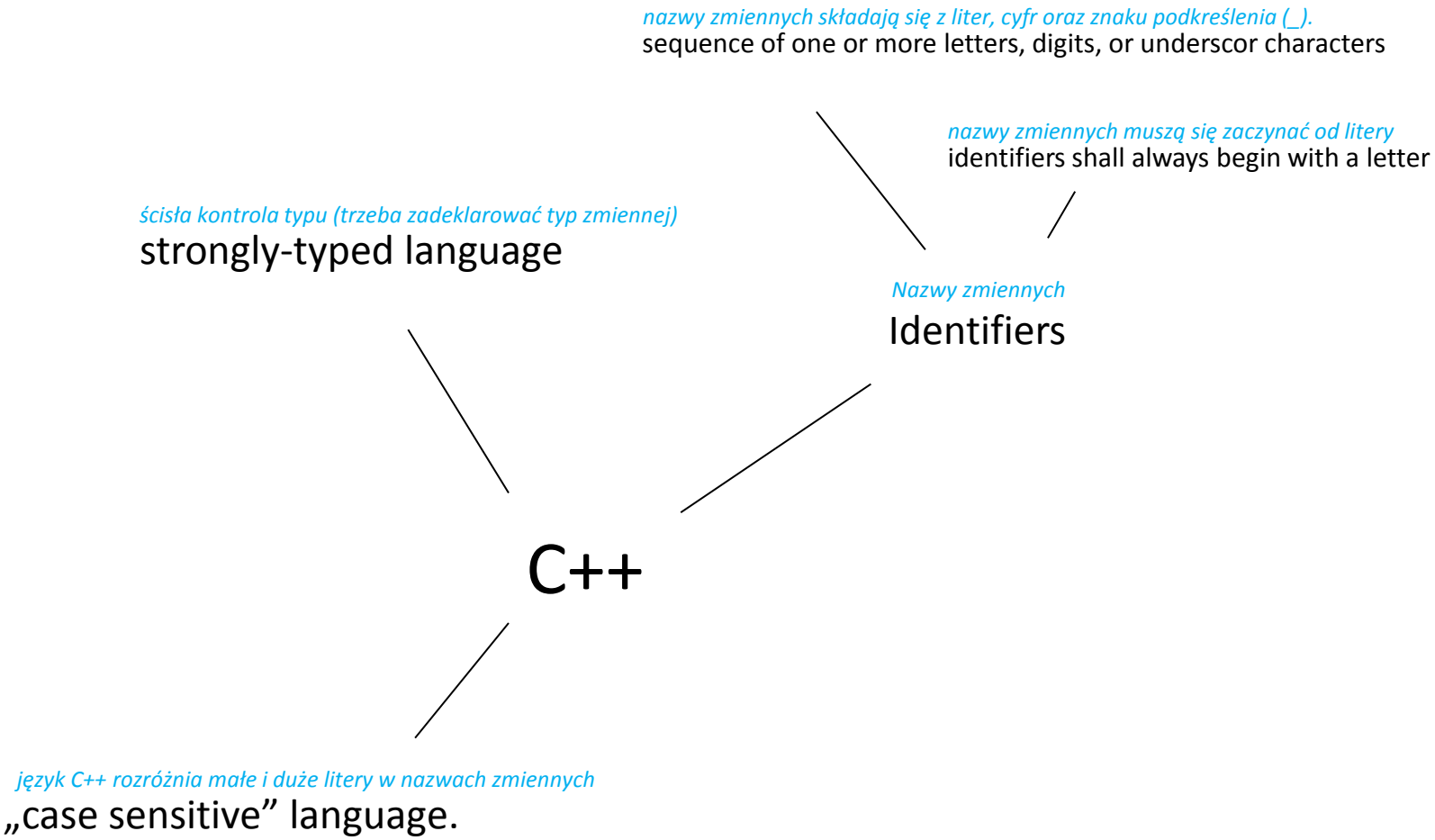
C++ standards

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

https://en.wikipedia.org/wiki/C%2B%2B#cite_note-isocpp1998-29

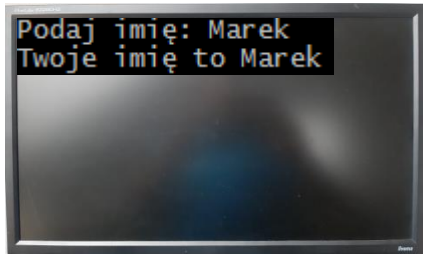

nazwy zmiennych składają się z liter, cyfr oraz znaku podkreślenia (_).
sequence of one or more letters, digits, or underscor characters
nazwy zmiennych muszą się zaczynać od litery
identifiers shall always begin with a letter
ścisła kontrola typu (trzeba zadeklarować typ zmiennej)
strongly-typed language
Nazwy zmiennych
Identifiers
C++
język C++ rozróżnia małe i duże litery w nazwach zmiennych
„case sensitive” language.


biblioteka
directive for the preprocessor
dyrektywa preprocesora
(dołączenie pliku iostream - rozszerzenie możliwosci c++)
#include<iostream>
standardowa przestrzeń nazw
using namespace std;
(rozwój c++ - dużo bibliotek – brakuje unikatowych nazw zmiennych i funkcji => wymyślono przestrzeń nazw)
typ zwracanej wartości
funkcja main
int main() {
instrukcja
komentarz
cout<<"Hello world!"<<endl;
//wypisz na konsoli
standardowy strumień wyjściowy
ciąg znaków
operator wyjścia
(przeciążony operator przesunięcia)
end of line
return 0;
}
instrukcja zwracająca zero
(informacja dla środowiska systemowego, że program zakończył się poprawnie)

Standardowe wejście/wyjście

standardowe urządzenie wyjściowe

standardowe urządzenie wejściowe




```
#include <iostream>

using namespace std;

int main()
{
    string imie;

    cout << "Podaj imię: ";

    cin >> imie;

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

    return 0;
}
```

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

insertion operator
operator wstawiania

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

extraction operator
operator wyodrębnienia

Standardowe typy danych

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

* 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
```

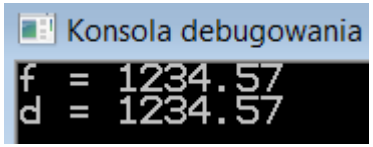
program zmienne

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

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

float, double - precision

```cpp
#include <iostream>

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

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

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

}
```



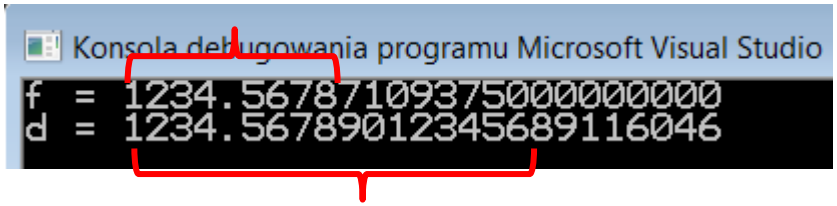
kompilator zaokrąglił liczbę do 6 cyfr (default)

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

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

float, double - precision

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

using namespace std;

int main(){


    // bez f na końcu literału liczba byłaby traktowana jak double
    // i konwerowana 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;
}
```



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:

liczba okresowa, nieskończona

$$0.1_{10} => 000(1100)_2$$

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

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_hbLZyIP0AhVHAxAIHVW7AJwQ6AF6BAgeEAM#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
```

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

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 ;

i = s;
```

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

promocja typu (wartość s jest promowana z short do int, z typu mniejszego do wiekszego)
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 spowodawała 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};  // inicjalizaja 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;  // c-like cast notation
```
rzutowanie w stylu języka C

lub

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

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;          //deklaracja i jednocześnie definicja
                //zmiennej a typu integer
```

nazwa

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

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

Inicjalizacja kopiująca
copy initialization

Inicjalizacja bezpośrednia
direct initialization

Inicjalizacja konstruktorowa
constructor initialization

int a = 1;

int a(1);

to samo

auto a = 1;

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 klamrowa
brace initialization

Inicjalizacja jednolita
uniform initialization

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

int a{1};

można też tak

int a = {1};

https://en.cppreference.com/w/cpp/language/initialization

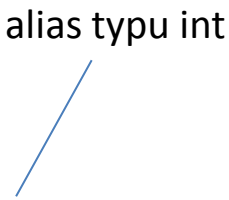
inicjalizacja klamrowa

przykłady

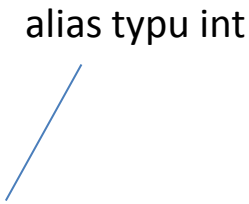
```
int a = {25};
double d = {13.5};
short tab[3] {5,7,9};
int * ptr = new int[3] {20, 30, 40};
```

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

Aliasy – nazwy zastępcze zmiennych



typedef int *integer*;
integer a = 1;



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 całkowiety*
  - 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 // 3.14x10^2 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*

https://en.cppreference.com/w/cpp/language/expressions
https://www.cplusplus.com/doc/tutorial/constants/

escape sequences

Znaki specjalne

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

Znaki specjalne

znak nowego wiersza

program znakiSpecjalne

```
cout << "znaki specjalne\n\n";
cout << "cudzysłów: \\\"\n";
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";
```

```
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 złożonego

operator przypisania złożonego
compound assignment operator

```
a += 2;          // to samo co a=a+2;
```

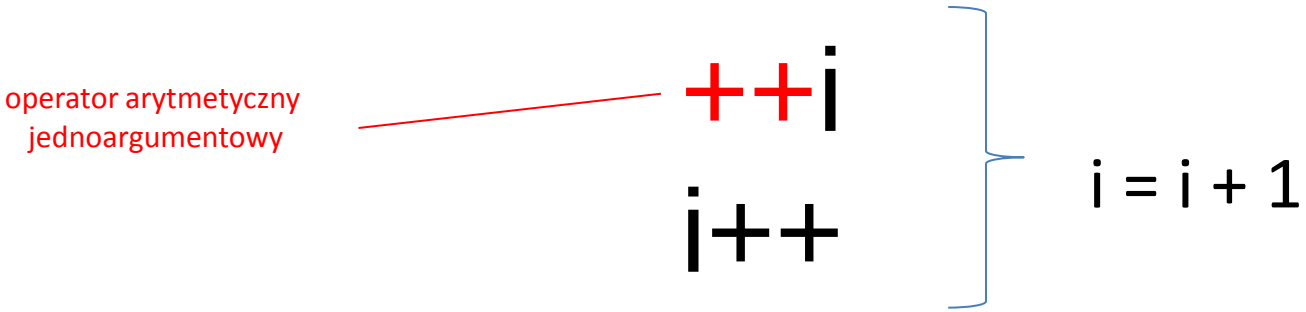
literał całkowity
literal constant integer

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	*addition*
-	odejmowanie	*subtraction*
*	mnożenie	*multiplication*
/	dzielenie	*division*
/	dzielenie całkowite np.: 10/4=2 (dzielimy 2 liczby całkowite – iloraz jest zaokrąglany do liczby całkowitej)	
%	modulo – reszta z dzielenie liczb całkowitych	*modulo*

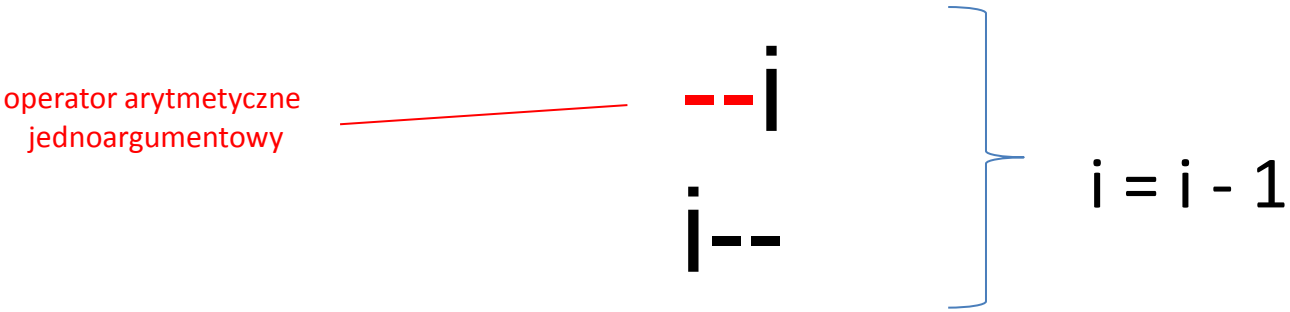
Operatory preinkrementacji i postinkrementacji




```
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




```
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
```

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, ...).

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

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; // rożne priorytety, uwaga na postinkrementację!
cout << liczba;  //?
```

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

Operatory bitowe *Bitwise operators*

&	Operator bitowy AND	*Bitwise AND*
|	Operator bitowy OR	*Bitwise inclusive OR*
^	Operator bitowy XOR	*Bitwise exclusive OR*
~	Operator bitowy NOT	*Unary complement (bit inversion)*
>>	Operator przesunięcia bitowego w prawo	*Shift bits right*
<<	Operator przesunięcia bitowego w lewo	*Shift bits left*

Operator bitowy AND

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

6 -> 110 — system binarny

4 -> 100

system dziesiętny

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

Tablica prawdy dla AND

```
  110
 &100
 ----
  100
```

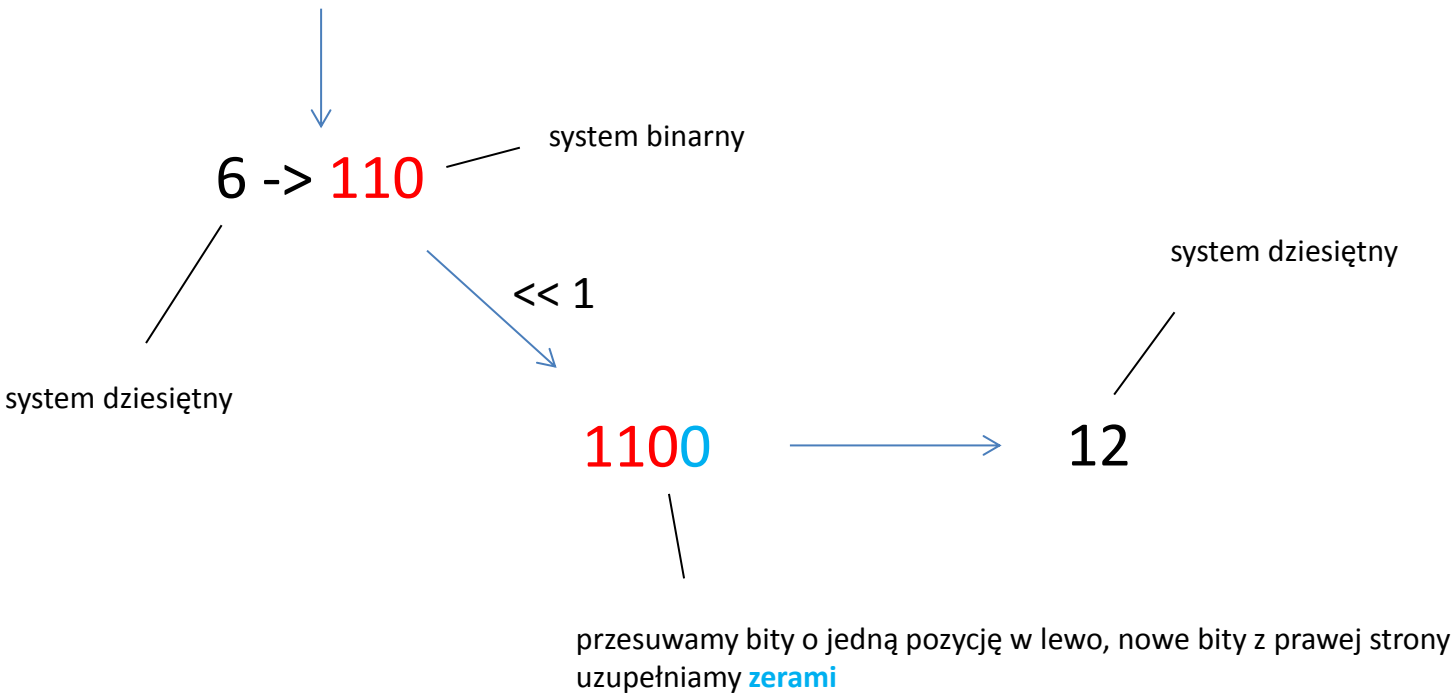
system dziesiętny

4

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

Operator przesunięcia bitowego w lewo

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



Operatory relacji (porównania)

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

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

grupowanie od lewej do prawej

```
int a, b;

a = 1;
b = (++a, a*20, a + 60);

cout << b;    // 62

return 0;
```

do b przypisywany jest
wynik ostatniego wyrażenia

Przestrzenie nazw *Namespaces*

```cpp
#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

https://en.cppreference.com/w/cpp/language/namespace

keywords

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

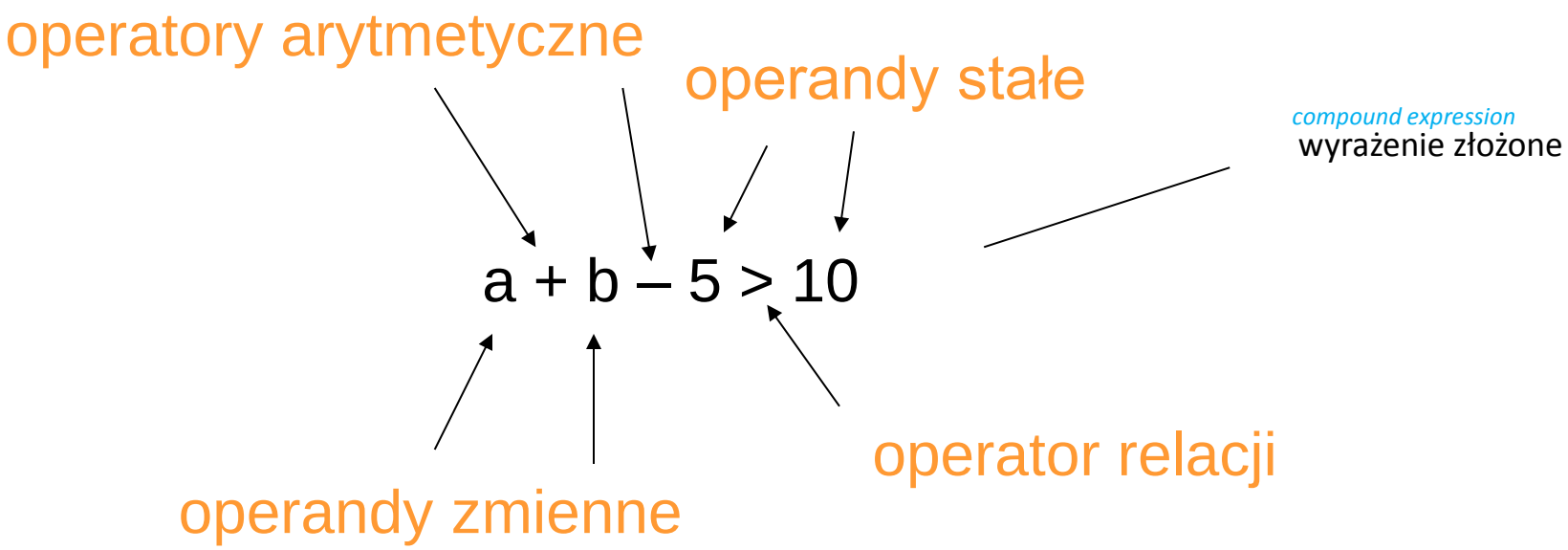
Tych słów nie można używać jako nazw zmiennych!

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

INSTRUKCJE STERUJĄCE

Wyrażenie *expression*

Sekwencja operatorów i operandów określająca obliczenie.



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

warunek

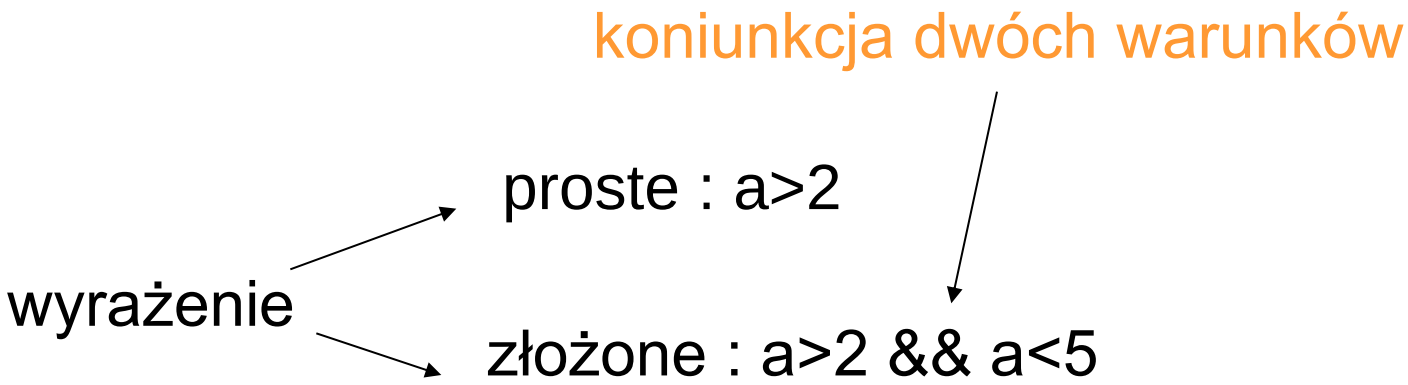
Jeśli warunek jest prawdziwy
(ma wartość true)
wykonywana jest instrukcja.
W przeciwnym wypadku
instrukcja nie jest wykonywana.

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

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*



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
```

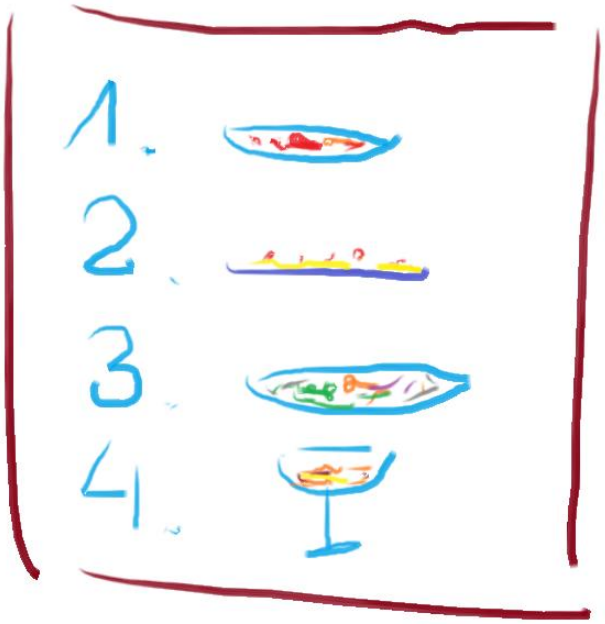
warunek

```
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



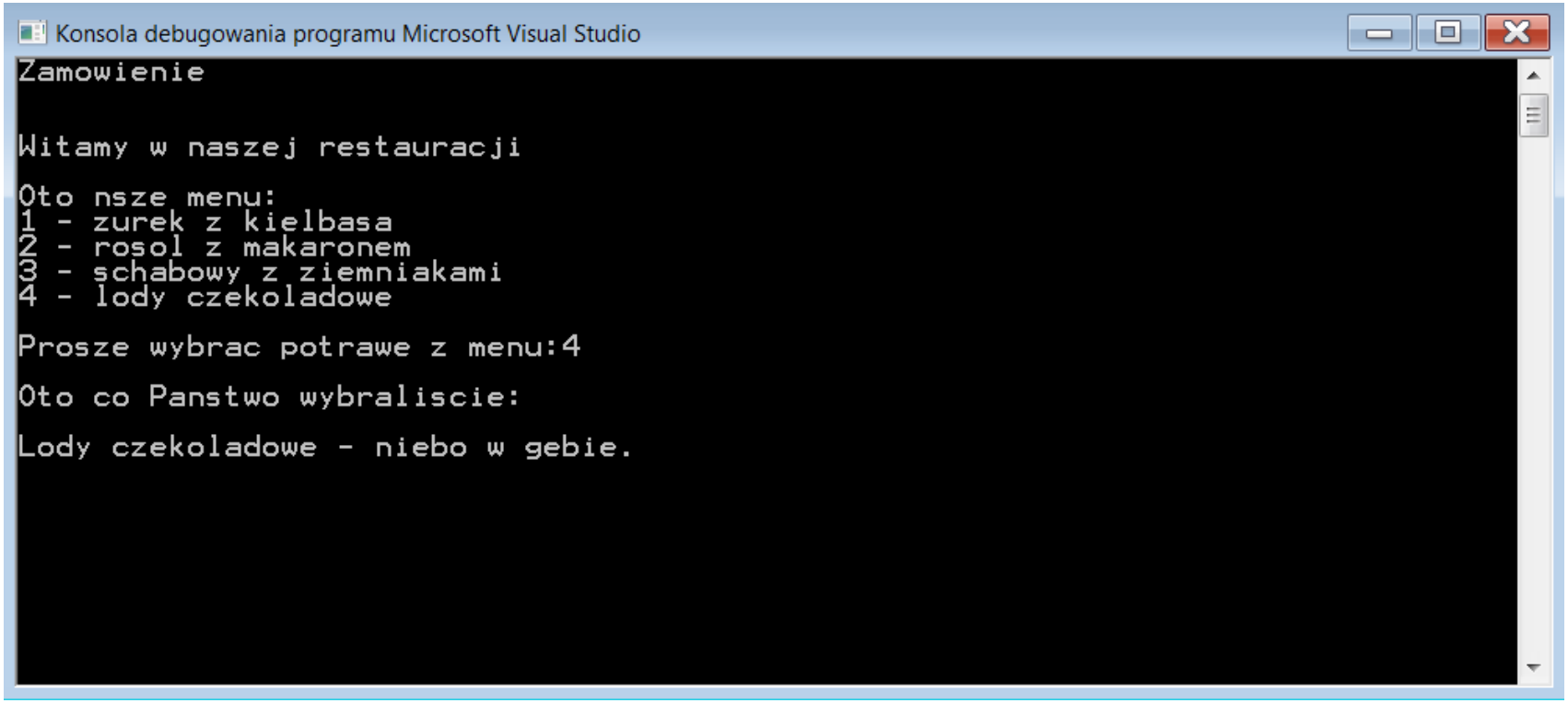
Menu

switch

Kelner

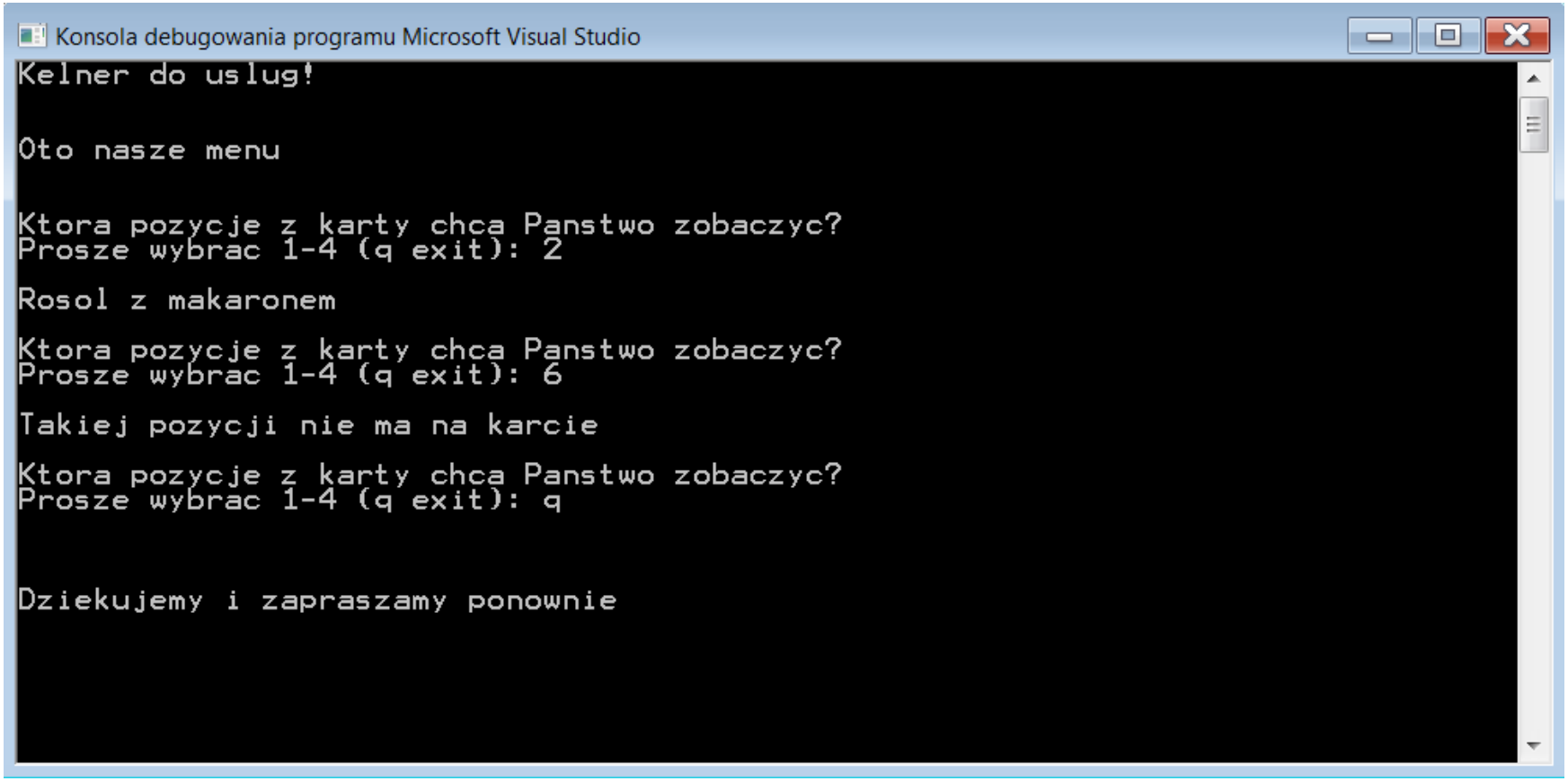
zadanie

zamowienie



zadanie

menu



Pętla for

Przed pierwszym uruchomieniem pętli wykonywana jest ta instrukcja

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

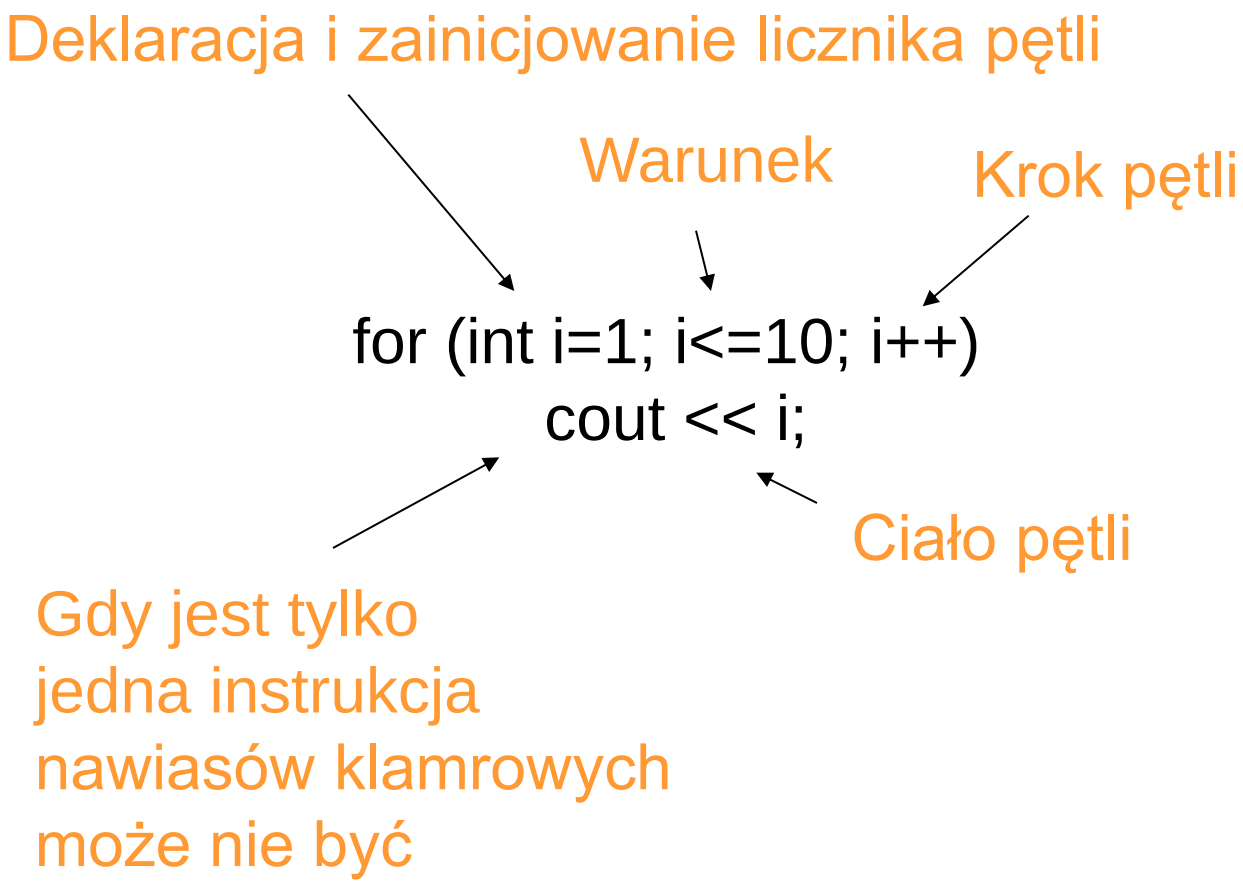
Gdy jest tylko jedna instrukcja nawiasów może nie być

Instrukcja wykonywana każdorazowo na zakończenie obiegu pętli

```
for (int i=1; i<=10; i++)
{
   cout<<"Licznik pętli: ";
   cout << i << endl;
}
```

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 i=1; i<=10; i++)
{
    cout << i;
}
```

1 2 4 3

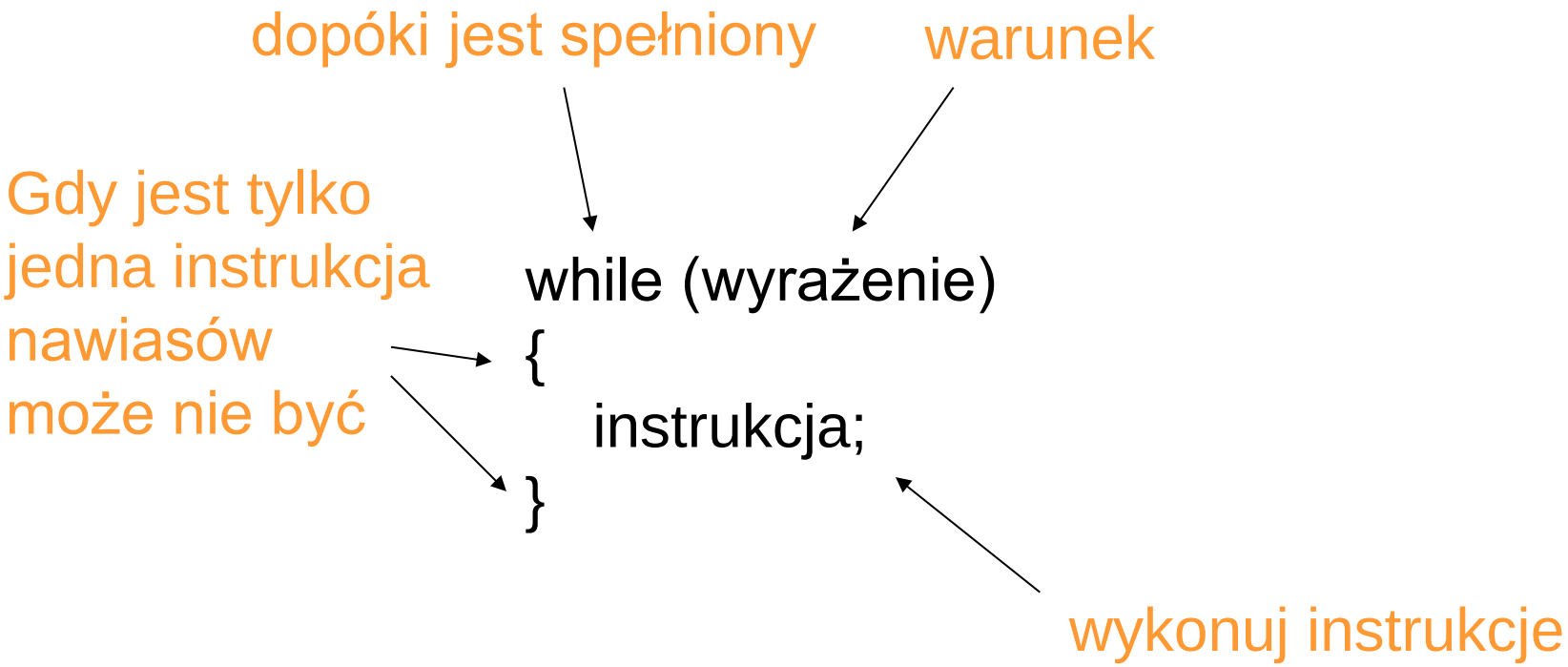
pierwszy obieg pętli 1234

drugi obieg pętli 234

trzeci obieg pętli 234

kolejne obiegi pętli 234

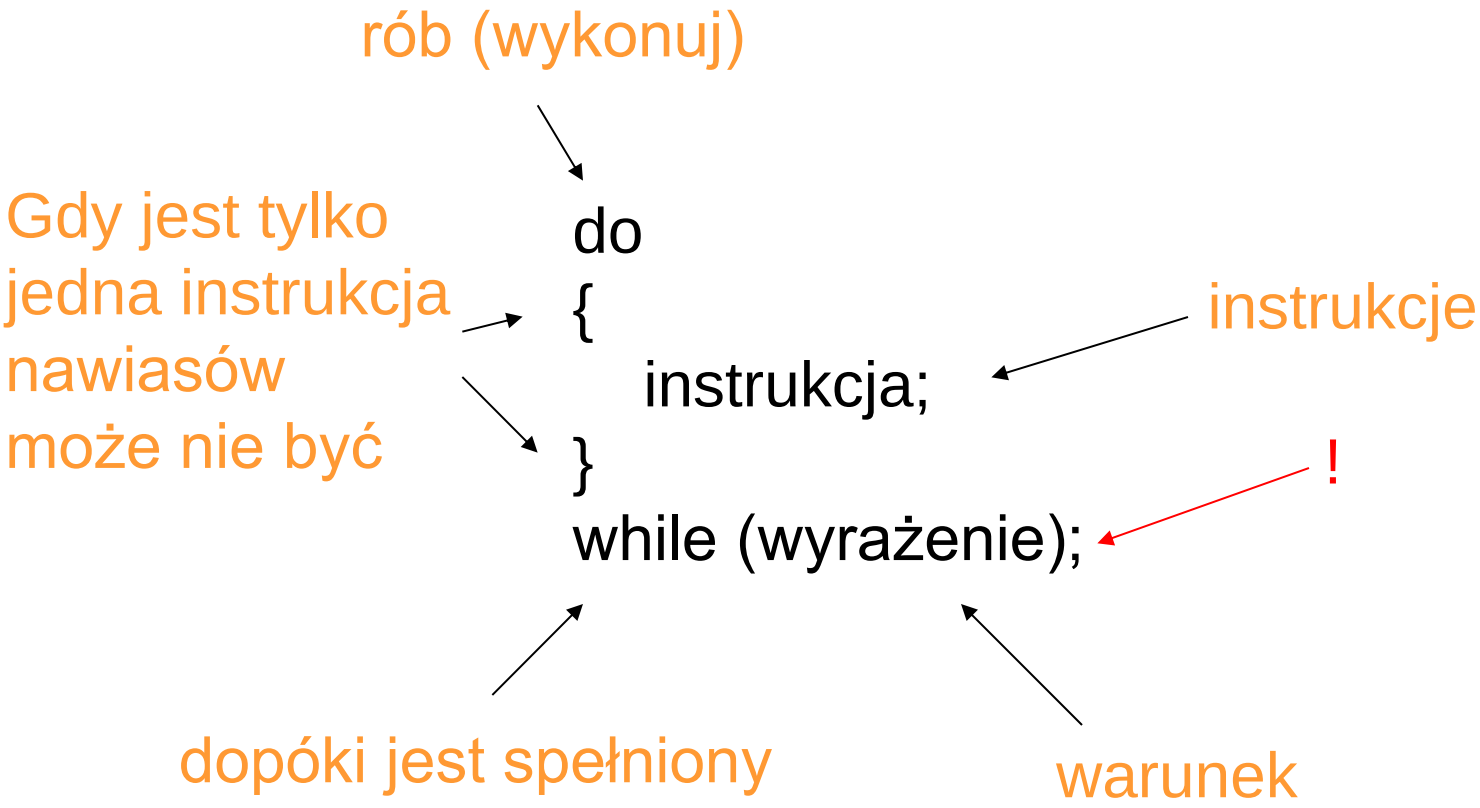
Pętla while



wyrażenie może przyjmować wartość logiczną:

- prawda
- fałsz

Pętla do while



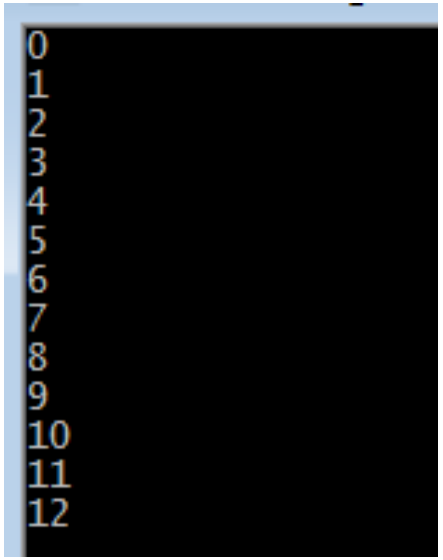
wyrażenie może przyjmować wartość logiczną:

- prawda
- fałsz

Instrukcja break

Pętla nieskończona while

```
int licznik = 0;

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

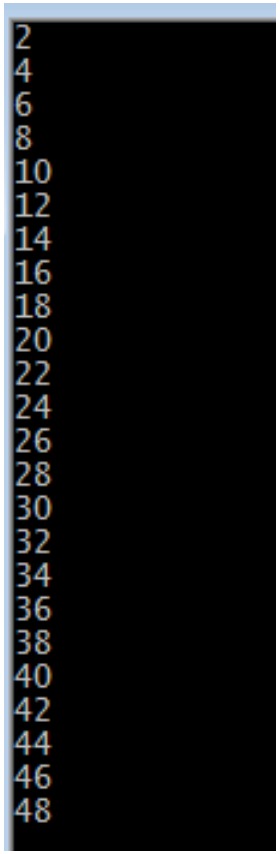


break przerywa wykonywanie pętli

Instrukcja continue

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

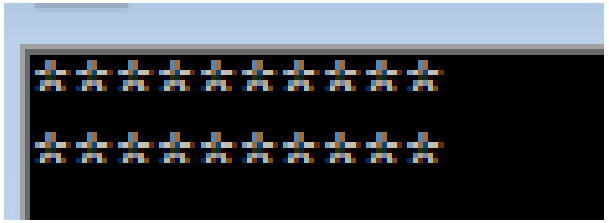
```
for (int i = 1; i < 50; i++)
{
    if (i % 2)
        continue;
    cout << i << endl;
}
```



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

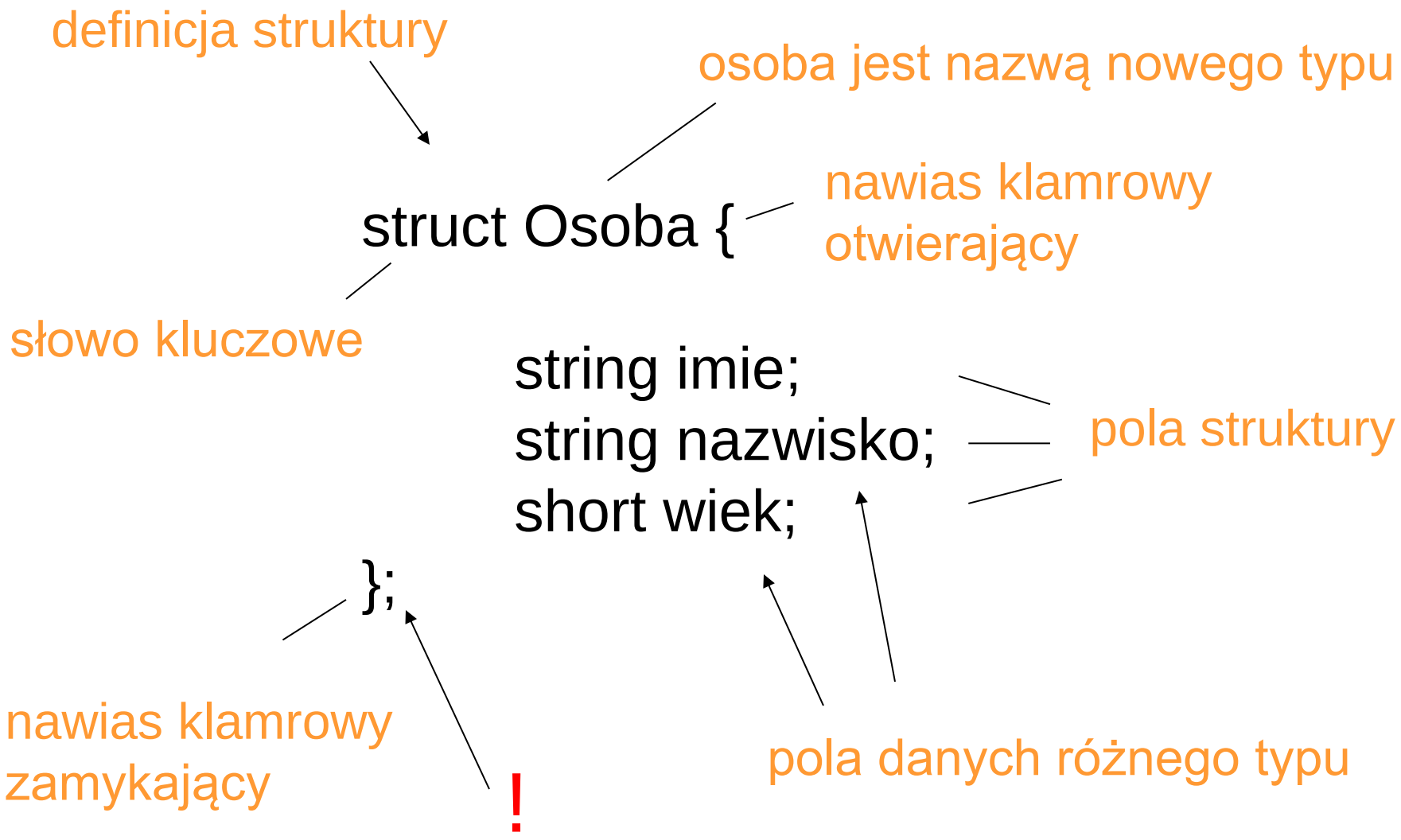
Zagnieżdżenie pętli

```
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
nawias klamrowy otwierający
struct Osoba {
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;
```

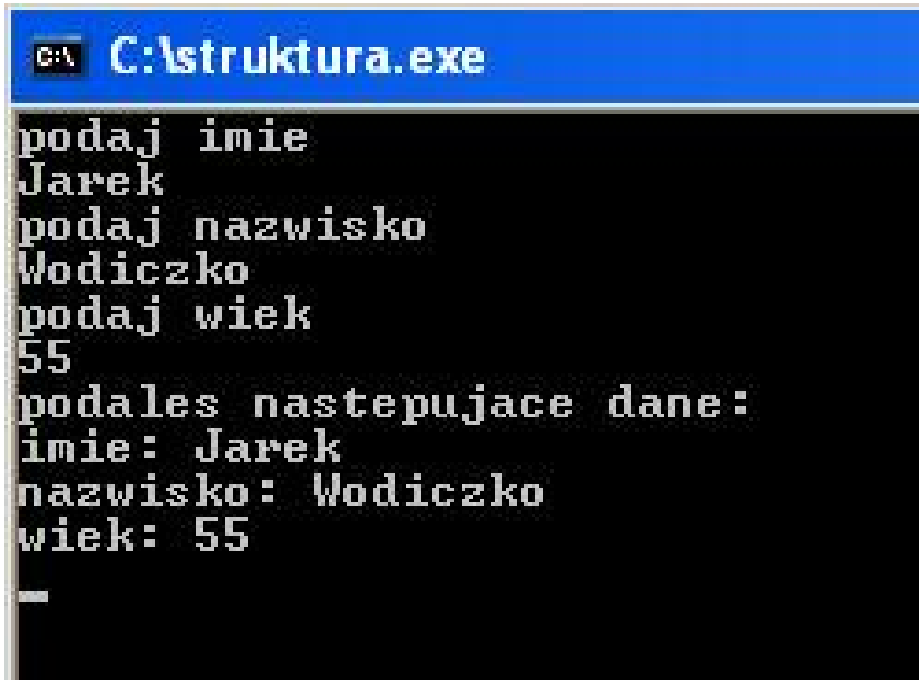
Struktury

```cpp
#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;
}
```

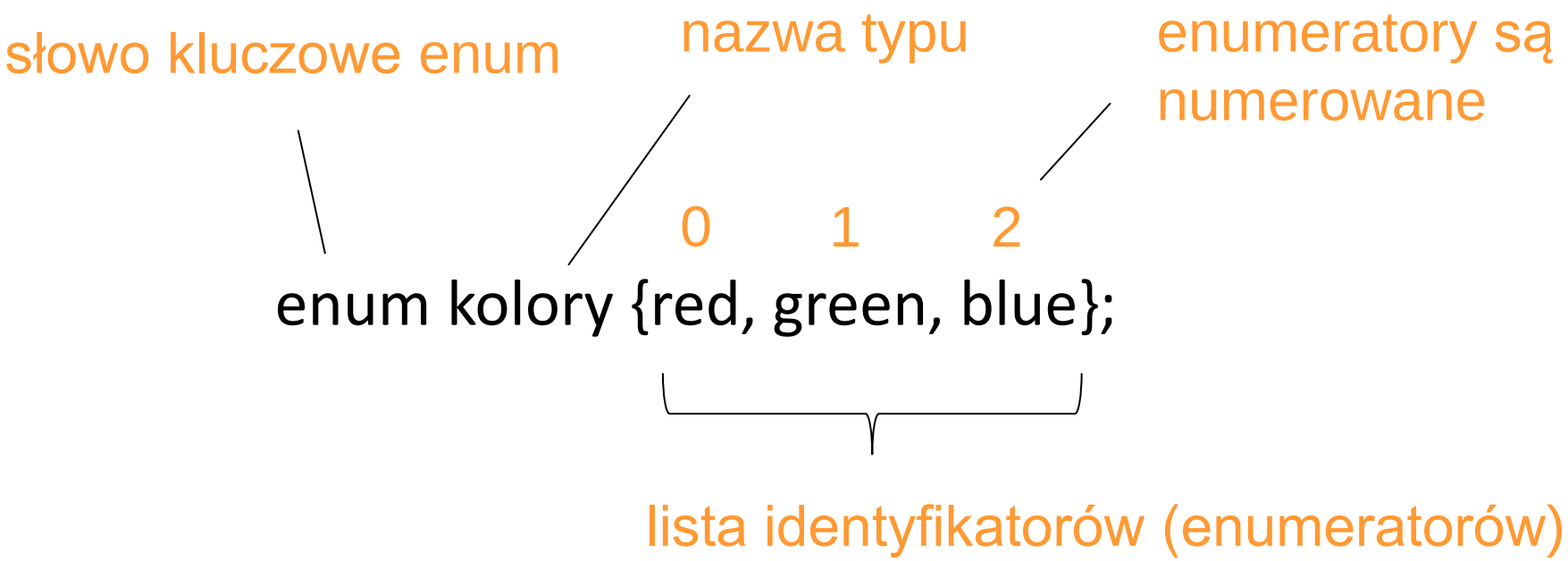
zadanie

Struktury

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



Typ wyliczeniowy enum



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,
    Wiktoria,
    Beata
};

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

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

nie można duplikować identyfikatorów

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

Enum class

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

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

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
}
```

Visual Studio preferuje ten typ enum

identyfikator z operatorem zakresu

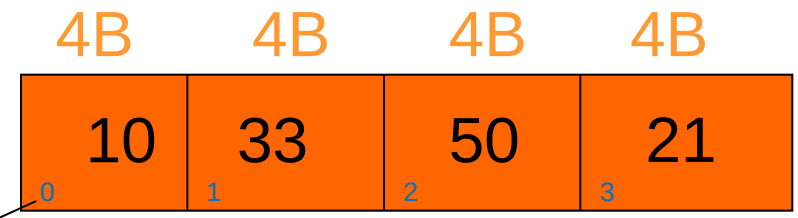
Tablica jednowymiarowa (statyczna)

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

indeks

operator indeksowania

tab[0]

tab[1]

tab[2]

tab[3]

odwołanie się do elementów tablicy

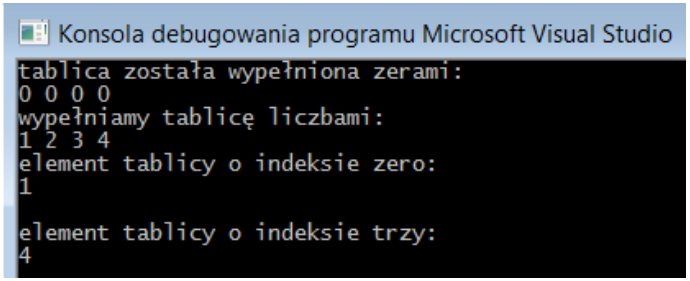
Tablica jednowymiarowa (statyczna)

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;
```

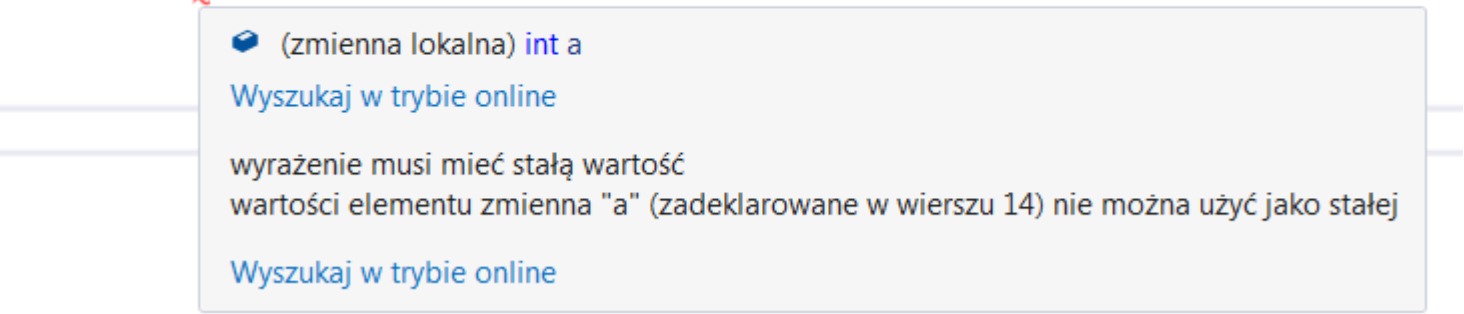


Tablica jednowymiarowa (statyczna)

fix array
compile-time array

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

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



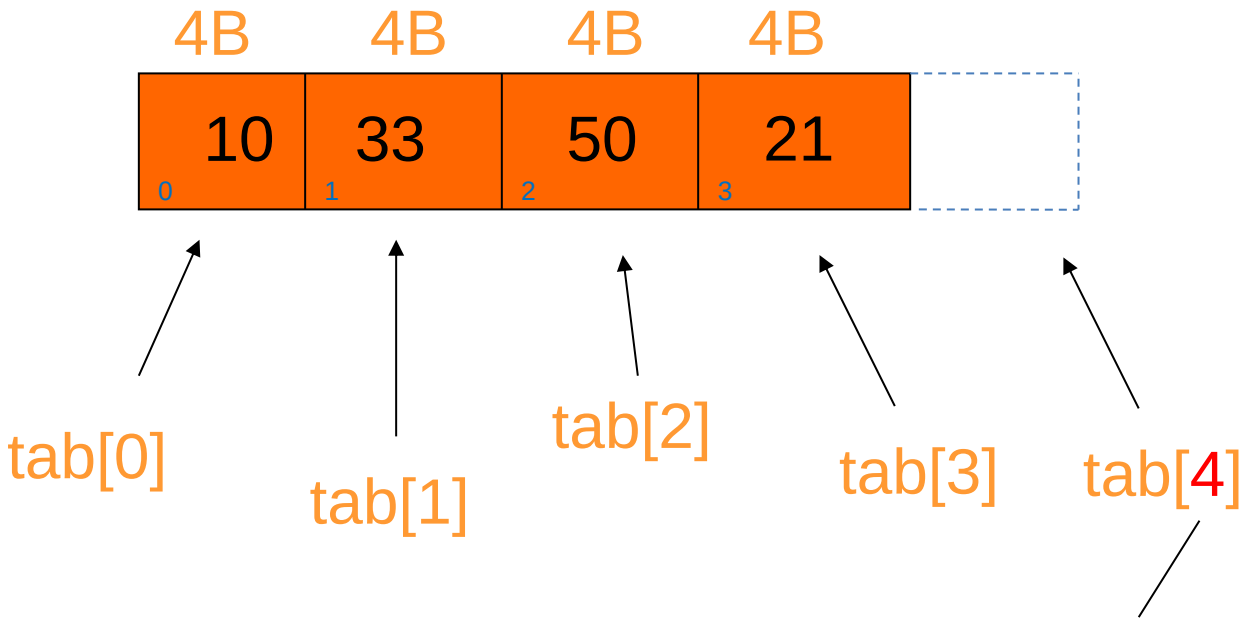
OK - wielkość tablicy jest stała w czasie kompilacji

```
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;
```

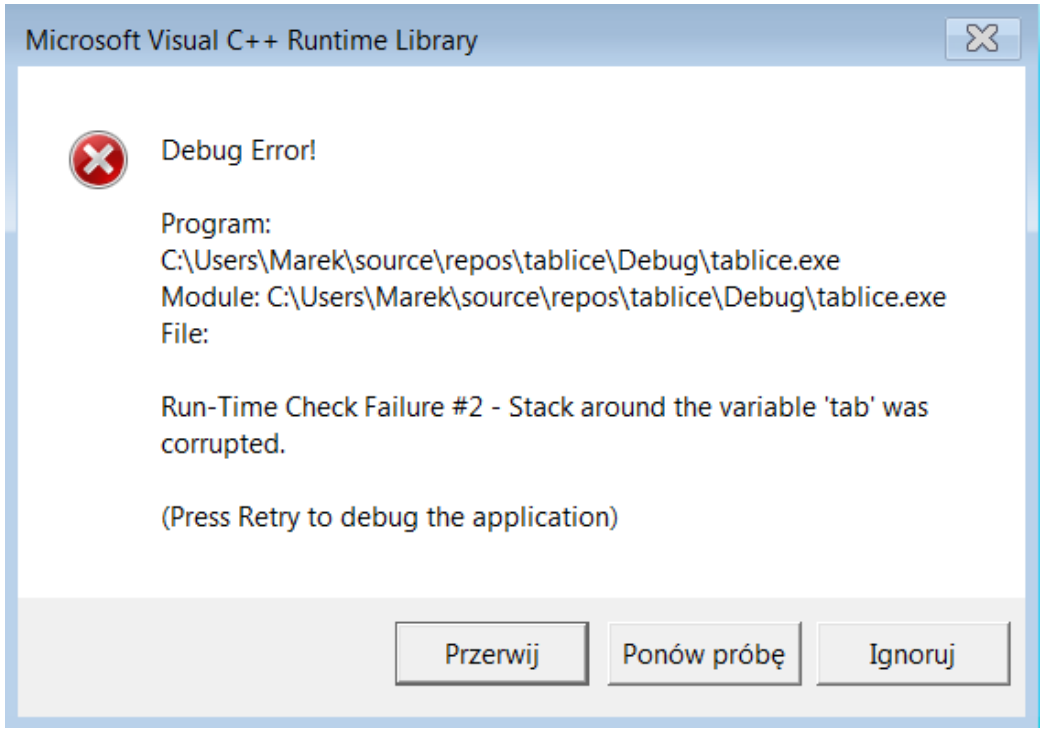
(int)33

Wyszukaj w trybie online

C6386: Przepełnienie 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



Tablica jednowymiarowa (statyczna)

zadanie

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

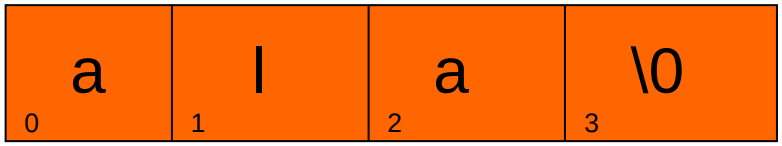
Tablica jednowymiarowa (statyczna)

C-string

C-napis

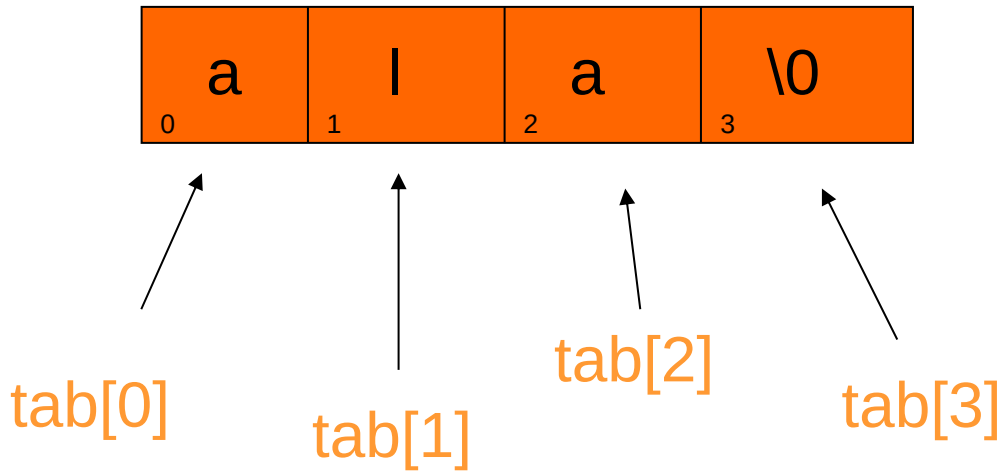
C –napis jest zakończony w pamięci znakiem specjalnym NUL *null character*

```
char slowo[4] = ”ala”;
```



tab[0] tab[1] tab[2] tab[3]

Tablica jednowymiarowa (statyczna)

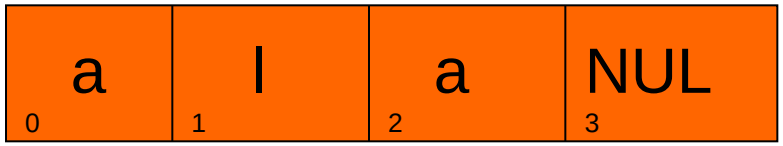



```
char slowo[4] = {”ala”};                    //alternatywnie
char slowo[ ] = {”ala”};                    //alternatywnie
char slowo[ ] {”ala”};                      //alternatywnie
char slowo[ ] =  {’a’,’l’,’a’,’\0’};        //alternatywnie
const char* slowo = ”ala”;                  //alternatywnie
char slowo[ ](”ala”);                       //alternatywnie
char slowo[4](”ala”);                       //alternatywnie

cout << slowo;                              //wypisanie słowa na konsoli
```

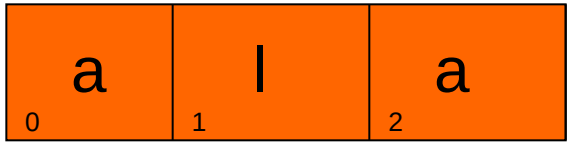
Tablica jednowymiarowa (statyczna)

Bajt zerowy

char slowo[]={”ala”};



char slowo[]={‘a’,’l’,’a’};



Tablica jednowymiarowa (dynamiczna)

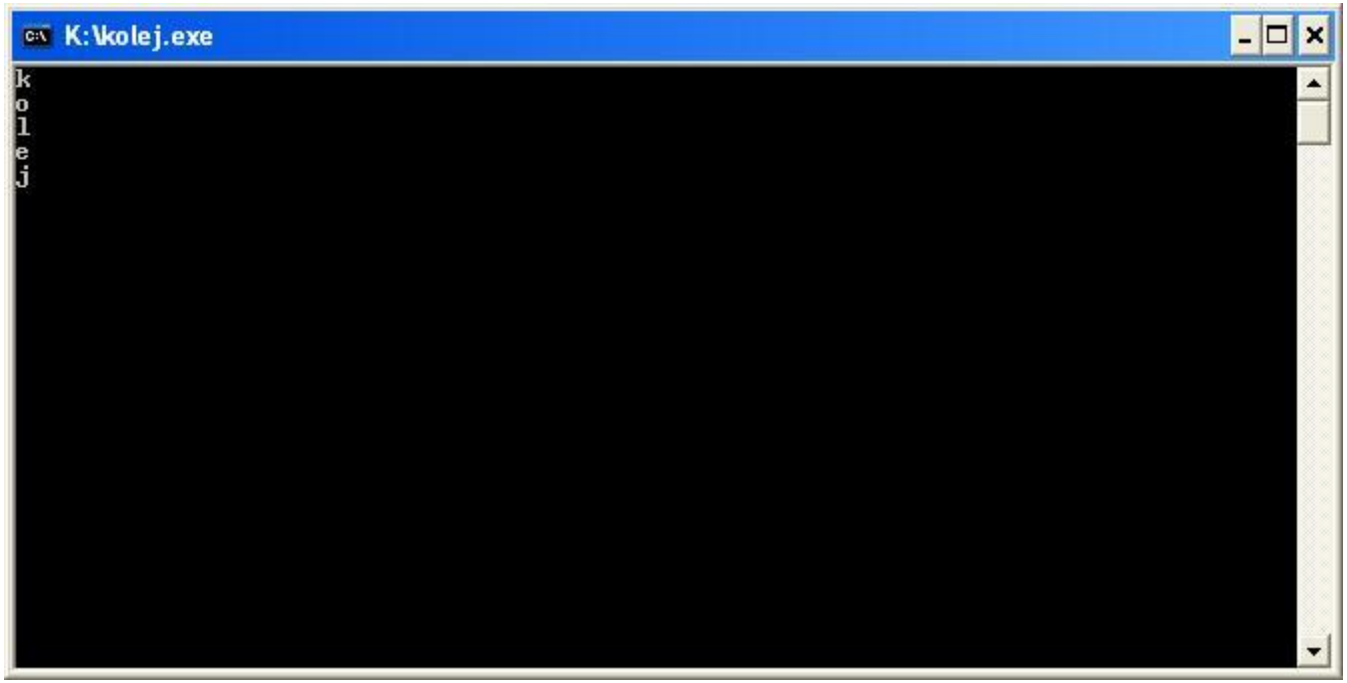
alokujemy tablicę
w trakcie działania
programu

```
// 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

// wypełnaimy 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

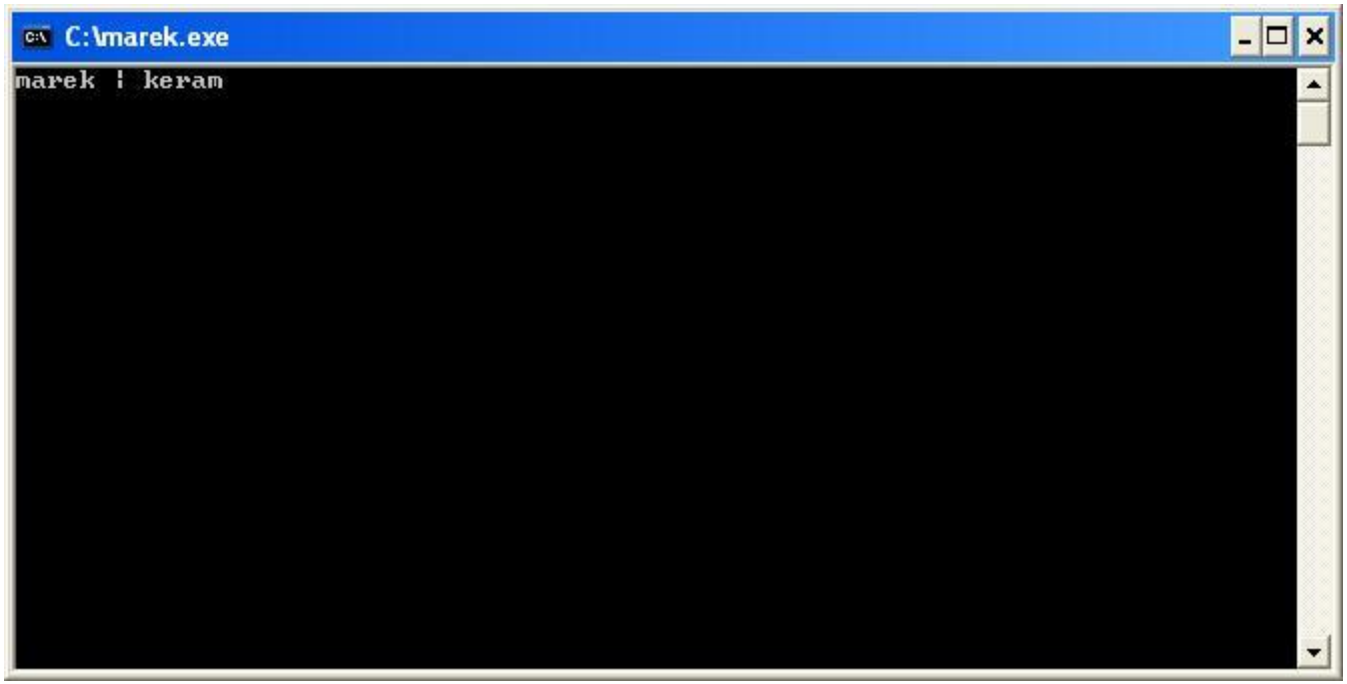
Wypisz słowo ***kolej*** na konsoli za pomocą pętli tak, aby kolejne litery były wypisane w nowych wierszach.



zadanie

Lustro

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



Tablica dwuwymiarowa (statyczna)

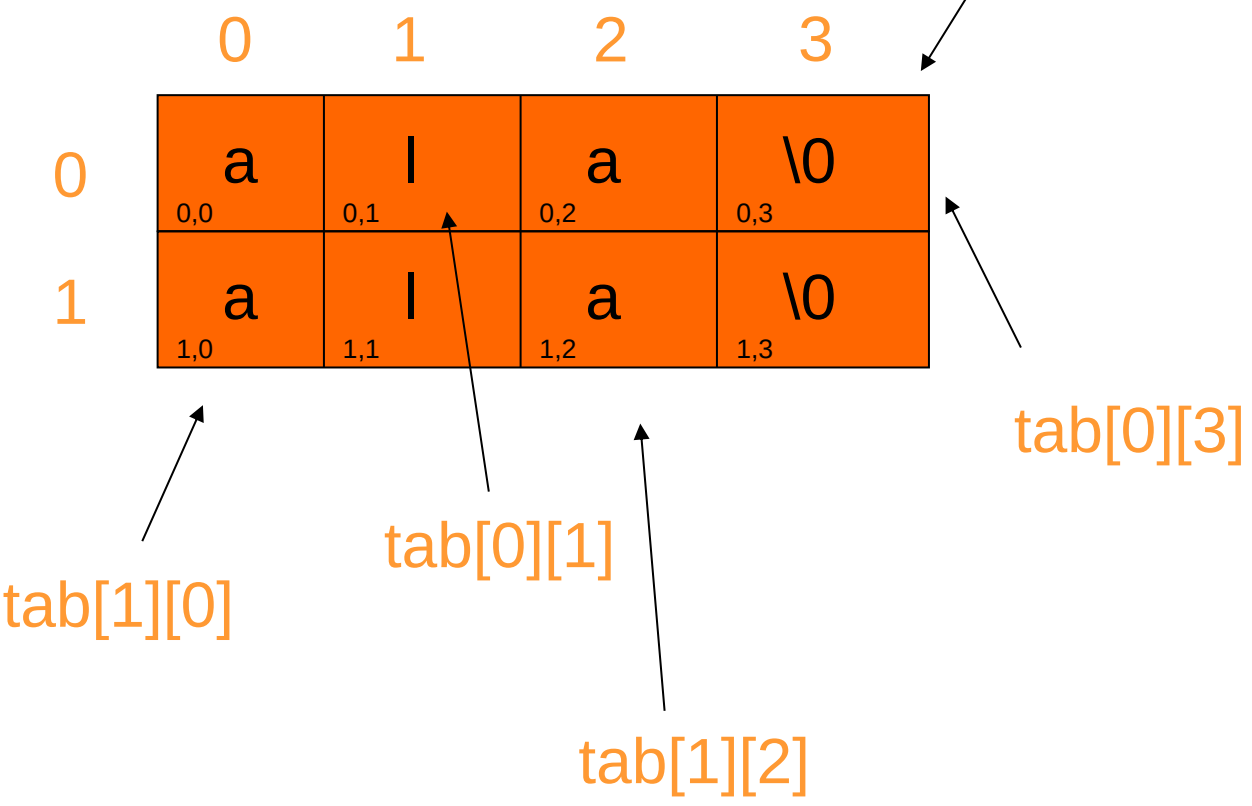
wiersze

kolumny

C-string
C-napis

char slowo[2][4] = {”ala”,”ala”};

null character
znak specjalny NUL



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;
}
```

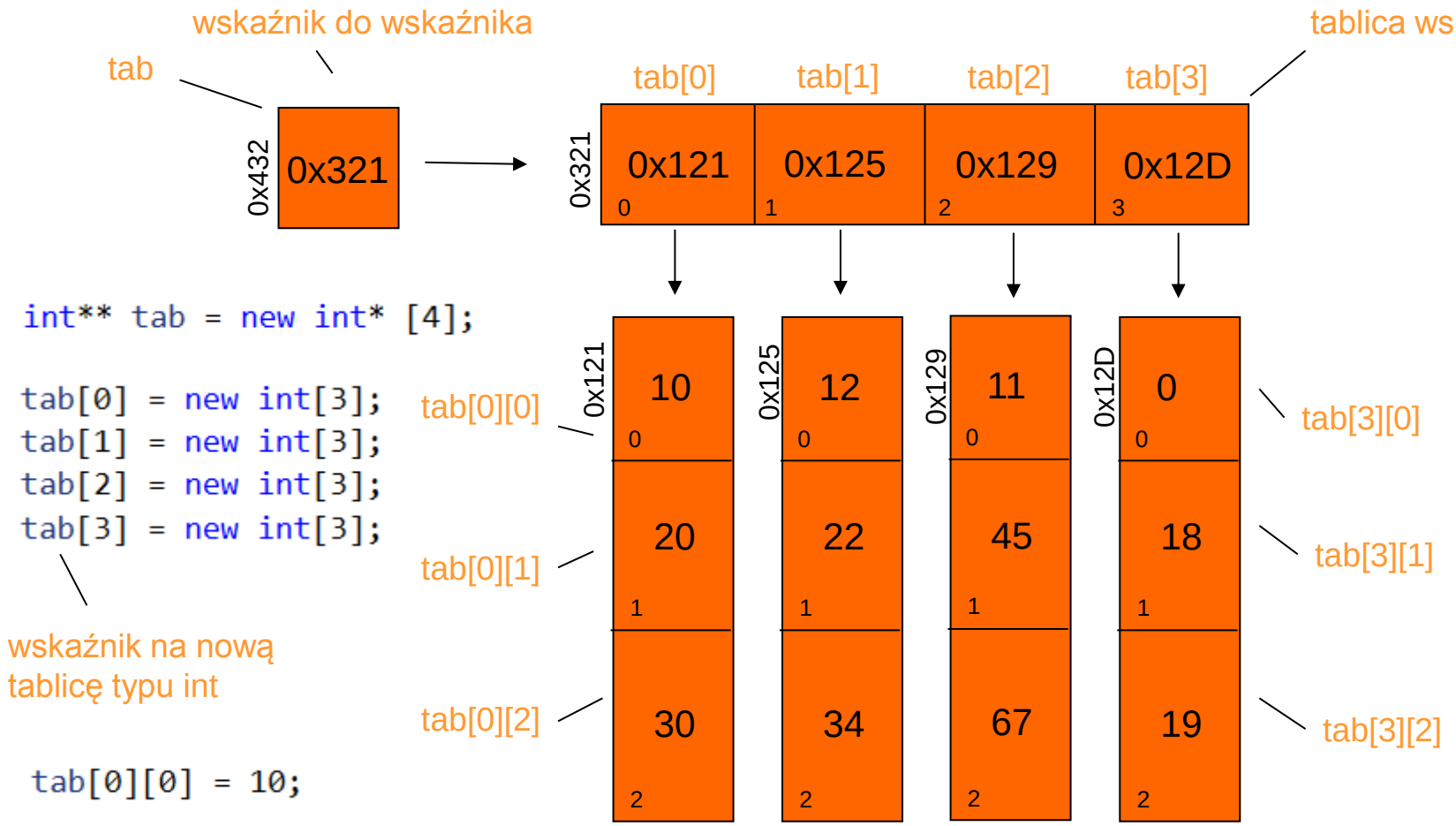
```
ala
ma
kota
```

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)



STRING

klasa stworzona w celu operowania napisami (ciągami tekstowymi)

Klasa string

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”;

//deklaracja, definicja i inizjalizacja
string imie = ”Malgosia”;
string imie (”Malgosia”);
```

łańcuch znaków (c-napis)

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.:

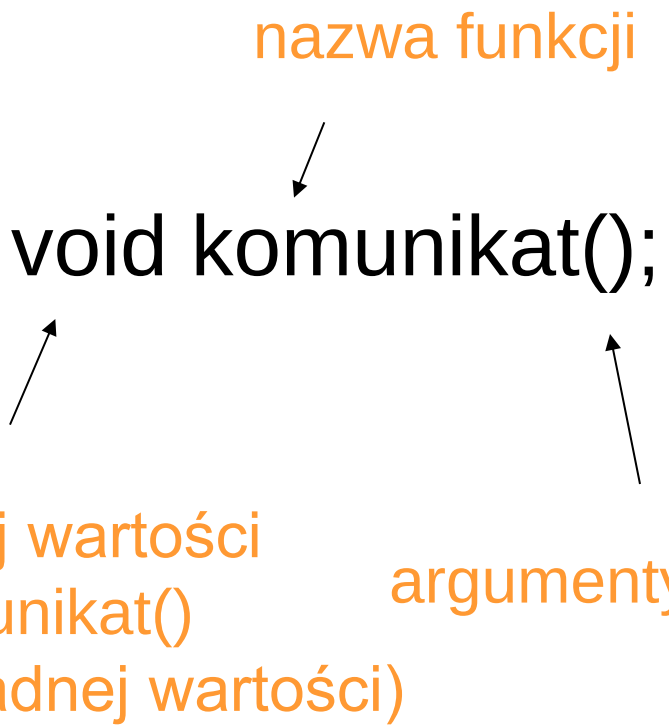
```
string imie;
cout<<”podaj imie”<<endl;
cin>>imie;
cout<<”Twoje imię zaczyna się na literę „<<imie[0];
```

FUNKCJE

Funkcja jest podprogramem.

Funkcje

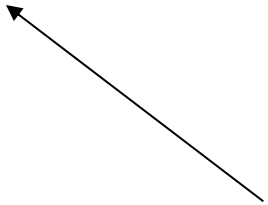
Deklaracja funkcji



Funkcje

Definicja funkcji

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



ciało funkcji

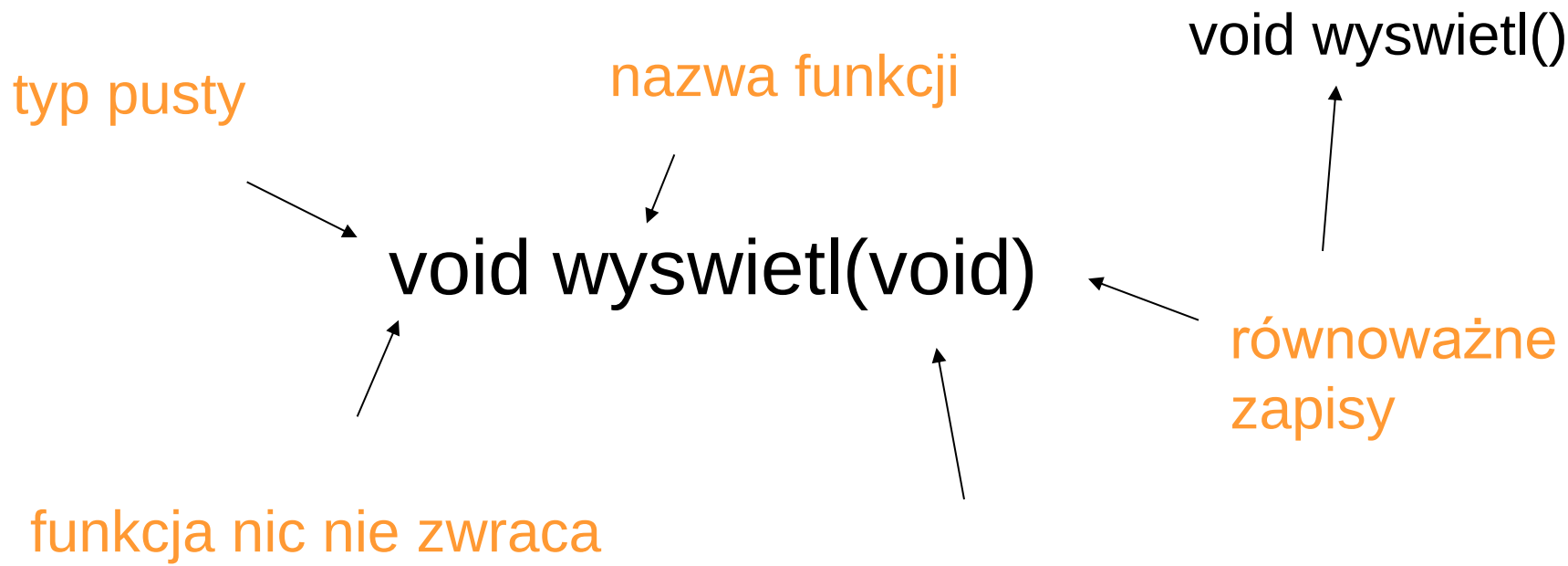
Funkcje

```
#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

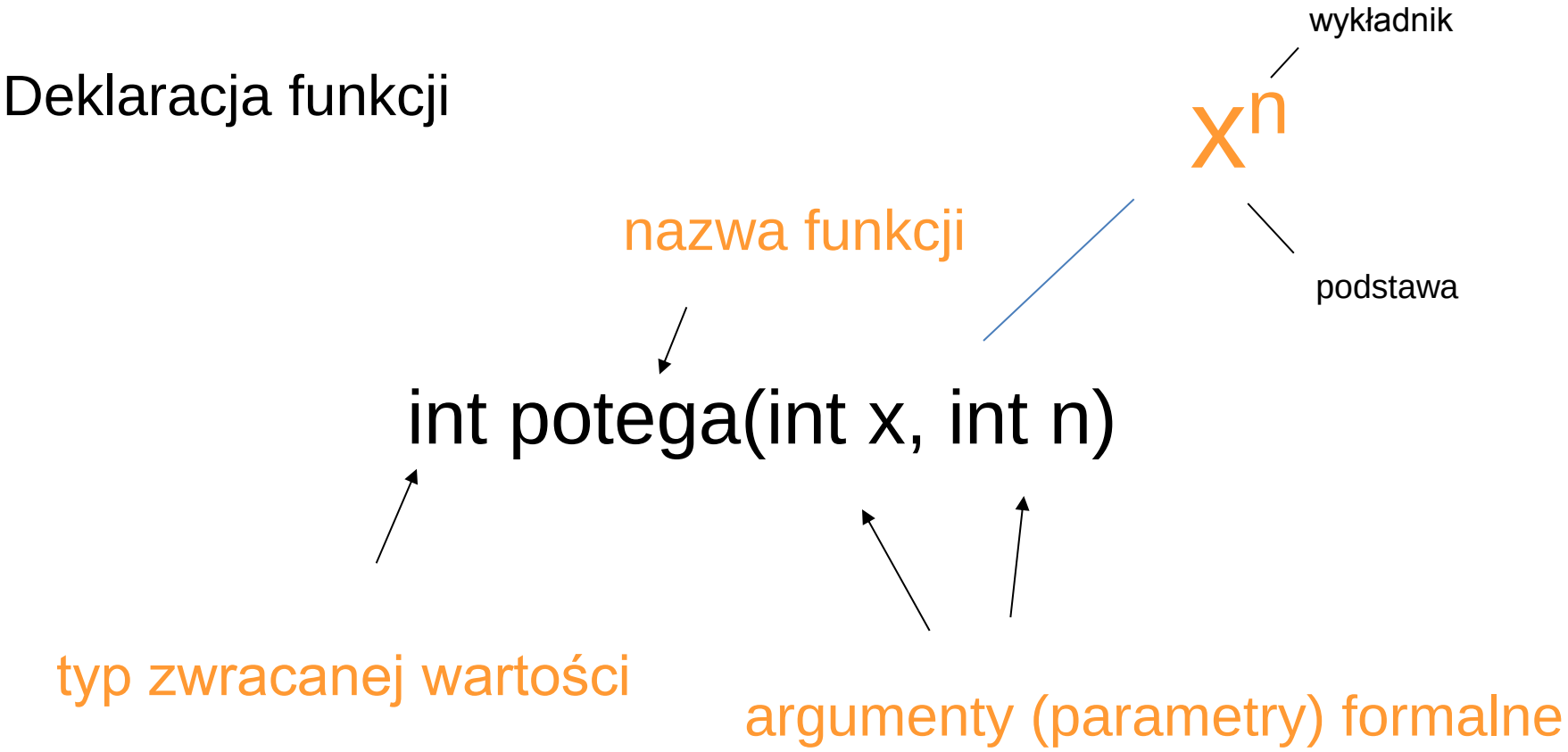



```
void wyswietl()
 {
  cout<<„Uwaga wirus !”;
 }
```

przykład

Funkcje

Deklaracja funkcji



zadanie

Funkcje

Definicja funkcji

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

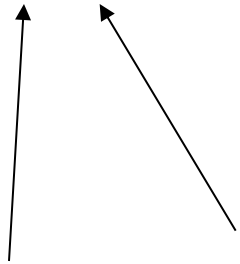
spróbuj napisać samodzielnie

ciało funkcji

Funkcje

Wywołanie funkcji

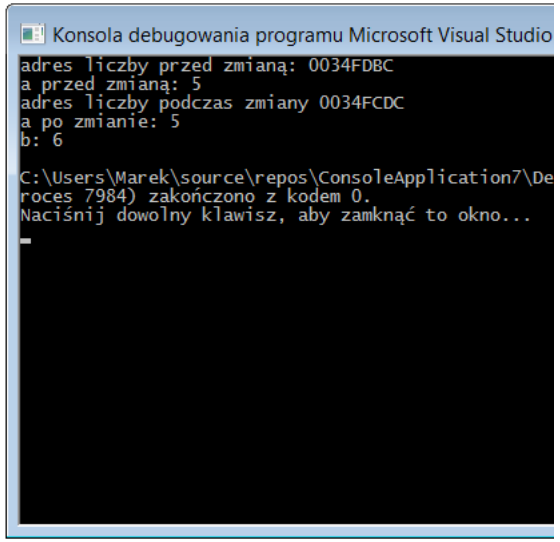
```
int k;
k=potega(2,3);
```



argumenty aktualne

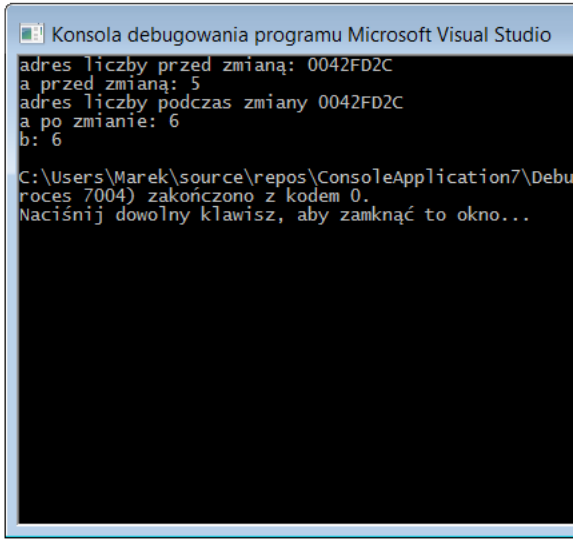
Argumenty funkcji przekazywane przez wartość

```cpp
#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;

}
```



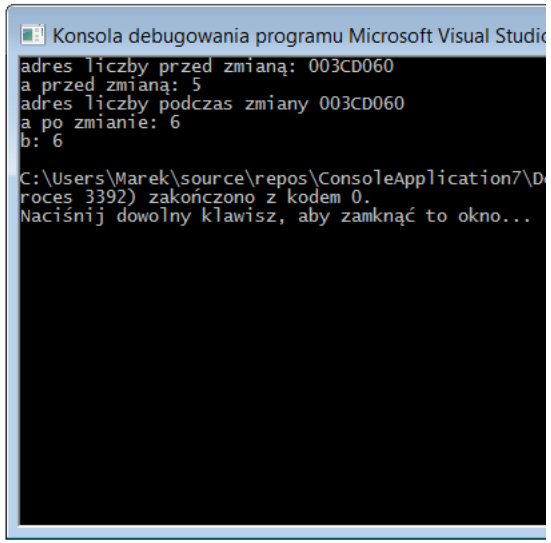
Argumenty funkcji przekazywane przez referencję

```cpp
#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;

}
```

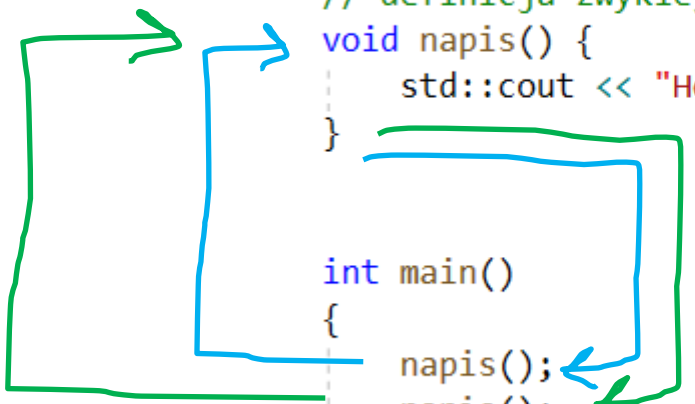


Argumenty funkcji przekazywane przez wskaźnik

```cpp
#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;
}
```



Funkcje - wywołanie zwykłej funkcji



```cpp
#include <iostream>

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


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

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

```cpp
#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.

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

```cpp
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, ze jej ciało powinno mieścić się w jednej lub 2 linijkach.

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

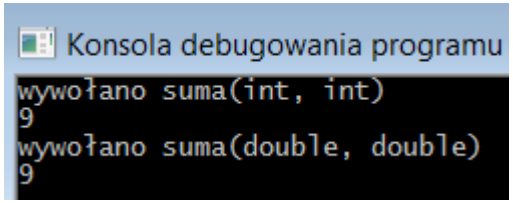
Przeciążenie funkcji

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

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


int main()
{
    suma(5, 4);
    suma(5.0, 4.0);
}

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;
}
```

polimorfizm funkcji – funkcja może przybierać wiele form

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

Którą funkcję wywoła kompilator?



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);
```

int suma(int, int)
+1 przeciążenie

Wyszukaj w trybie online

nie można przeciążyć funkcji różniących się tylko typem zwracanej wartości

Wyszukaj w trybie online

nie można przeciążyć funkcji w ten sposób

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
```

https://edu.pjwstk.edu.pl/wyklady/pro/scb/PRG2CPP_files/node68.html

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

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

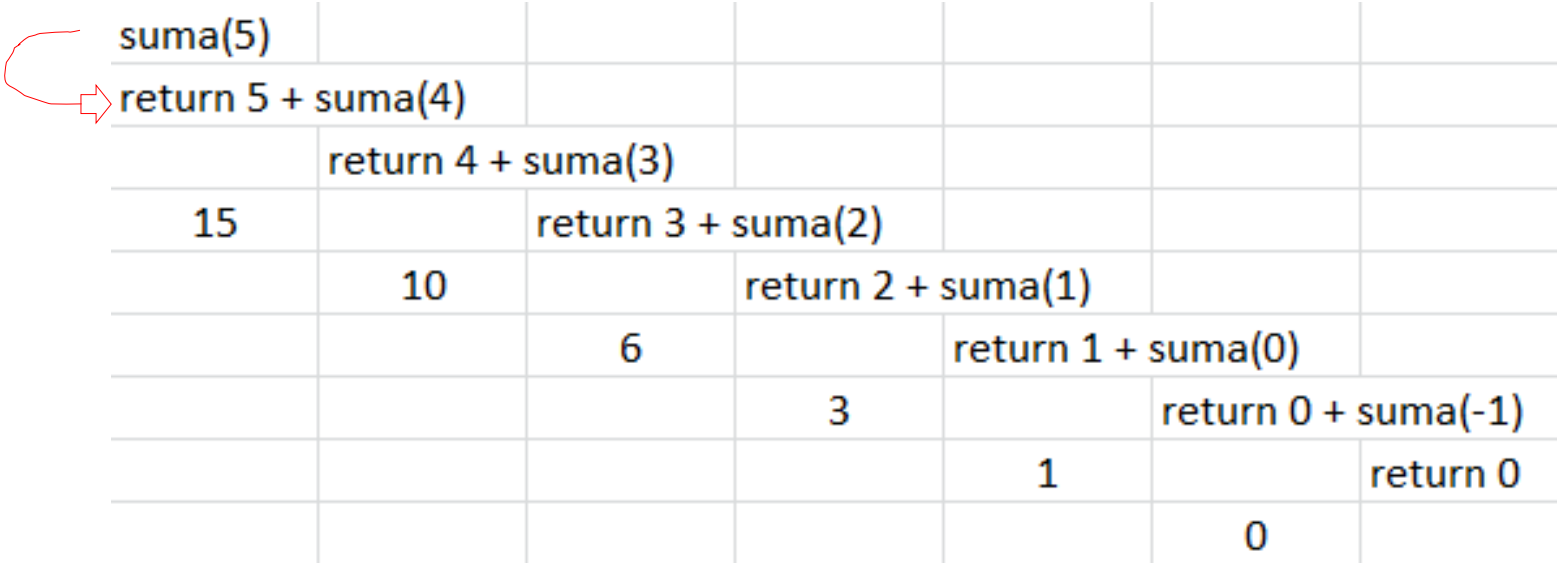
Rekurencja

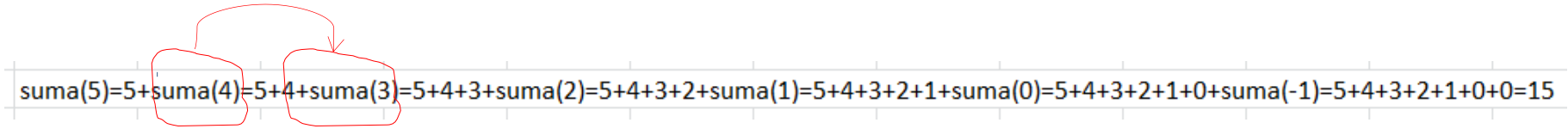
```cpp
#include <iostream>

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

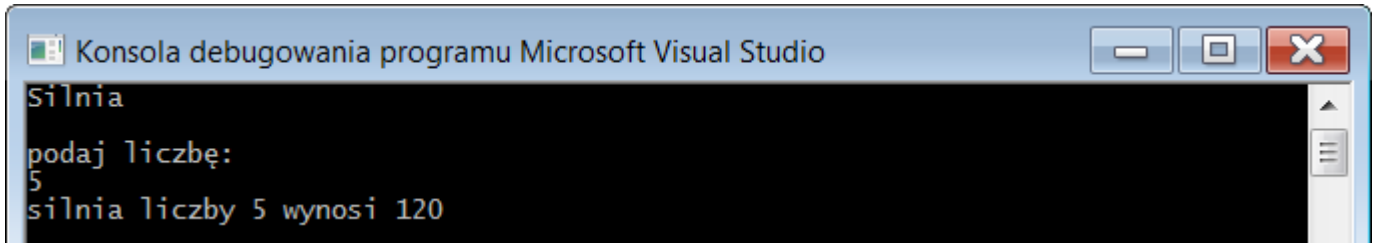
Rekurencja


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


suma(5)=5+suma(4)=5+4+suma(3)=5+4+3+suma(2)=5+4+3+2+suma(1)=5+4+3+2+1+suma(0)=5+4+3+2+1+0+suma(-1)=5+4+3+2+1+0+0=15

zadanie

Rekurencja



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

```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;

}
```

komunikat.h

```cpp
// 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

```cpp
#include<iostream>

//definicja funkcji komunikat
void komunikat() {

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

}
```

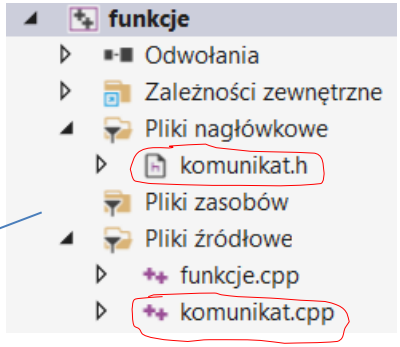
dołączamy plik nagłówkowy komunikat.h z deklaracją funkcji za pomocą dyrektywy preprocesora #include

funkcje.cpp

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

using namespace std;

int main() {

    //wyświetlamy komunikat
    komunikat();

}
```

definicję funkcji możemy przenieść do pliku komunikat.cpp

Eksplorator rozwiązań z nowymi plikami



REFERENCJA

Referencja *Reference*

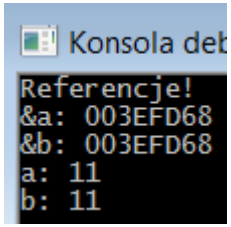
są często używane przy przekazywaniu argumentów do funkcji przez referencję (funkcja nie tworzy wtedy kopii argumentów tylko pracuje na oryginalnych wartościach)

inna nazwa (alias) zmiennej

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

short a = 10;

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

// 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
```

b staje się inną nazwą zmiennej a

adres zmiennej a oraz b jest taki sam w pamięci RAM

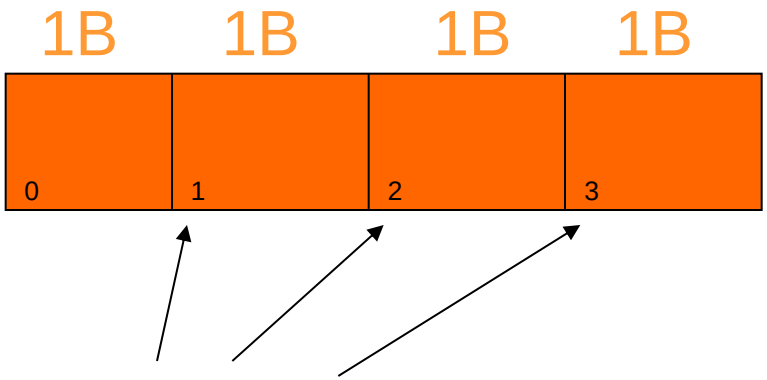


po inkrementacji zmiennej a inkrementowała się również zmienna b

WSKAŹNIKI

Wskaźnik *Pointer*

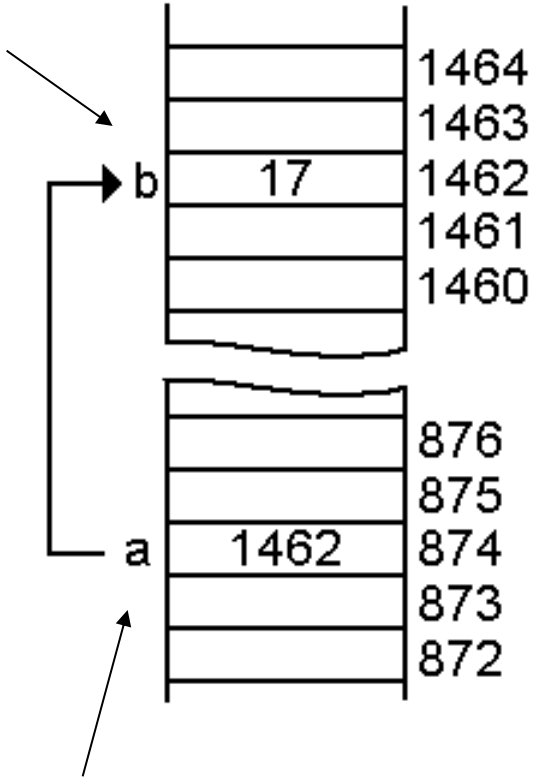
Pamięć w komputerach może być reprezentowana
jako jednowymiarowa tablica bajtów (płaski model pamięci)



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

Wskaźnik *Pointer*

zmienna



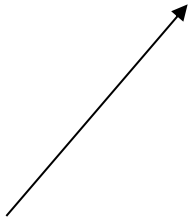
wskaźnik na zmienną

http://en.wikipedia.org/wiki/Pointer_(computing)

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*.

http://pl.wikipedia.org/wiki/Zmienna_wskaźnikowa

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
```

zerowy adres 32 bitowy

#define NULL 0

Jest rozwijane do:0

Wyszukaj w trybie online

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

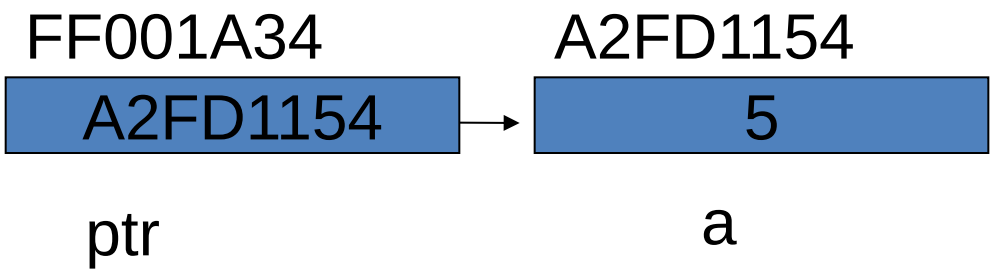
Deklaracja (i definicja) wskaźnika pokazującego na liczbę typu int

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

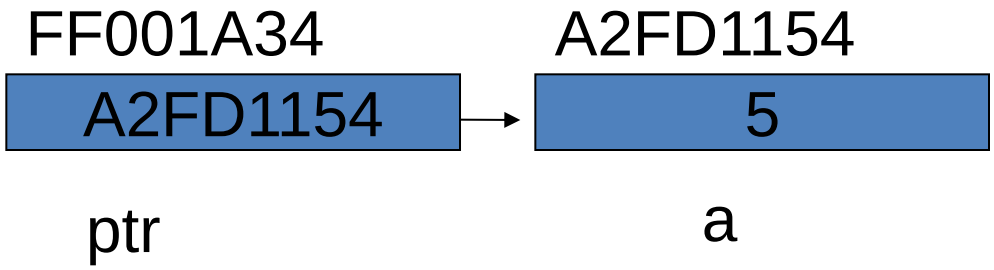
Wskaźniki

```
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;

//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;
```

operator wyłuskania, dereferencji (gwiazdka)

Wskaźniki

wskaźnik na typ int

stworzenie nowego obiektu typu int
adres obiektu przypisujemy do wskaźnika

heap

alokuje pamięć na stercie

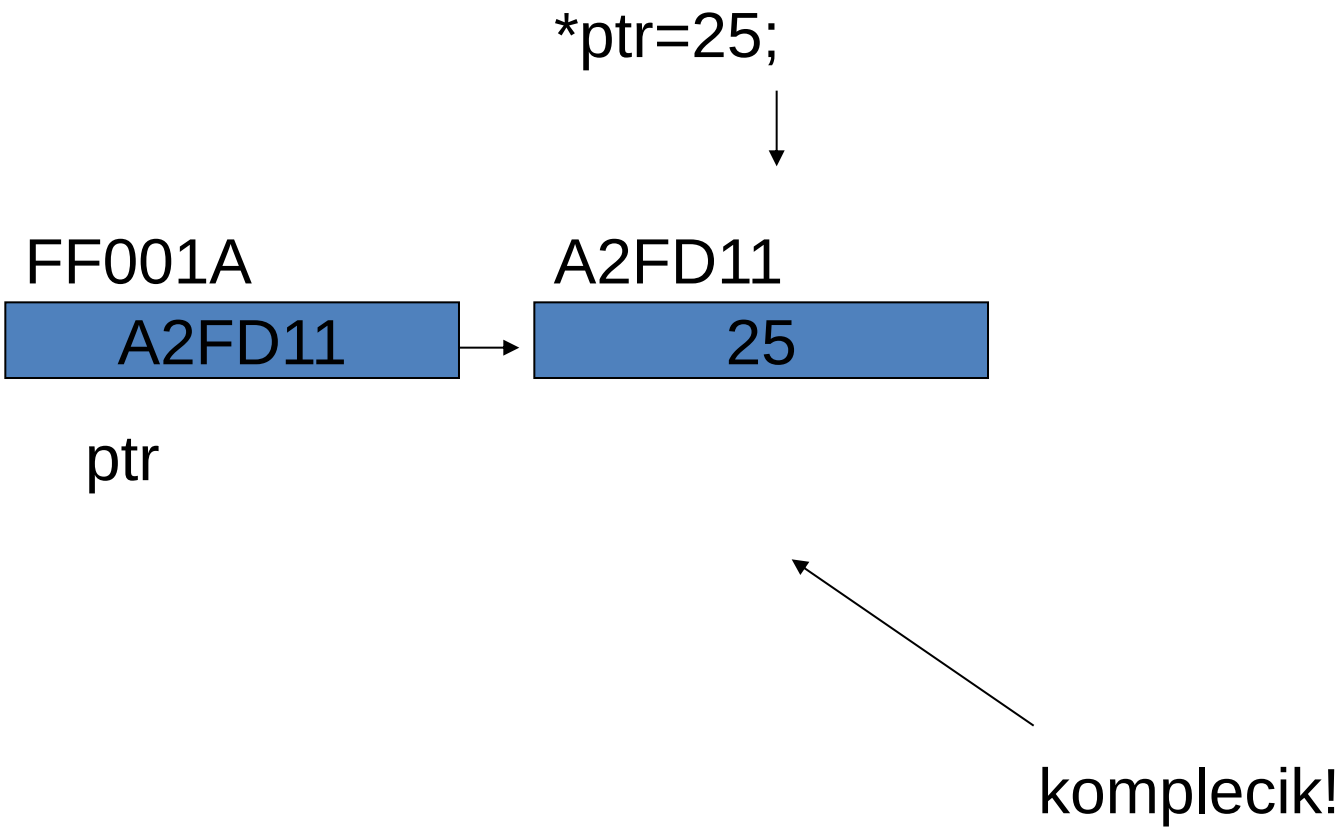
```
int* ptr;
```

```
ptr = new int;
```

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



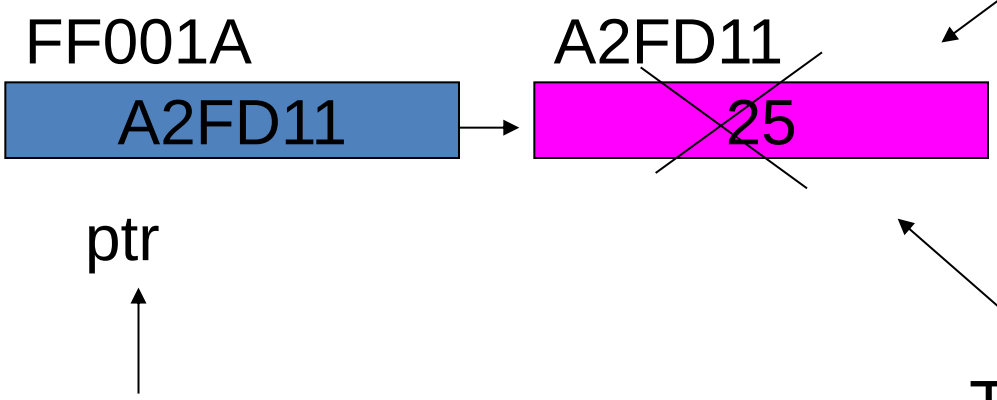
Wskaźniki

bez tego wyciek pamięci w programie

delete ptr;

usuniecie obiektu typu int

Uwaga!
Nie jest to usuniecie wskaźnika!



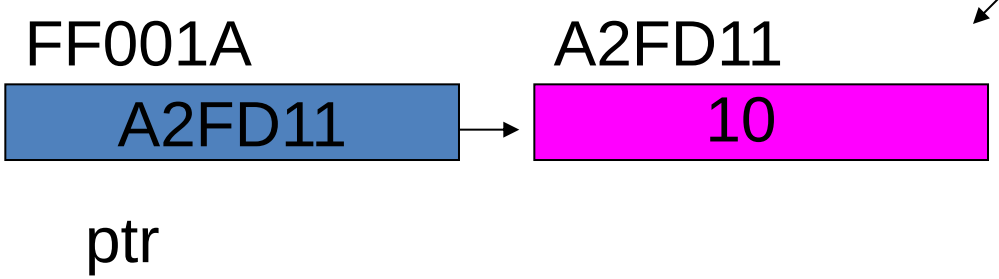
we wskaźniku nic się nie zmienia!

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

Załóżmy, że jakiś program wpisuje pod adresem A2FD11 liczbę 10



Jeśli po raz drugi wykonamy instrukcję:

delete ptr;

to …

https://stackoverflow.com/questions/9169774/what-happens-in-a-double-delete/38201184

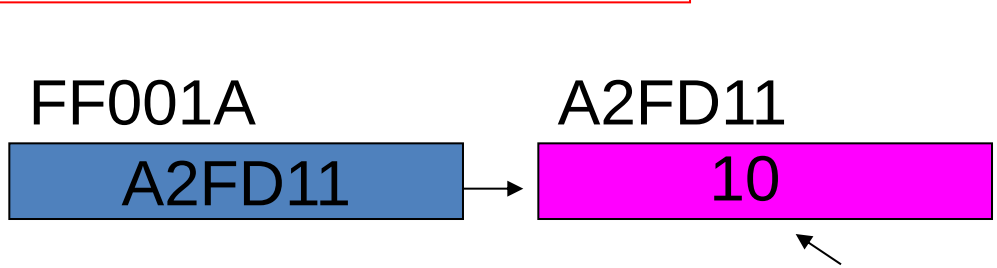
Wskaźniki

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

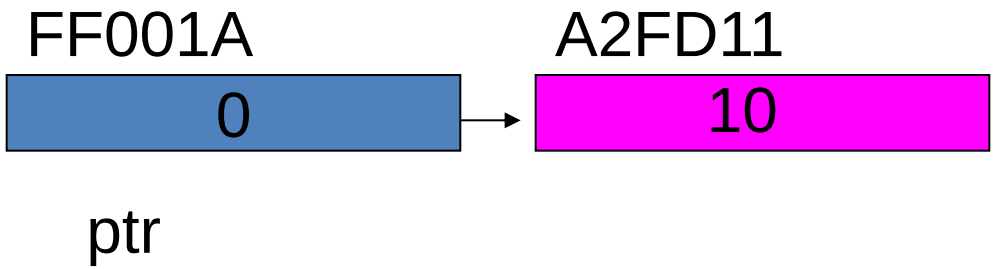


Możemy przypadkowo zniszczyć obiekty nie należące do naszego programu

Dlatego należy wyzerować wskaźnik:

```
ptr = nullptr;
```

Wskaźniki



Jeśli teraz wywołamy ponownie instrukcję:

delete ptr;

Ten wiersz zostanie zignorowany.

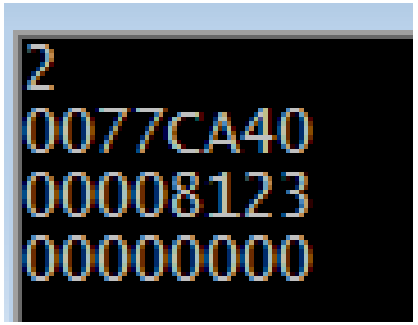
Wskaźnik wskazuje na adres zerowy, którego nie ma.
Wskaźnik nie wskazuje na żadną konkretną komórkę pamięci.
Nie ma czego usuwać!

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;
```

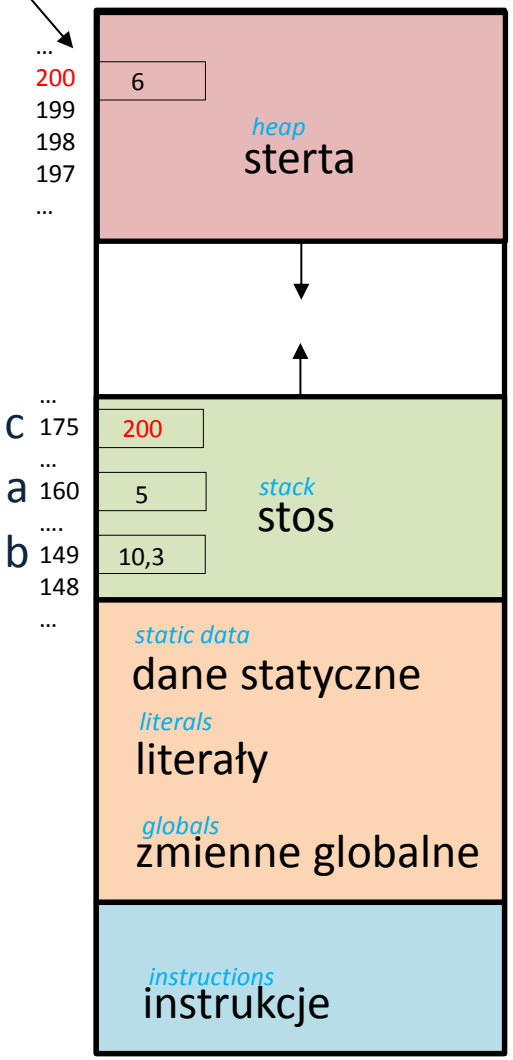
OK

don’t do it !

Pamięć programu

zmienne na stercie 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 dynamiczne - 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

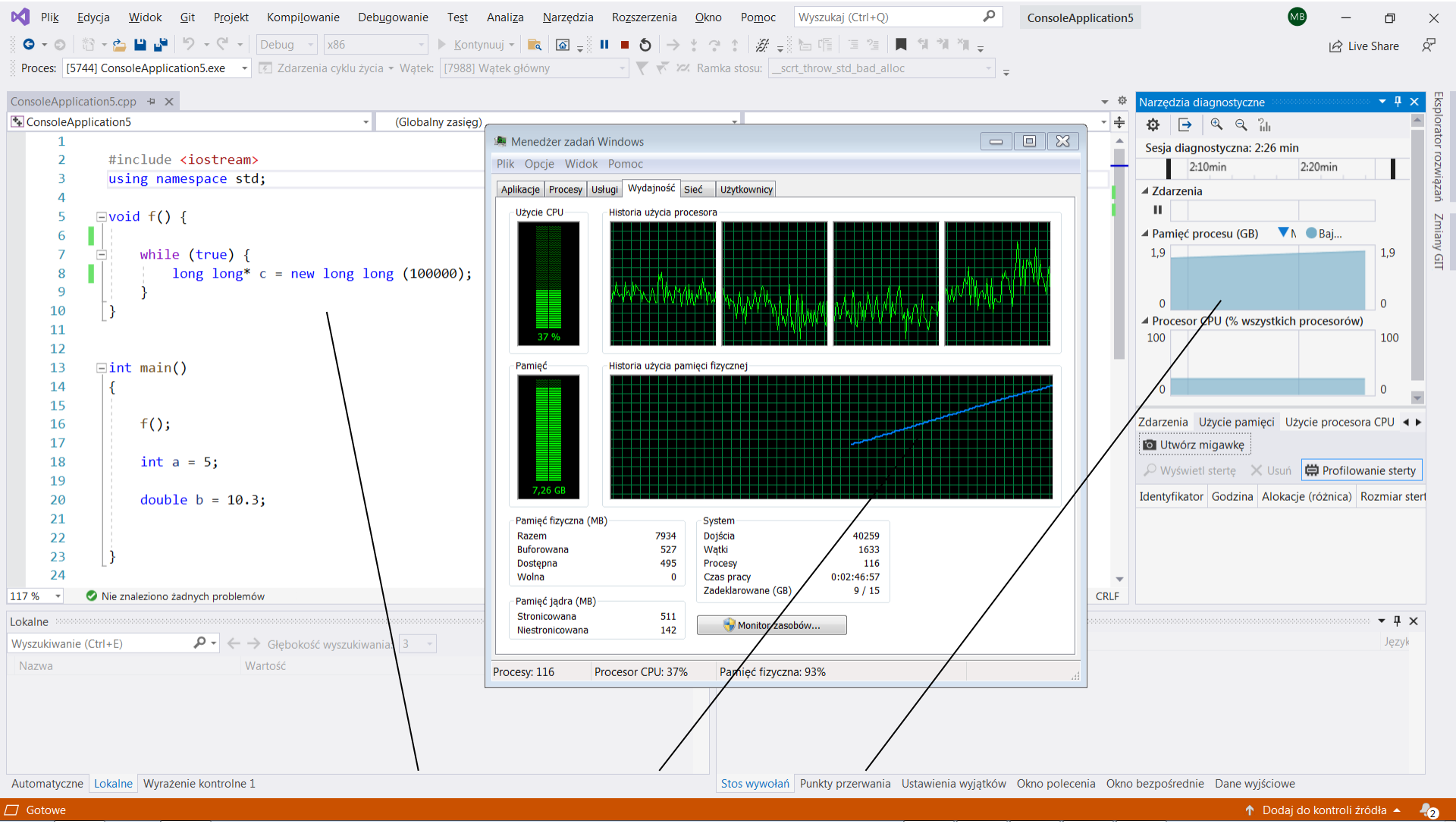
Dynamiczna alokacja pamięci na stercie – wyciek pamięci

Przykład 1

```
void f() {

    // alokujemy pamięć na stercie w pętli nieskończonej
    while(true)
    {
        long long* c = new long long(100000);
    }
}

int main()
{
    // funkcja alokująca pamięć na stercie
    f();

    int d = 5;

    double b = 10.3;

}
```

Dynamiczna alokacja pamięci na stercie – wyciek pamięci

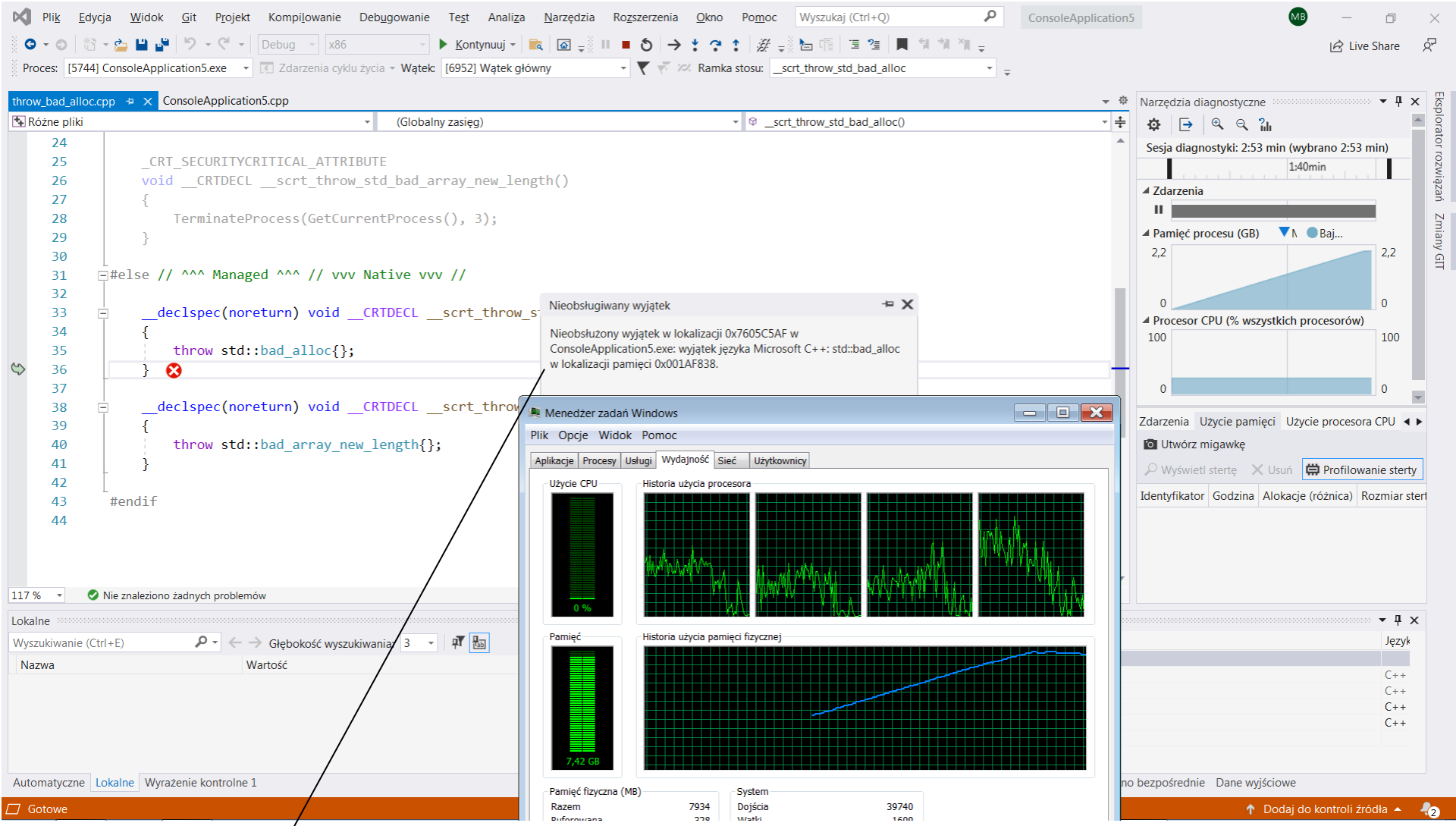
Przykład 1



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



Po wyczerpaniu dostępnej pamięci - wyjątek

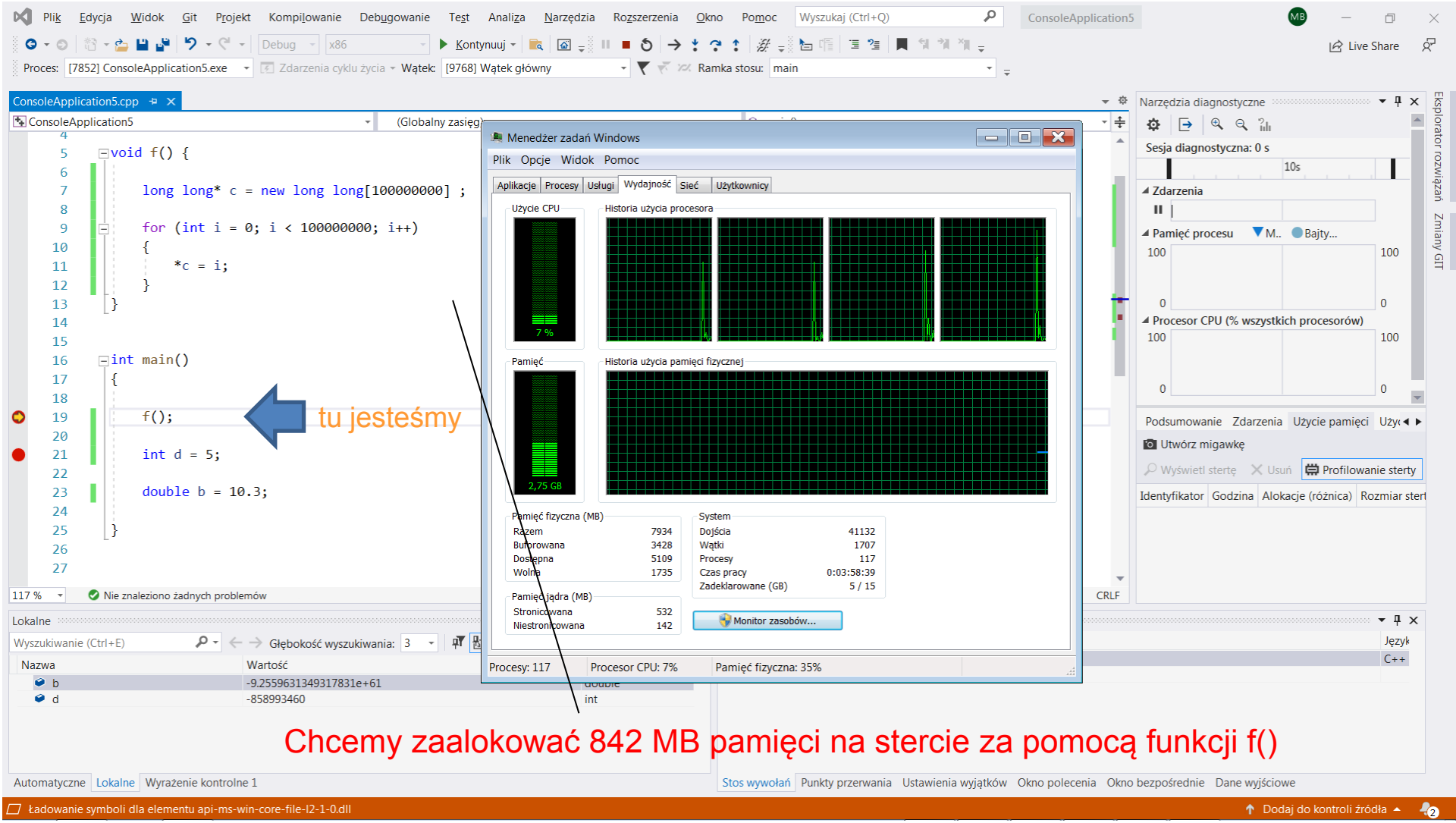
Dynamiczna alokacja pamięci na stercie – wyciek pamięci

Przykład 2

```
void f() {

    // deklarujemy tablicę typu long long na stercie
    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 stercie
    f();

    int d = 5;

    double b = 10.3;

}
```

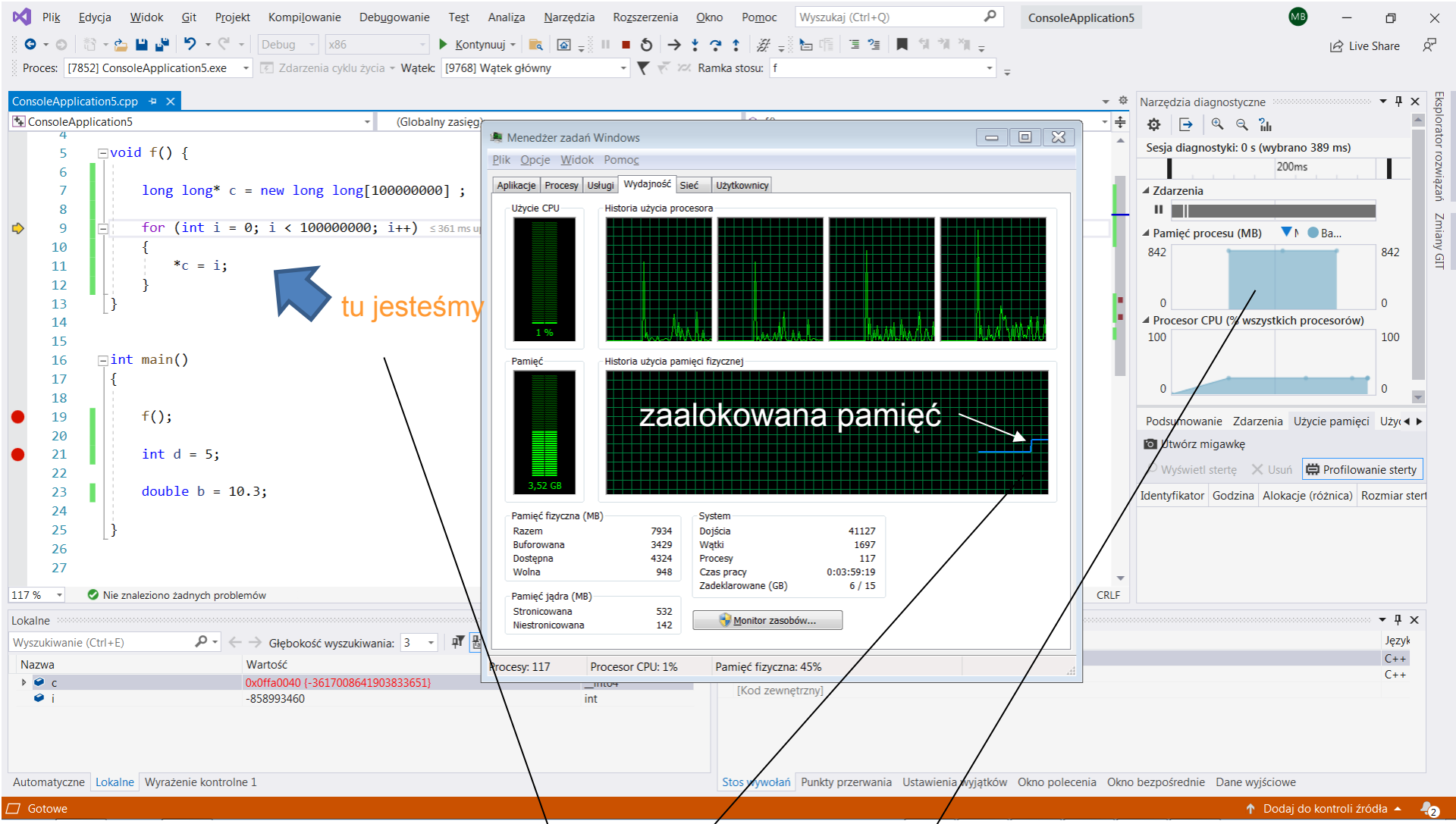
Dynamiczna alokacja pamięci na stercie – wyciek pamięci

Przykład 2



Dynamiczna alokacja pamięci na stercie – wyciek pamięci

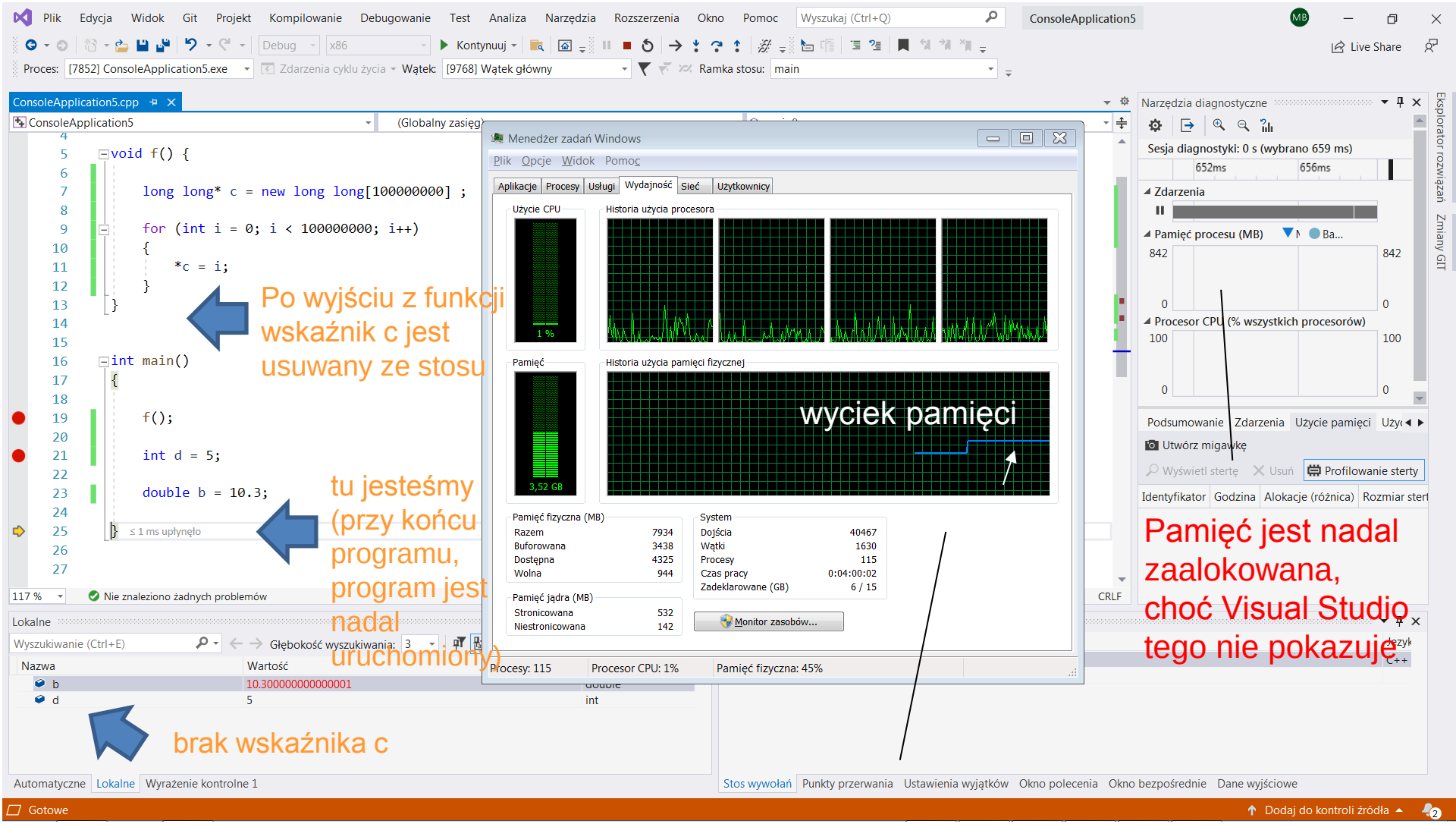
Przykład 2



Zaalokowaliśmy 842 MB pamięci na stercie

Dynamiczna alokacja pamięci na stercie – wyciek pamięci

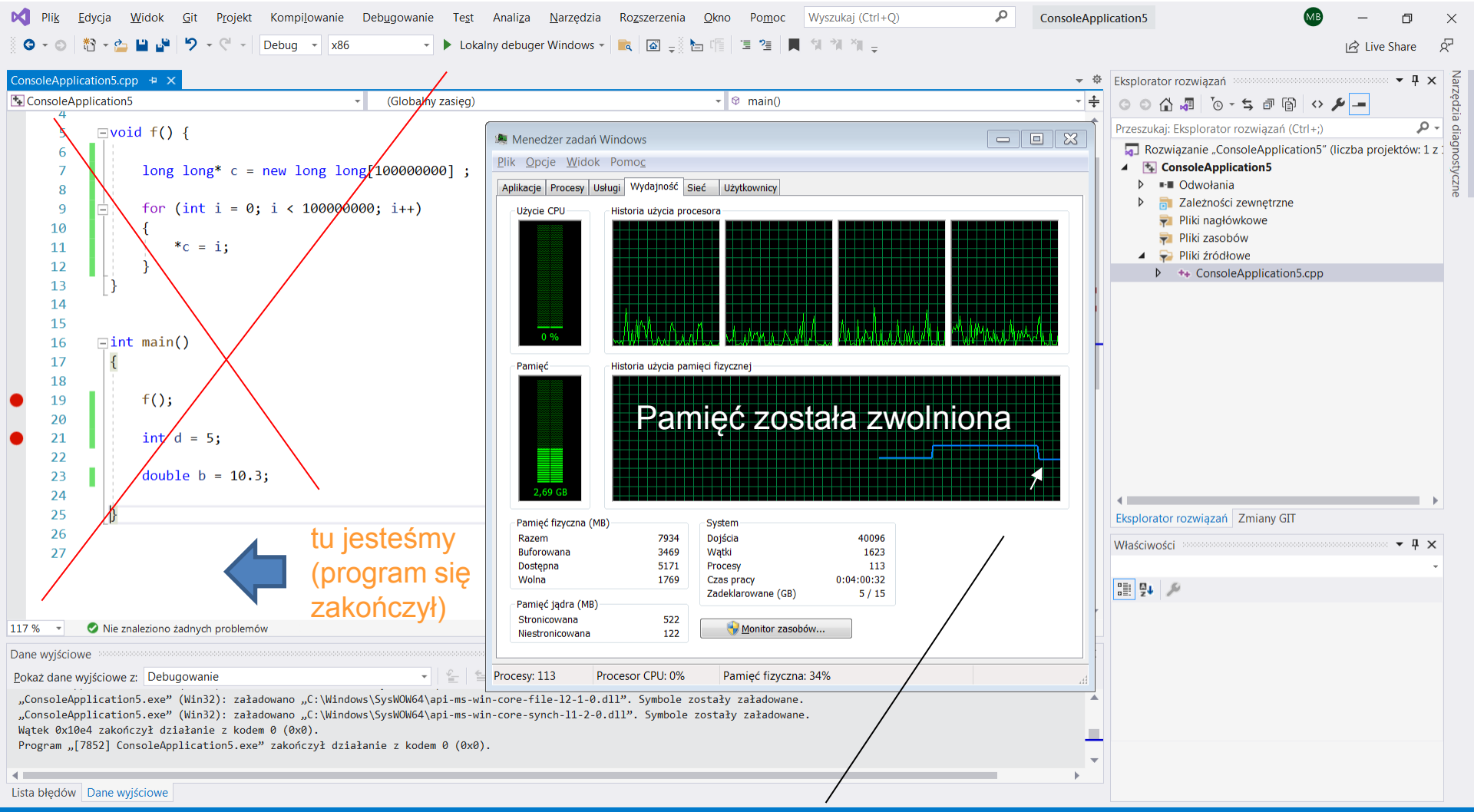
Przykład 2



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 stercie – wyciek pamięci

Przykład 2



Zaalokowana pamięć na stercie jest zwalniana do puli dostępnej pamięci dopiero po zakończeniu programu

Dynamiczna alokacja pamięci na stercie – wyciek pamięci

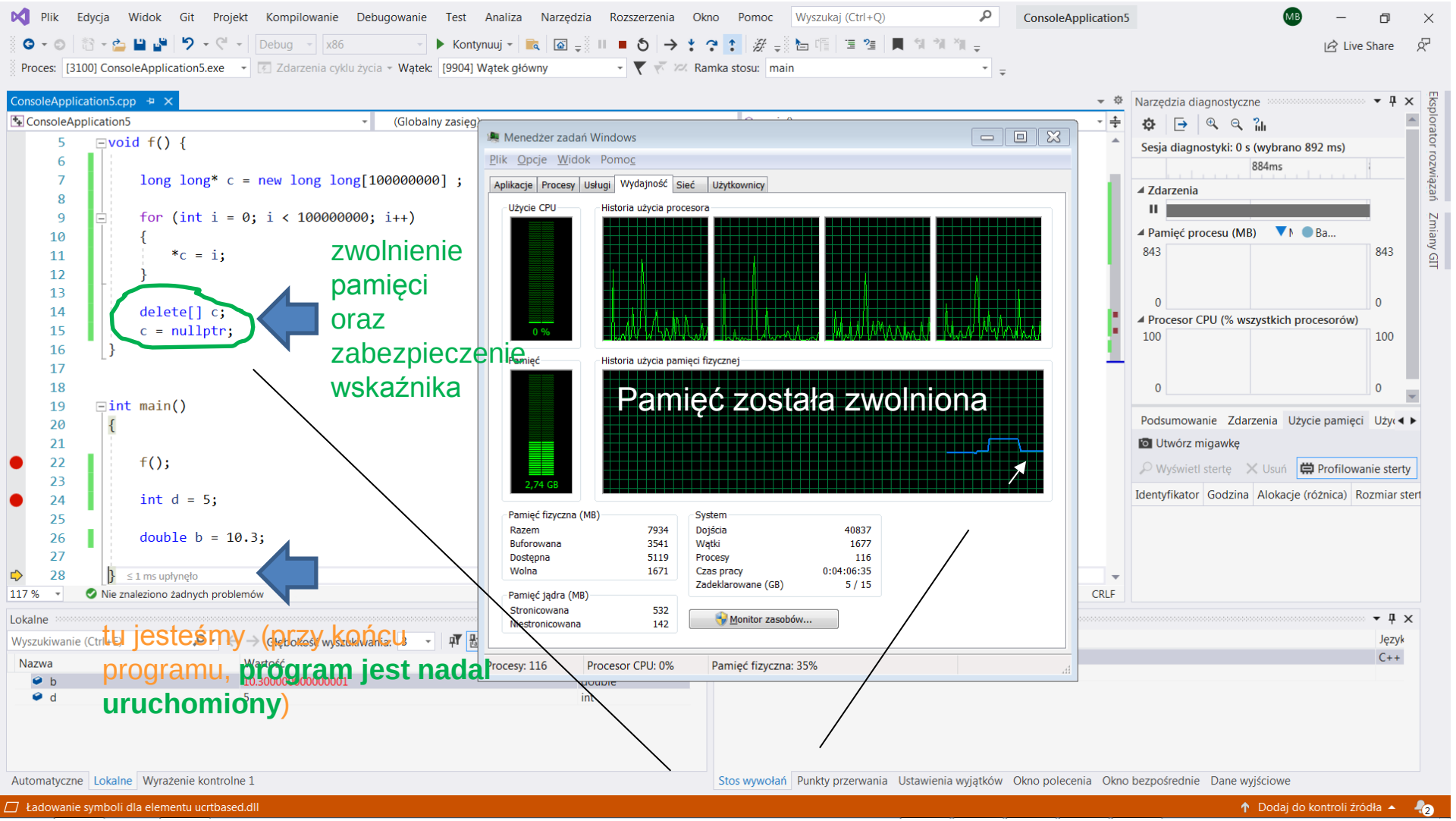
Przykład 2

```
void f() {

    // deklarujemy tablicę typu long long na stercie
    long long* c = new long long[100000000] ;

    // wypełniamy tablicę liczbami
    for (int i = 0; i < 100000000; i++)
    {
        *c = i;
    }

    // usówamy wskaźnik na tablicę
    delete[] c;

    // przypisujemy do wskaźnika nullptr
    c = nullptr;
}


int main()
{
    // funkcja alokująca pamięć na stercie
    f();

    int d = 5;

    double b = 10.3;

}
```

zwolnienie Pamięci oraz zabezpieczenie wskaźnika

Dynamiczna alokacja pamięci na stercie – wyciek pamięci

Przykład 2



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

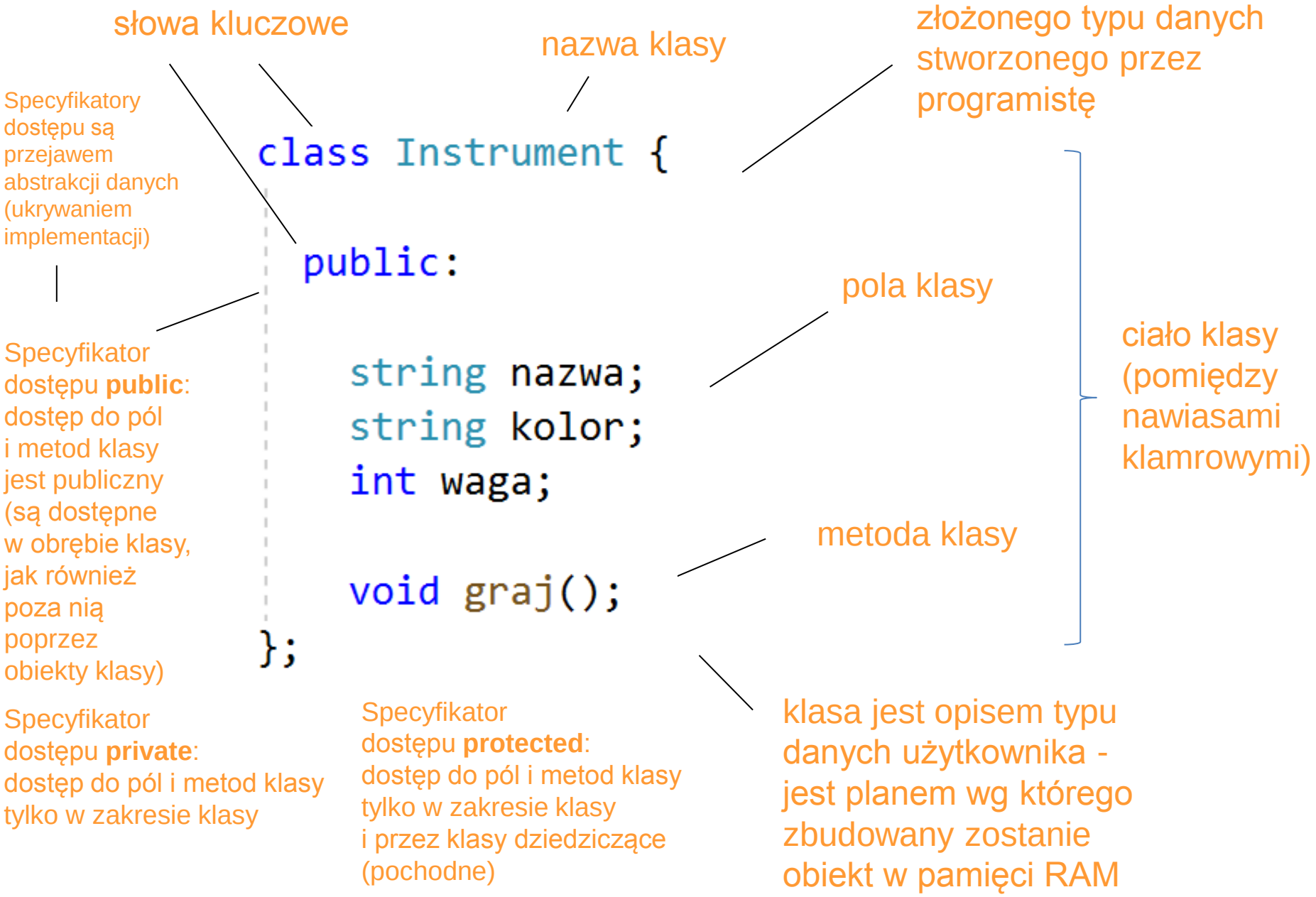
definiowanie klas dziedziczących (pochodnych) na podstawie klasy bazowej (klasy matki):
klasy pochodne dziedziczą po klasie bazowej pola i metody

polimorfizm

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

Klasy

Klasa


słowa kluczowe
nazwa klasy
definicja klasy – złożonego typu danych stworzonego przez programistę
Specyfikatory dostępu są przejawem abstrakcji danych (ukrywaniem implementacji)
class Instrument {
public:
string nazwa;
string kolor;
int waga;
void graj();
};
pola klasy
ciało klasy (pomiędzy nawiasami klamrowymi)
metoda klasy
Specyfikator dostępu public: dostęp do pól i metod klasy jest publiczny (są dostępne w obrębie klasy, jak również poza nią poprzez obiekty klasy)
Specyfikator dostępu private: dostęp do pól i metod klasy tylko w zakresie klasy
Specyfikator dostępu protected: dostęp do pól i metod klasy tylko w zakresie klasy i przez klasy dziedziczące (pochodne)
klasa jest opisem typu danych użytkownika - jest planem wg którego zbudowany zostanie obiekt w pamięci RAM

Klasa – metoda inline

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

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

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

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

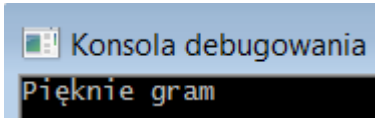
Metoda jest zdefiniowana wewnątrz klasy, dlatego domyślnie przyjmuje się dla niej modyfikator **inline** (to samo tyczy się metod statycznych, konstruktorów i destruktorów). Metody takie powinny być krótkie (jedna linijka kodu ciała metody)

Tworzymy obiekt gitara klasy Instrument

Wywołanie metody graj() na rzecz obiektu gitara

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

Kompilator rozwija metodę graj() w miejscu wywołania



Klasa

```cpp
#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; };
};

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

dostęp do prywatnego pola x wewnątrz klasy A ☺ OK

dostęp do chronionego pola y wewnątrz klasy A ☺ OK

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 destruktorem

```cpp
#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
};

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

// 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;
}

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

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

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

nazwa kwalifikowa

Klasa Instrument z konstruktorem domyślnym i destruktorem

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

obiekt gitara jest budowany w pamięci przez konstruktora domyślnego

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

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

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

Konsola debugowania programu Microsoft Visual Studio

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

nadajemy nazwę instrumentowi: gitara
nazwa instrumentu: gitara

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

Klasa Instrument z konstruktorem parametrowym

konstruktor wieloparametrowy
z argumentami domyślnymi oraz
listą inicjalizacyjną

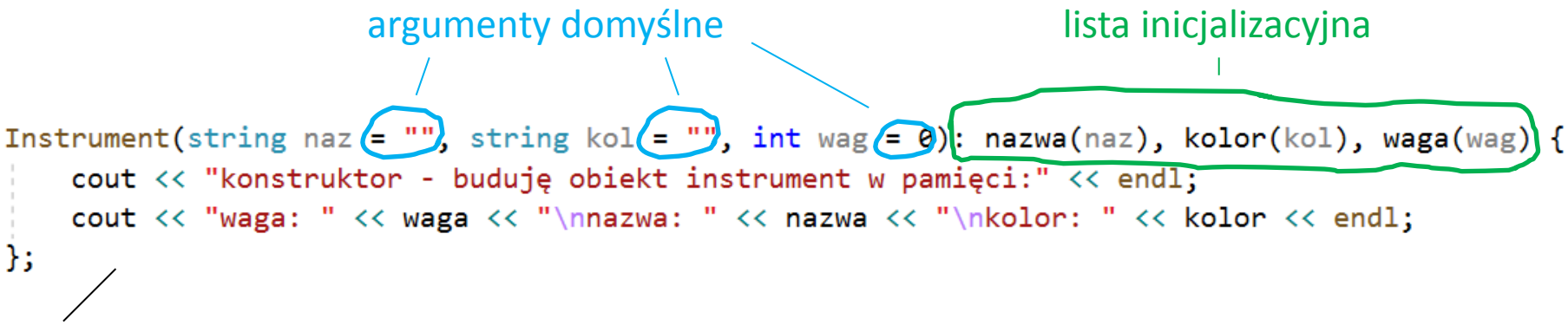
```cpp
#include <iostream>
using namespace std;

// 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();
}
```

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



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;

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);
};
```

ukrywanie danych

prywatne pola klasy dostępne w zasięgu klasy

interfejs publiczny klasy
dostępny w klasie i poza nią

gettery
oraz settery
są przejawem
abstrakcji
danych
(ukrywaniem
implementacji)

telewizor ma części
ukryte przed użytkownikiem
(private)

interfejs telewizora
(public)

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

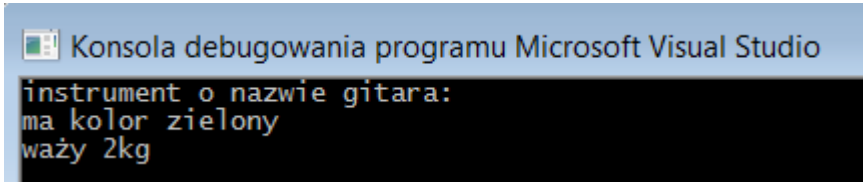
do prywatnego pola klasy mamy dostęp za pomocą specjalnych metod dostępowych : gettera i settera

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

}
```

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



napisz program korzystając z enkapsulacji

Klasa Komputer

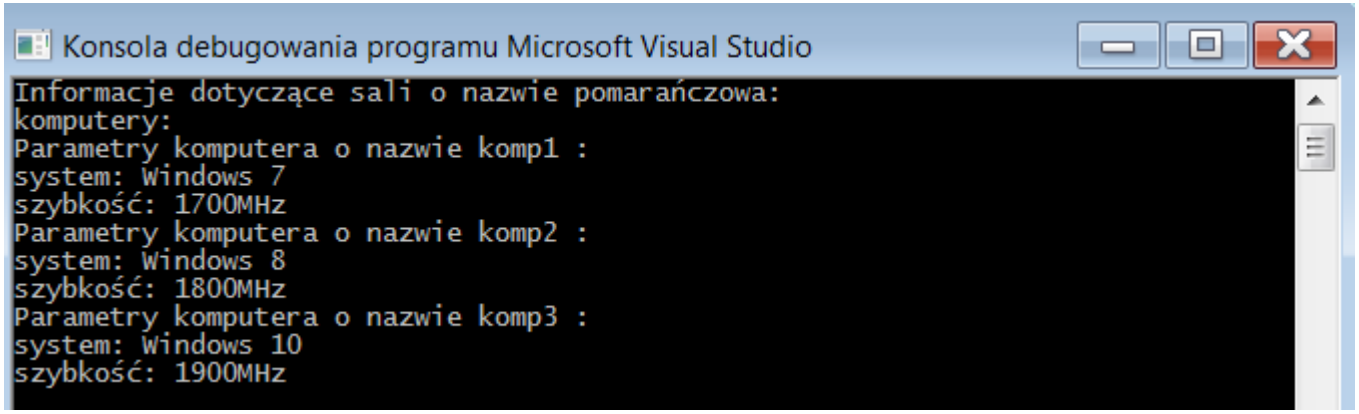
```cpp
#include <iostream>
#include <vector>
using namespace std;

// 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();  // wypisuuje 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 getParameryKompoterow();     // wypisuje na konsoli parametry komputerów
    void addKomputerDoListy(Komputer komp);    // dodaje komputer do listy komputerów
    void podajInformacje();           // wypisauje informacje o sali na kosoli
};

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

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

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

Obiekt pomaranczowa klasy Sala

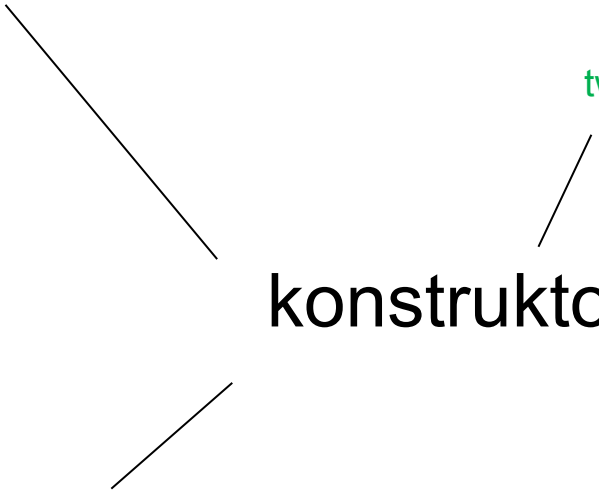
```
int main()
{
    // tworzymy obiekty klasy komputer
    Komputer komp1("komp1", "Windows 7", 1700);
    Komputer komp2("komp2", "Windows 8", 1800);
    Komputer komp3("komp3", "Windows 10", 1900);

    // 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();
}
```



Copy constructor

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 automatyczne

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

int main() {

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

    // tworzymy nowego ucznia kopiując istniejącego ucznia
    // działa konstruktor kopiujący wygenerowany automatycznie
    Uczen 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;
}
```

typ wbudowany string

```
Marek
Marek
Jarek
Marek
```

Shallow Copy

uczen1	uczen2
imie: Marek	imie: Marek
uczen1	uczen2
imie: Jarek	imie: Marek

Konstruktor kopiujący wygenerowany automatycznie wystarcza

Konstruktor kopiujący wygenerowany automatyczne

```
class Uczen {
private:
    string* imie;
public:
    // konstruktor
    Uczen(string nazwa) {
        imie = new string(nazwa);
    };
    // destruktor
    ~Uczen() {
        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

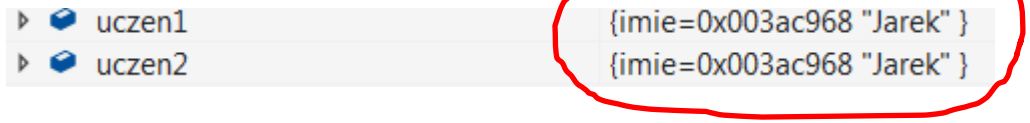
uczen1 uczen2

imie -> Jarek <- imie

```
int main() {

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

    // działa konstruktor kopiujący wygenerowany automatycznie
    Uczen 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;
}
```

```
Marek
Marek
Jarek
Jarek
```

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



Konstruktor kopiujący wygenerowany automatyczne

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

wskaźnik typu string

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

Zgłoszony wyjątek

Zgłoszono wyjątek: naruszenie dostępu do odczytu.
_Pnext było 0xDDDDDDE1.

Destruktor usuwa 2x ten sam wskaźnik

```
int main() {

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

    // działa konstruktor kopiujący wygenerowany automatycznie
    Uczen 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

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

    string getImie() {
        return *imie;
    }
    void setImie(string nazwa) {
        *imie = nazwa;
    }
};

int main() {

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

    // działa konstruktor kopiujący napisany przez programistę (Deep Copy)
    Uczen 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;
}
```

wskaźnik typu string

napisany konstruktor kopiujący

nie ma problemu z destruktorem

Deep Copy

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

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

```
Marek
Marek
Jarek
Marek
```

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 zainicjalozowac 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 moze być zainicjowana wewnatrz 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 powstajace obiekty klasy Cat
    Cat(string i = "") :name(i) {
        counter++;
    };
};

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

stałe pole statyczne max

pole statyczne counter:

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

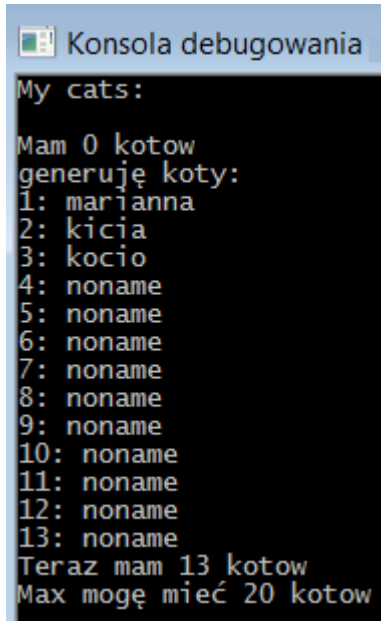
Jeśli metoda byłaby niestatyczna, nie mogłaby korzystać ze zmiennych statycznych

inicjalizacja pola statycznego klasy Cat

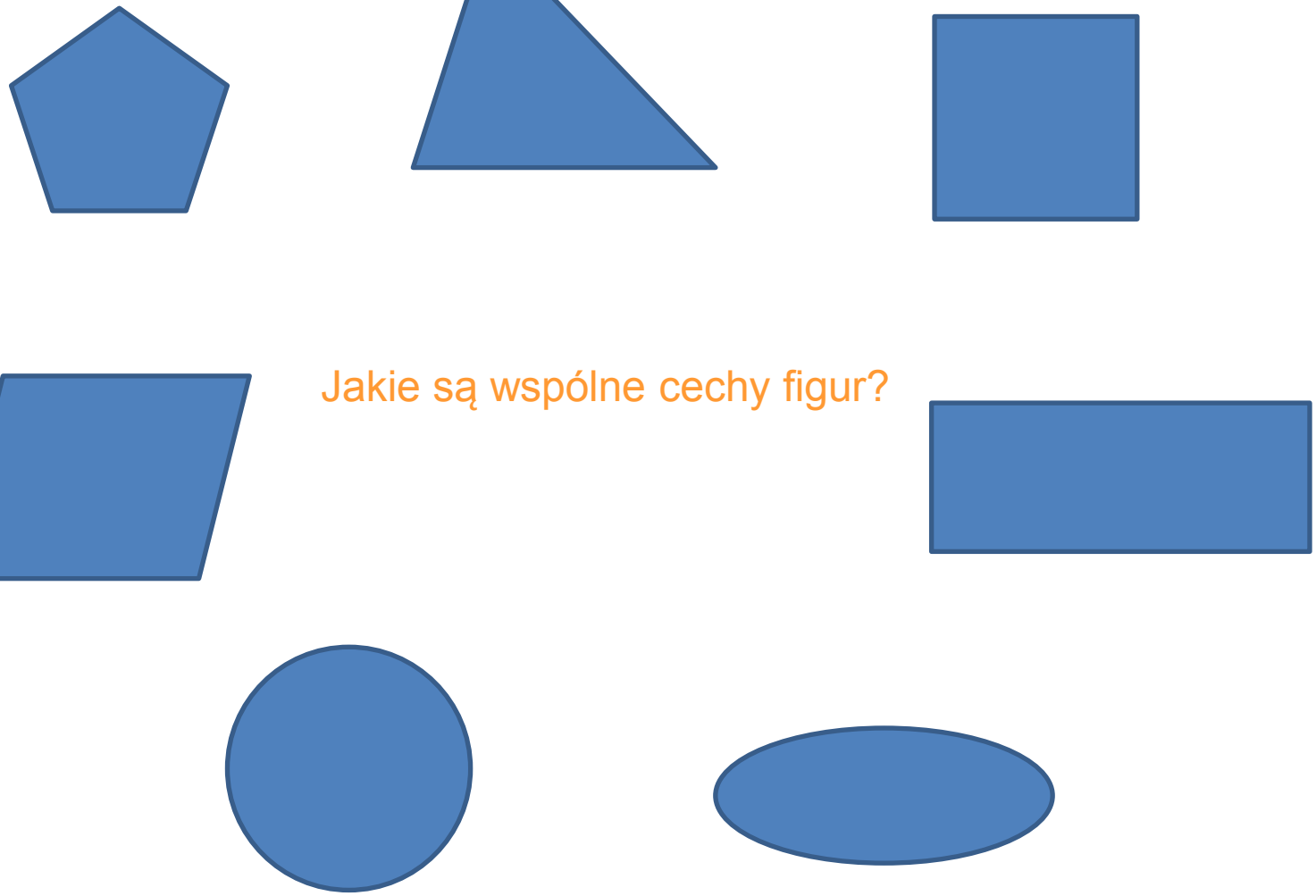
Pola i metody statyczne klasy

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

    // do pola statycznego możemy sie 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



Dziedziczenie



Jakie są wspólne cechy figur?

Dziedziczenie

Klasa bazowa Figura

wspólne cechy figur

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

class Figura {
public:
    string nazwa;
    double pole;
    double obwod;
    // konstruktor z wartościami domyslnymi 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

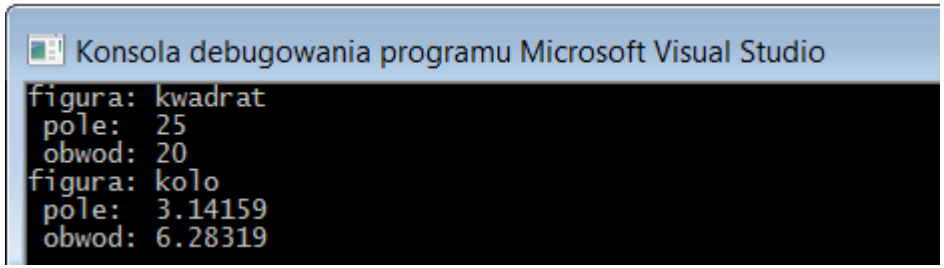
```
class Kwadrat : public Figura {
public:
    double bok;
    // konstruktor z wartościami domyslnymi oraz listą inicjalizacyjną,
    // tworzymy tu również obiekt klasy Figura (wywołujemy konstruktor klasy Figura)
    Kwadrat(string n, double b, double p = 0.0, double o = 0.0) : Figura(n, p, o), bok(b)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

Klasa pochodna Kolo

```
class Kolo : public Figura {
public:
    double promien;
    // konstruktor z wartościami domyslnymi oraz listąinicjalizacyjną,
    // 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();
}
```

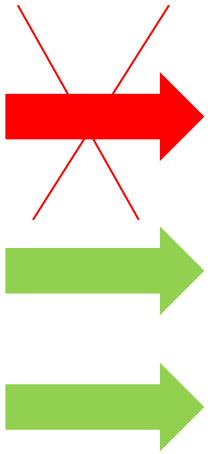


Dziedziczenie public

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 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;
}
```

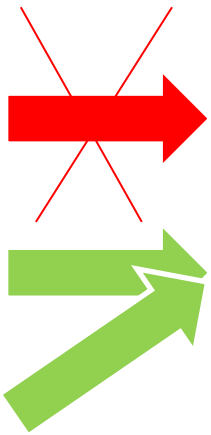
☺ OK

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 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

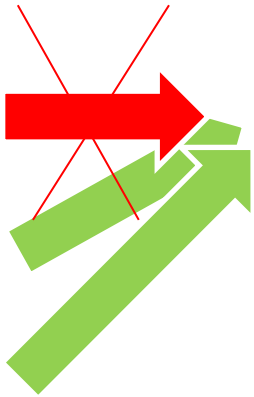
Wyszukaj w trybie online

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

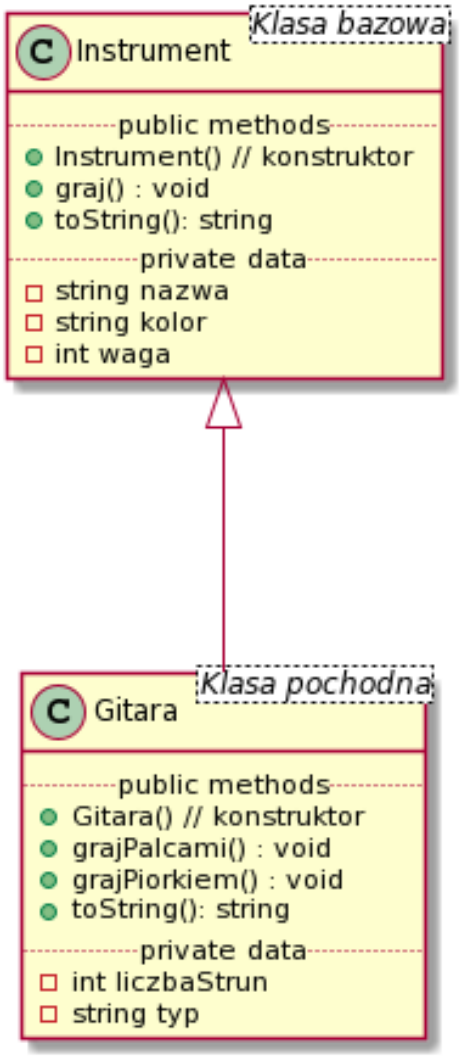
Dziedziczenie public

```
#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:
    // dostep 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:
    // dostep 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 zewnatrz 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

https://www.planttext.com/

zadanie

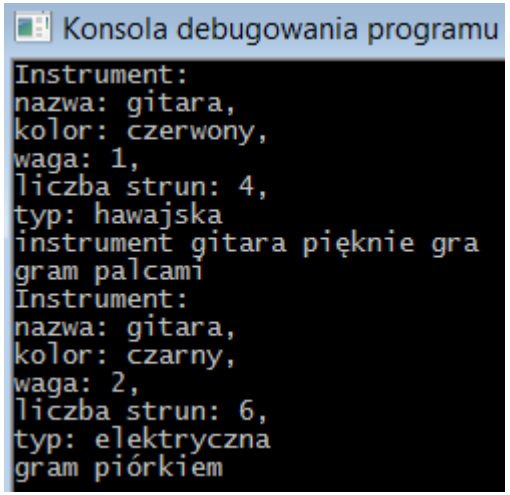
Dziedziczenie

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 opisujaca prywatne pola instrumentu
    string toString();
    void graj();

};
```

napisz program

```
// 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();
};
```



Polimorfizm *Polymorphism*

wielopostaciowość metod klasy, które mogą przybierać wiele form

methods overloading

Polimorfizm statyczny – przeciążanie metod

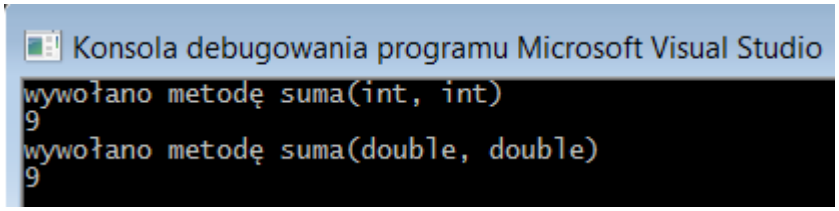
```
#include <iostream>
using namespace std;

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)

Którą metodę wywoła kompilator?

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

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;
}
```



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).

methods overloading

Polimorfizm statyczny – przeciążanie metod

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

methods overriding

Polimorfizm statyczny – przesłanianie metod

```
#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()
{
    Tata tata;
    tata.przedstawSie();

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

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

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

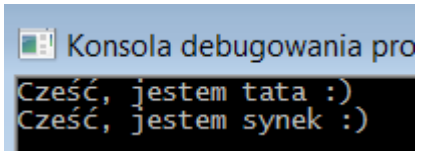
klasa Syn dziedziczy publicznie z klasy Tata

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

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

kompilator wybiera metodę z klasy bazowej

kompilator wybiera metodę z klasy pochodnej



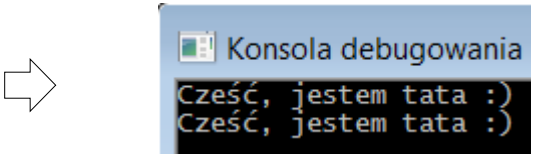
Polimorfizm statyczny – przesłanianie metod

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

```cpp
#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

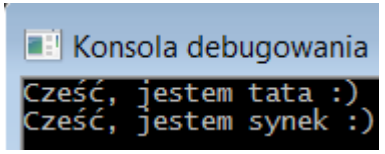


Polimorfizm dynamiczny

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



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

ukrywanie przed użytkownikiem szczegółów implementacji klas, udostępnienie użytkownikom funkcjonalności

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”

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

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

abstract method
- 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”)

Klasa abstrakcyjna

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)

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

klasy pochodne

```cpp
#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; };
};
```

nie można tworzyć obiektu klasy abstrakcyjnej

```cpp
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

```cpp
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

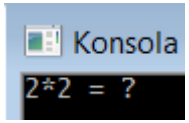
Klasa abstrakcyjna

```cpp
#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; };
};
```

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

wywołanie metody zgadnij() na rzecz obiektu zm klasy ZagadkaDlaDzieciMatematyczna

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



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

Klasa abstrakcyjna - interfejs

implementujemy interfejs

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

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

using namespace std;

// klasa bazowa z metodą czysto wirtualną
// (interfejs - są tylko metody czysto wirtualne, brak pól,
// mogą wystepować 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 wirualnej
    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; };
};
```

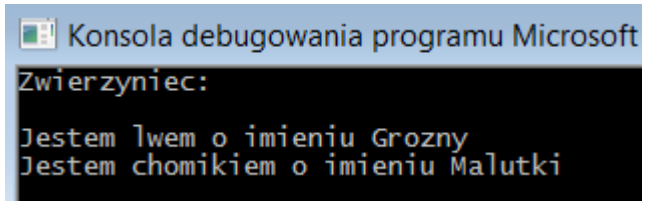
wielodziedziczenie – klasa Lew
dziedziczy z interfejsu metodę
przedstawSie(), a z klasy Nazwa pole
imie.

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 dzidziczenia
oraz słowu kluczowemu virtual
uzyskujemy właściwość polimorfizmu
dynamicznego (późne wiązanie)



Funkcja zaprzyjaźniona z klasą

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

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


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 << "dzieki!" << endl;
}
```

słowo kluczowe friend

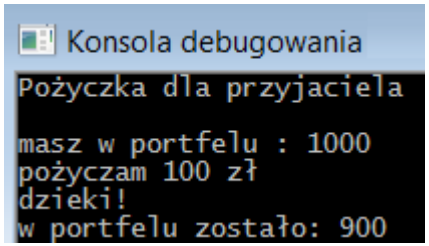
funkcja zaprzyjaźniona jest zwykłą funkcją

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



funkcja zaprzyjaźniona operuje na prywatnym polu klasy Portfel

Funkcja (metoda) zaprzyjaźniona z klasą

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

// deklaracja klasy Portfel
class Portfel;

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


class Portfel {
private:
    int kasa;
public:
    // konstrultor 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; };
};
```

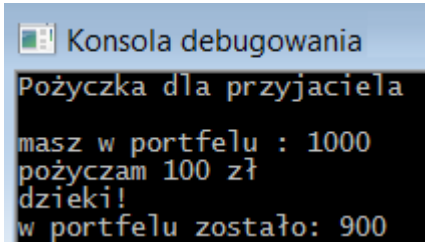
ważna jest kolejność klas!

```cpp
int main()
{
    cout << "Pożyczka dla przyjaciela\n\n";
    Portfel portfel;
    Przyjaciel przyjaciel;
    przyjaciel.pozyczKase(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 << "dzieki!" << endl;
}
```

wywołanie funkcji zaprzyjaźnionej

funkcja zaprzyjaźniona jest zdefiniowana w klasie Przyjaciel

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

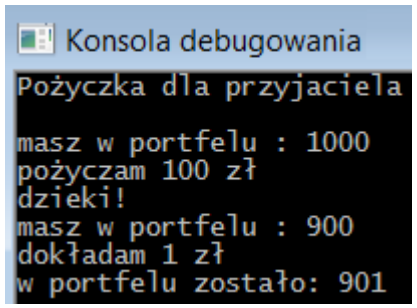


Klasy zaprzyjaźnione

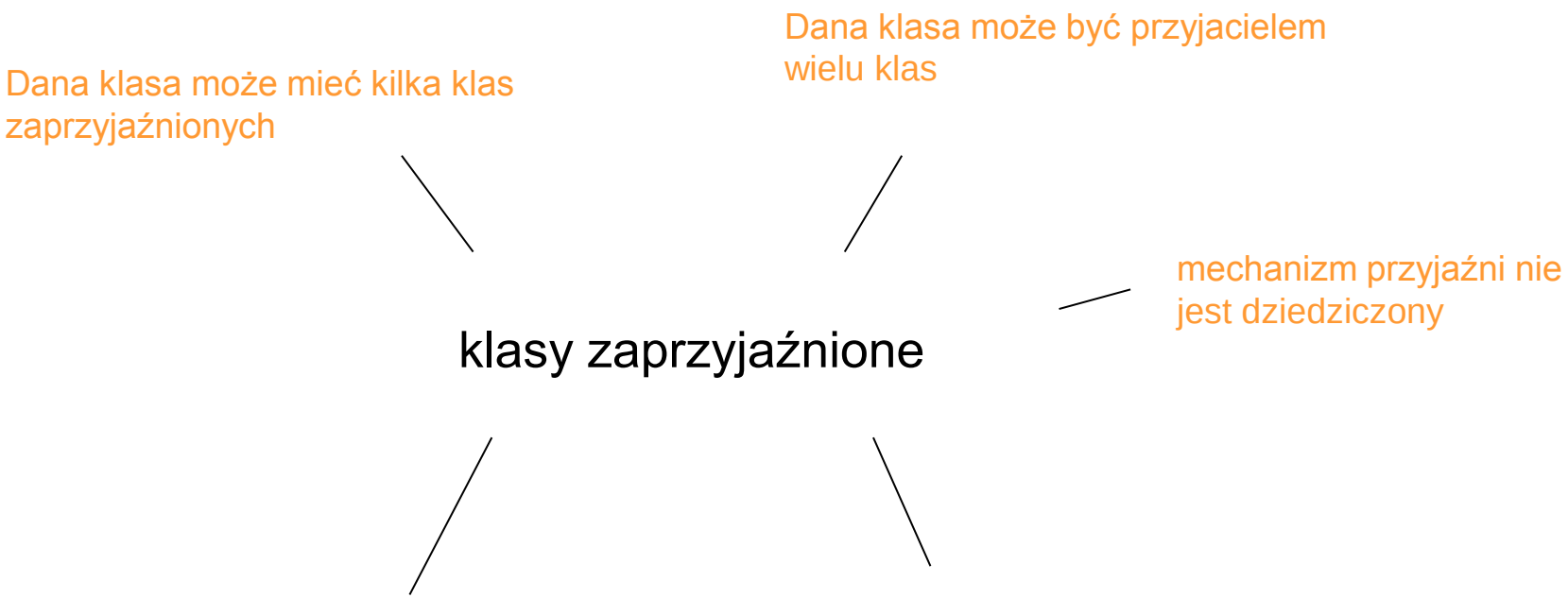
```cpp
#include <iostream>
using namespace std;

// 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:
    // konstrultor 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

```cpp
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 << "dzieki!" << endl;
}

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



Klasy zaprzyjaźnione



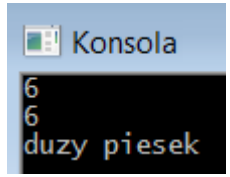
function template

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?



function template

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;
}
```

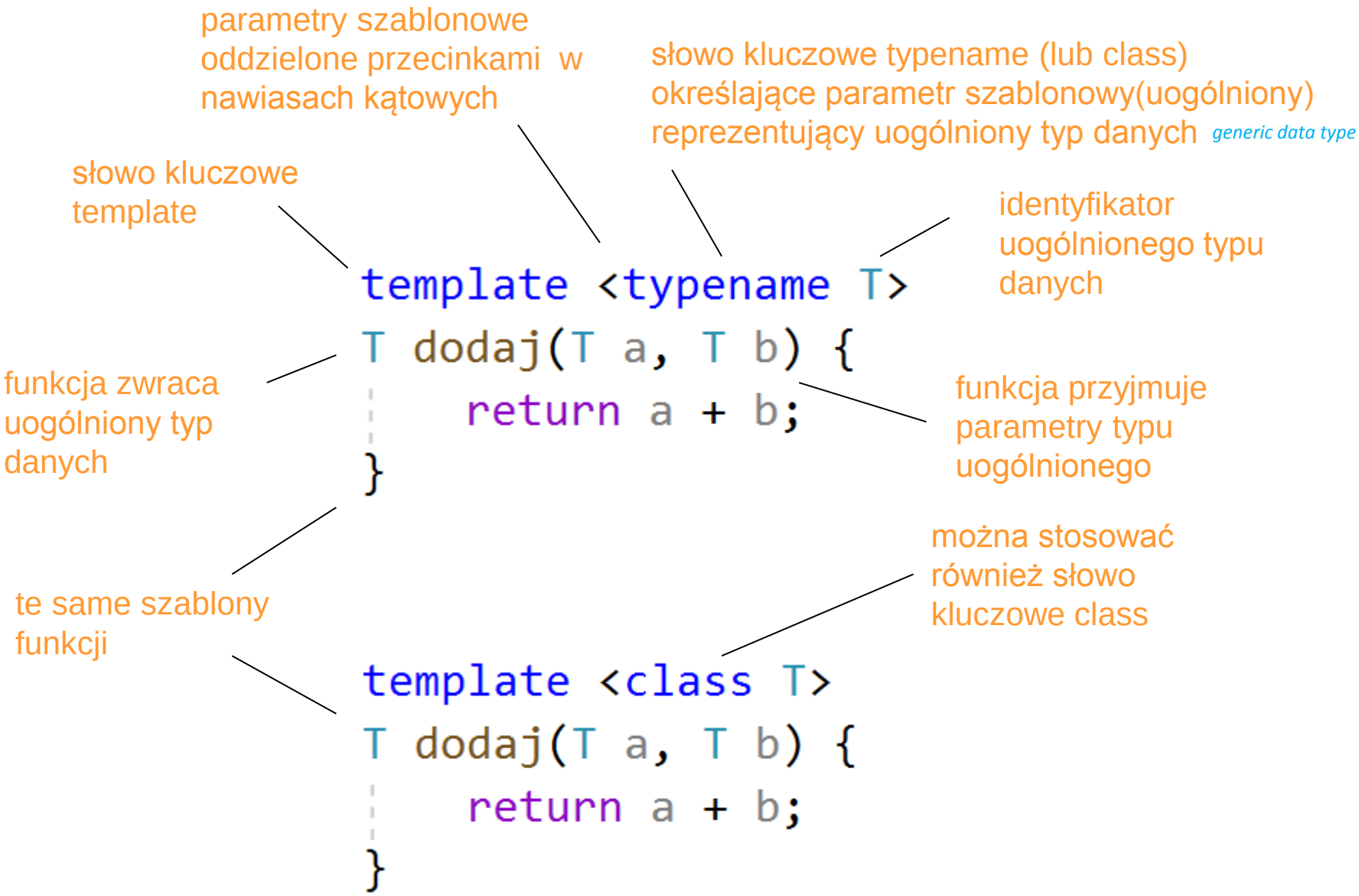
szablon funkcji dodaj()

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

szablony funkcji realizują
tę samą funkcjonalność
dla różnych typów danych

function template

Szablony funkcji



Szablony funkcji - wywołanie

przy wywołaniu funkcji szablonowej 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;
}
```

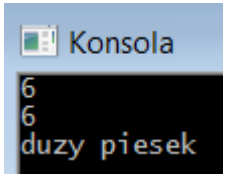
kod funkcji szablonowej jest taki sam dla każdego jej wywołania, różni się skonkretyzowanym typem

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;
}
```

```
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;
}
```

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



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

Szablony funkcji - specjalizacja

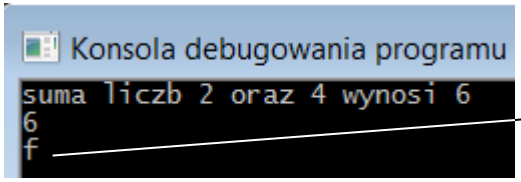
```
#include <iostream>
using namespace std;

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

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;
}
```

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

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

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

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



dlaczego f ?

class template

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();
};

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;
}

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

dzięki generycznemu szablonowi 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

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

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)

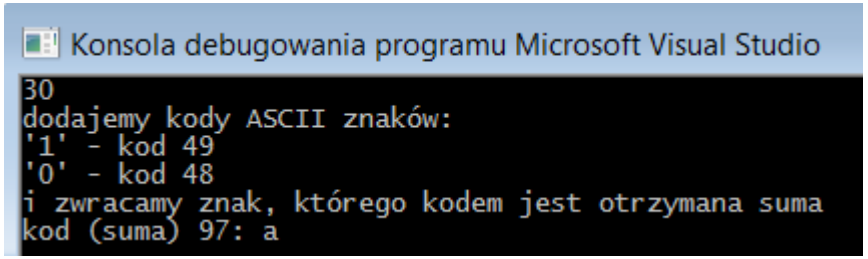
definicja szablonu specjalizowanego klasy Dodawanie, parametrem szablonu jest typ char

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



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;
}
```

przejawy poliformizmu (wielopostaciowości)

specjalizowany szablon funkcji ma odmienną implementację

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

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

```cpp
#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

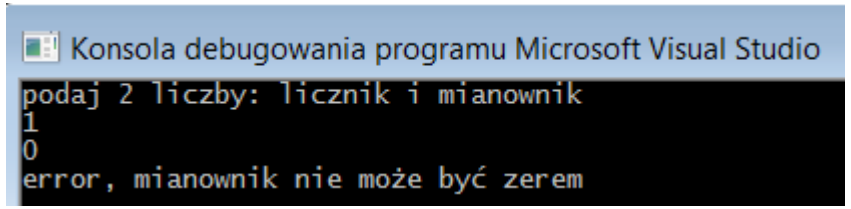
Obsługa błędów i wyjątków

komunikaty błędów zaimplementowane przez programistę

```
#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;
    }
}
```

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

komunikat błędu



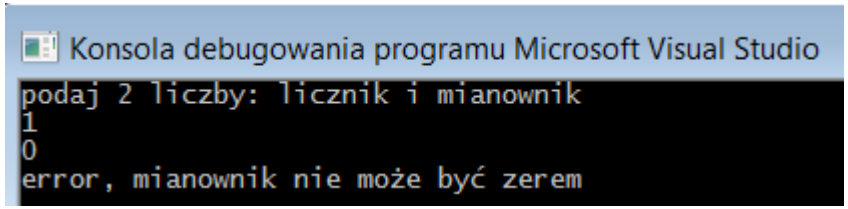
Obsługa błędów i wyjątków

komunikaty błędów zaimplementowane przez programistę

```cpp
#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;
}
```

obsługa błędu za pomocą instrukcji warunkowej oraz funkcji sprawdz()

funkcja weryfikująca błąd i ustawiająca stosowny komunikat



Mechanizm obsługi wyjątków konstrukcja try, throw, catch

mechanizm obsługi wyjątków jest wbudowany w C++

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

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

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

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

Mechanizm obsługi wyjątków

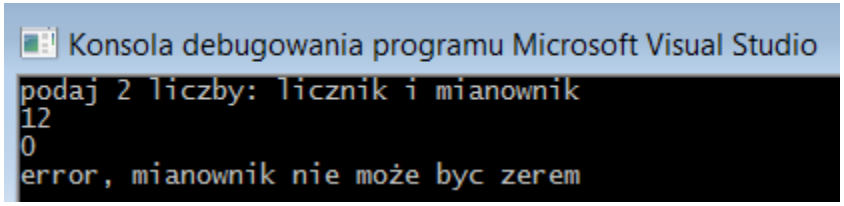
wyjątek typu string

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


int main()
{
    double a, b;
    string komunikat = "error, mianownik nie może byc 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



Mechanizm obsługi wyjątków

własne klasy wyjątków

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


// klasa abstrakcyjna wyjątek z metodą wczysto wirtualną - komunikatem
// korzystamy z poliformizmu dynamicznego
class Wyjatek{
public:
    virtual string komunikat() = 0;
};

// klasa Mianownik - wyjątek dzielenia przez zero
class MianownikZero : public Wyjatek {
public:
    string komunikat() {
        string k = "error, mianownik nie może byc zerem";
        return k;
    }
};

// klasa Liczba - wyjątek wielkości licznika
class LicznikZlaWielkosc : public Wyjatek {
public:
    string komunikat() {
        string k = "error, licznik musi byc większy od 10";
        return k;
    }
};
```

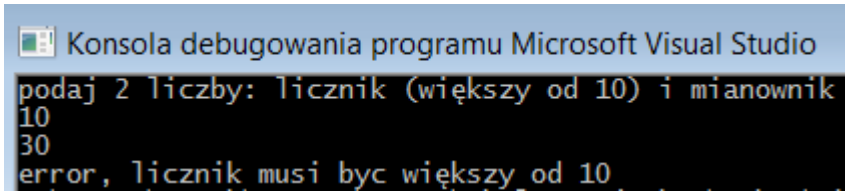
definicje klas wyjątków zaprojektowane przez programistę z abstrakcyjną klasą Wyjatek na górze hierarchii dziedziczenia, w ten sposób skorzystamy z polimorfizmu dynamicznego

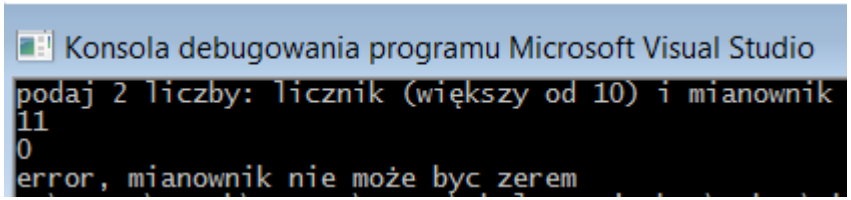
klasy wyjątków dziedziczą publicznie z abstrakcyjnej klasy bazowej

Mechanizm obsługi wyjątków

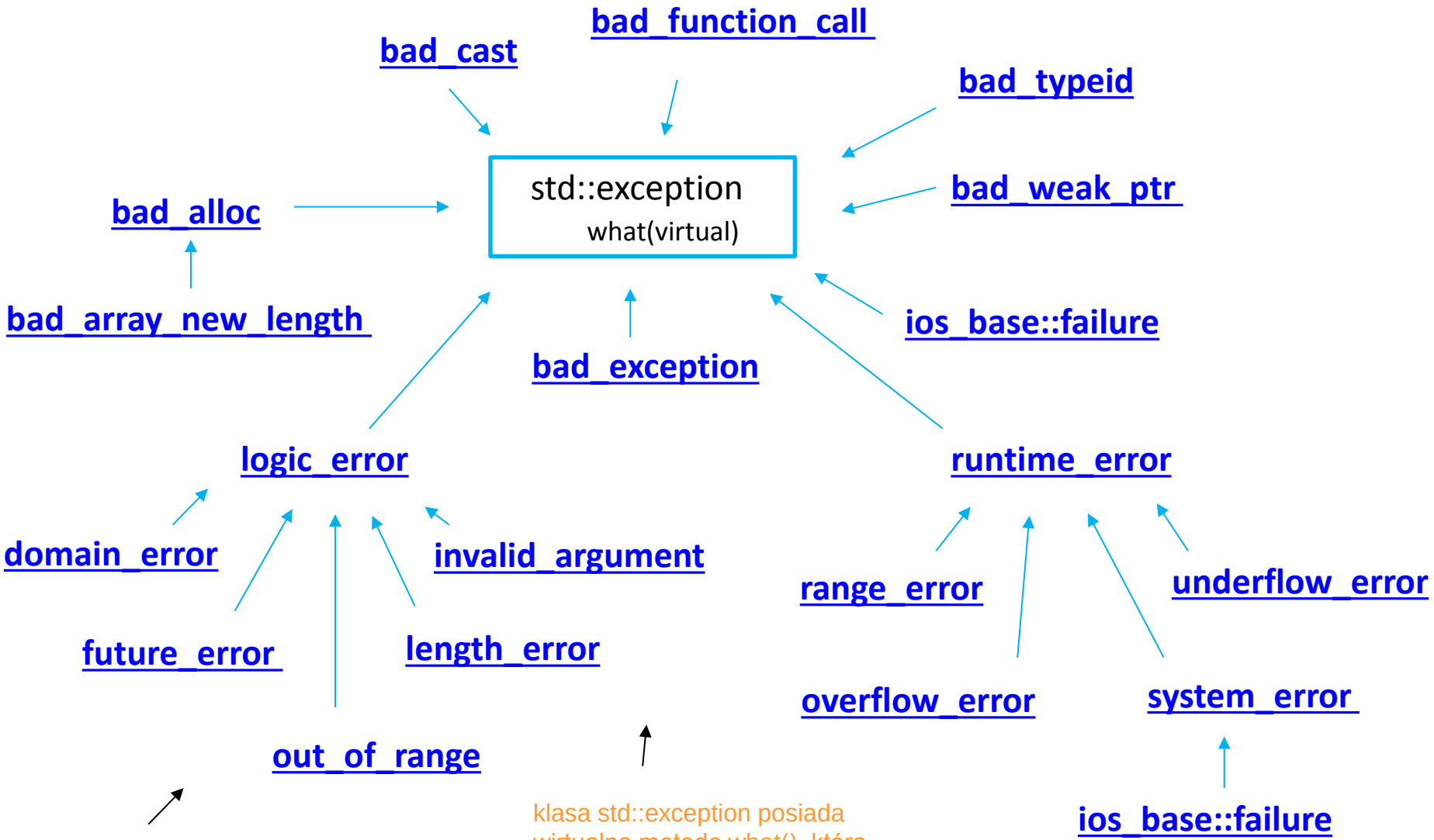
własne klasy wyjątków

```
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 wyjatek 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 sposob korzystamy z mechanizmu polimorfizmu dynamicznego
    catch(Wyjatek& w){
        // wypisujemy stosowny komunikat w zależnosci od obiektu błędu
        cout << w.komunikat();
    }

}
```





Mechanizm obsługi wyjątków



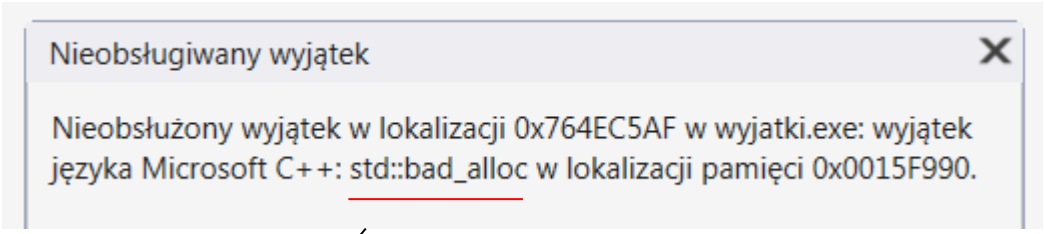
https://www.cplusplus.com/reference/exception/exception/

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 stercie
aż do jej wyczerpania



wyjątek std::bad_alloc

Mechanizm obsługi wyjątków

opis wyjątku – dostęp przez e.what()

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

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

Nieobsługiwany wyjątek

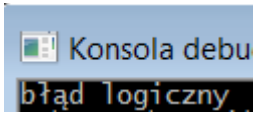
Nieobsłużony wyjątek w lokalizacji 0x764EC5AF w wyjatki.exe: wyjątek języka Microsoft C++: std::logic_error w lokalizacji pamięci 0x0027FC2C.

Kopiuj szczegóły | Rozpocznij sesję rozszerzenia Live Share...

Ustawienia wyjątków

wyjątek std::logic_error nieobsłużony

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

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



wyjątek std::logic_error obsłużony

Mechanizm obsługi wyjątków

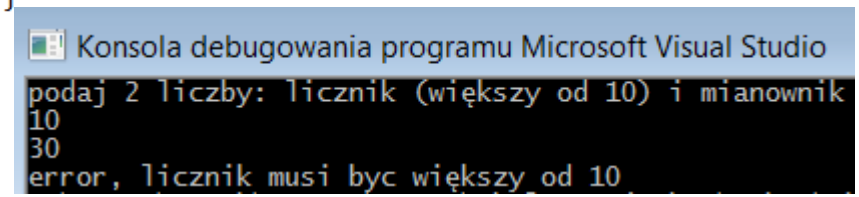
własne klasy wyjątków dziedziczące z bibliotecznej klasy exception

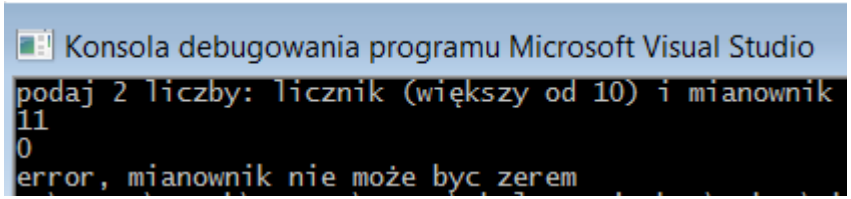
```
#include <iostream>
#include <exception>   // klasa exception
using namespace std;


// 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 byc 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 byc większy od 10";
    }
};
```

Mechanizm obsługi 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 wyjatek 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 sposob korzystamy z mechanizmu polimorfizmu dynamicznego
    catch(exception& e){
        // wypisujemy stosowny komunikat w zależnosci od obiektu błędu
        cout << e.what();
    }

}
```

własne klasy wyjątków dziedziczące z bibliotecznej klasy exception





Standard Template Library STL

standardowa biblioteka szablonów

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

https://www.cplusplus.com/reference/stl/

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.

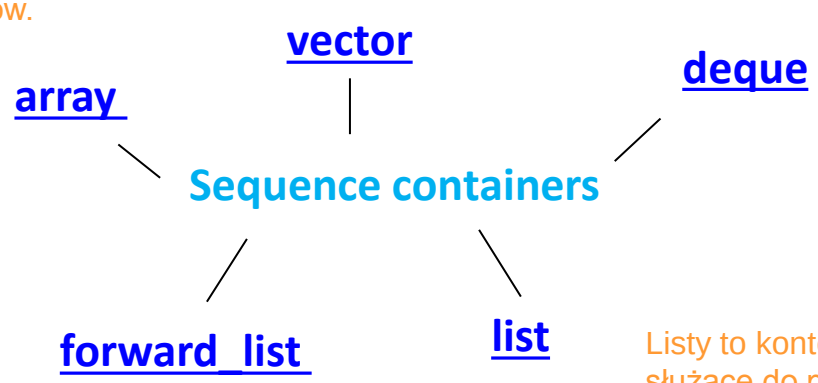
Sequence containers to kontenery sekwencyjne – elementy są przechowywane w kolejności, w której zostały wstawione, można je iterować, czyli przechodzić od elementu kontenera do elementu kontenera po kolei.

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 dynamicznych rozmiarach, 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).

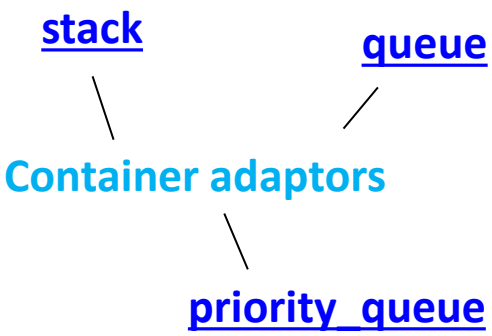
https://www.cplusplus.com/reference/stl/

kontenery

Container adaptors 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)



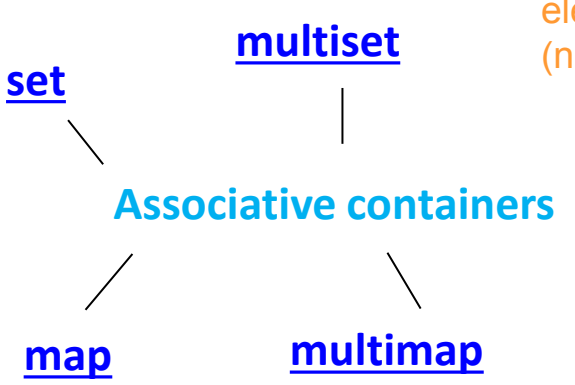
Kolejka, której elementy maja przypisany priorytet (na wyjściu pojawiają się dane, które mają największy priorytet np. kolejka po zakupy, w której stoją kobiety ciężarne lub z dzieckiem na ręku)

https://www.cplusplus.com/reference/stl/

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)

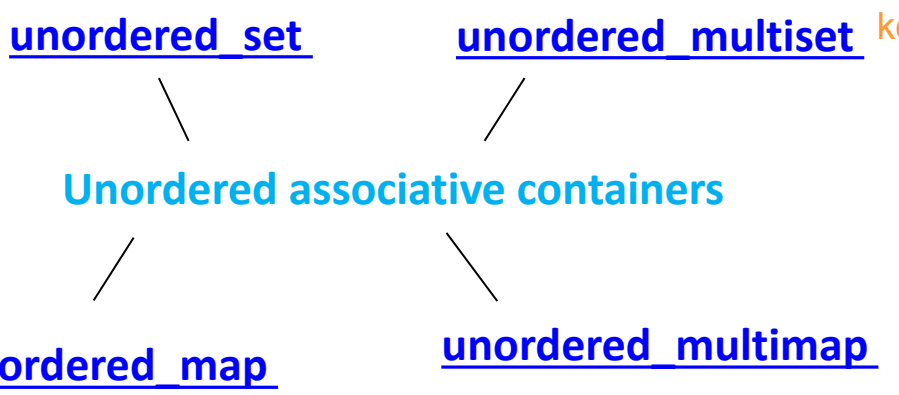
https://www.cplusplus.com/reference/stl/

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 multiset to kontener podobny do Multiset tylko elementy (wartości) nie są ułożone w określonej kolejności.



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.

https://www.cplusplus.com/reference/stl/

Vector

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

int main()
{
    // deklaracja i inicjalizacja vectora
    vector<string> napisy = {"wiosna jest łagodna",
                             "lato jest ciepłe",
                             "jesień jest chłodna",
                             "zima jest mroźna"};

}
```

aby używać wektora trzeba dodać nagłówek

vector jest szablonem klasy

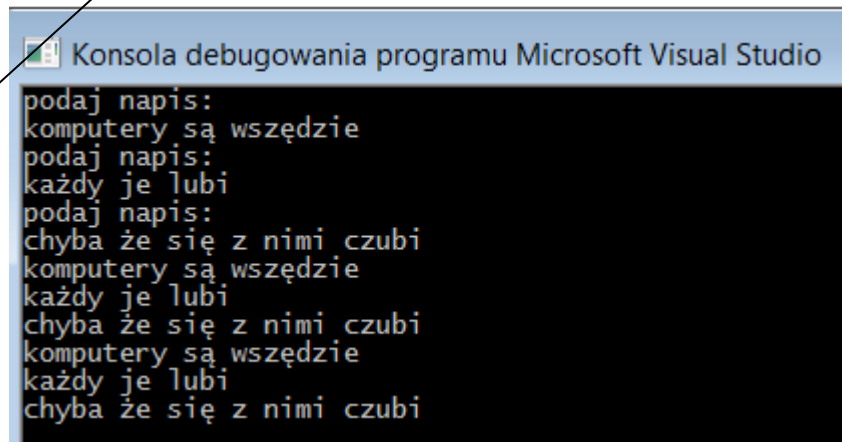
Wektor to tablica dynamiczna. Jest jedną ze kilku struktur danych zwanych kontenerem (obok list, map, set itd.)
Jest zaimplementowany jako szablon klasy, czyli ogólny framework, który może być zdefiniowany jako np.: wektor wartości całkowitych, wektor łańcuchów, wektor obiektów klasy zdefiniowanej przez użytkownika itp.

https://www.cplusplus.com/reference/vector/vector/

Vector

```cpp
#include <iostream>
#include <vector>
#include <string>  // getline()
using namespace std;

int main()
{
    // deklaracja trzyelementowego 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 orginalnym elemencie wektora
    // a nie na kopii
    for (string& napis : napisy)
    {
        cout << "podaj napis:" << endl;
        // pobieramy cała 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;
    }
}
```

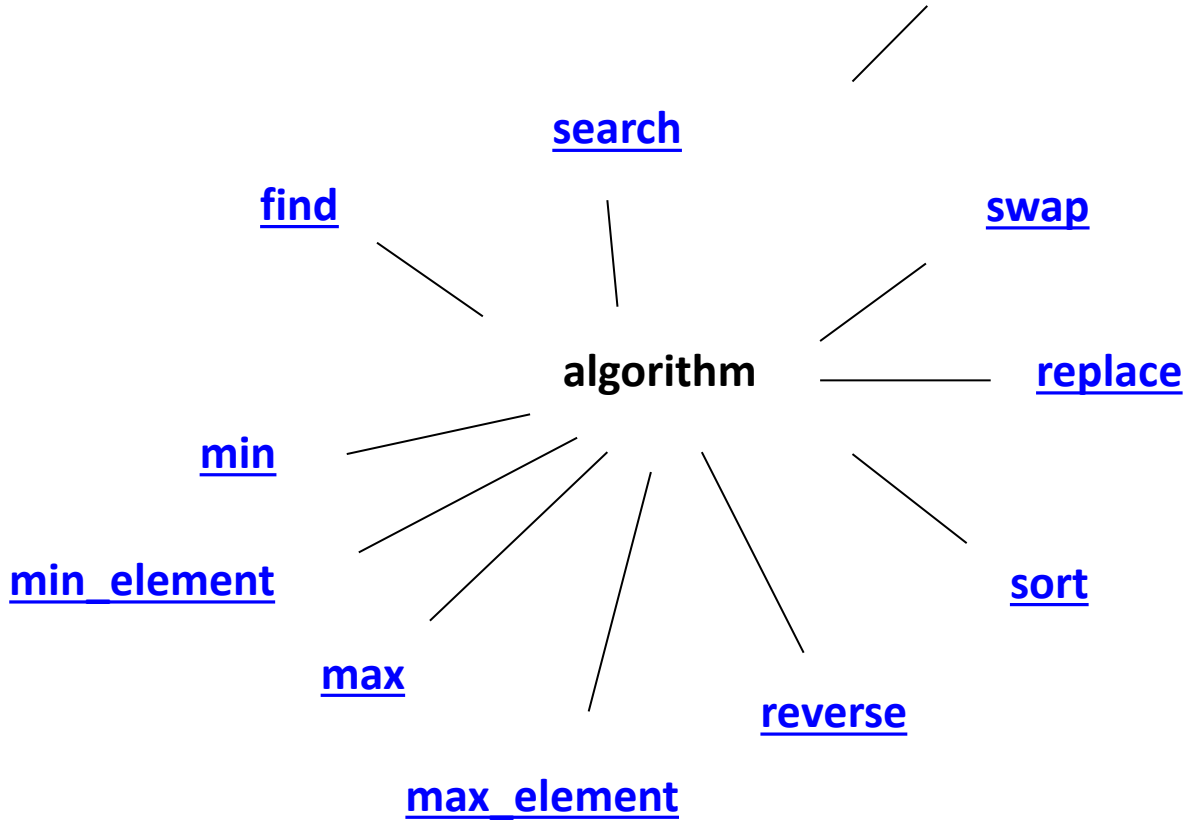
pętla foreach (C++11)



algorytmy

Biblioteka STL zawiera duży zbór algorytmów, które rozwiązują podstawowe problemy, na jakie może się natknąć programista.

Niektóre algorytmy z biblioteki <algorithm>



https://www.cplusplus.com/reference/algorithm/

algorytmy sort

```cpp
#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 };

    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 << " ";
    }
}
```

szablon funkcji sort() służy do sortowania tablic

algorytmy sort

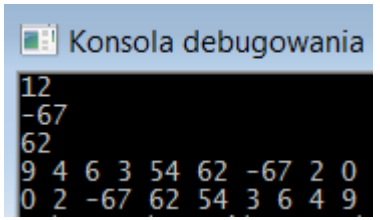
sortowanie wektora

```cpp
#include <iostream>
#include<algorithm> // sort()
#include<vector>    // vector
using namespace std;

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

```cpp
#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 wartosc z zakresu elementów (tablicy)
    cout <<*std::min_element(tab, tab + 9) << endl;
    // zwraca najwiekszą wartosc 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 << " ";
    }
}
```



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

odpowiedź

menu

```cpp
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";
}
```

odpowiedź

zamowienie

```cpp
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 gebie.\n";
    default:
        cout << "Takiej pozycji nie ma w menu.\n";
    }

}
```

odpowiedź

Struktury

```cpp
#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 << "podales nastepujace dane:" << endl;
    cout << "imie: " << pierwsza.imie << endl;
    cout << "nazwisko: " << pierwsza.nazwisko << endl;
    cout << "wiek: " << pierwsza.wiek << endl;


    system("PAUSE");
    return 0;
}
```

odpowiedź

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
```

odpowiedź

Kolej

```
char slowo[6]={"kolej"};          //deklaracja tablicy

for(int i=0; i<5;i++)             //pętla wyświetlająca
    cout<<slowo[i]<<endl;
```

odpowiedź

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];
```

odpowiedź

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;
  }
```

odpowiedź

Funkcje

```cpp
#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;
}
```

odpowiedź

Rekurencja

```cpp
#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

odpowiedź

Klasa Instrument - hermetyzacja (enkapsulacja)

```cpp
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";
}
```

odpowiedź

Dziedziczenie

```
// 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();
}
```