|  |  |
| --- | --- |
|  | Wskaźniki, referencje, struktury, unie i enum – zjazd 5 |

# Przykłady

## Wskaźniki i Referencje

### Wskaźniki w programowaniu są zmiennymi, które przechowują adresy pamięci innych zmiennych. Wskazują one na lokalizację pamięci, gdzie przechowywana jest wartość innej zmiennej.

#include <iostream>

#include <string>

int main**()**

**{**

int liczba **=** 10**;**

int**\*** wskaznik**;**

wskaznik **=** **&**liczba**;** // Inicjalizacja wskaźnika adresem zmiennej liczba

printf**(**"Wartość zmiennej liczba: %d\n"**,** liczba**);** // Wyświetla wartość zmiennej liczba

printf**(**"Adres zmiennej liczba: %p\n"**,** **&**liczba**);** // Wyświetla adres zmiennej liczba

printf**(**"Wartość przechowywana przez wskaźnik: %d\n"**,** **\***wskaznik**);** // Wyświetla wartość przechowywaną przez wskaźnik

printf**(**"Adres przechowywany przez wskaźnik: %p\n"**,** wskaznik**);** // Wyświetla adres przechowywany przez wskaźnik

**}**

Referencje w niektórych językach programowania, takich jak C++, pozwalają tworzyć aliasy dla istniejących zmiennych.

#include <iostream>

**using** **namespace** std**;**

int main**()** **{**

int liczba **=** 10**;**

int**&** referencja **=** liczba**;** // Inicjalizacja referencji jako aliasu dla zmiennej liczba

cout **<<** "Wartość zmiennej liczba: " **<<** liczba **<<** endl**;** // Wyświetla wartość zmiennej liczba

cout **<<** "Wartość referencji: " **<<** referencja **<<** endl**;** // Wyświetla wartość referencji

cout **<<** "Adres zmiennej liczba: " **<<** **&**liczba **<<** endl**;** // Wyświetla adres zmiennej liczba

cout **<<** "Adres referencji: " **<<** **&**referencja **<<** endl**;** // Wyświetla adres referencji

**return** 0**;**

**}**

Ze wskaźnika możemy również pobrać wartość i przypisać ją do zmiennej.

#include <iostream>

**using** **namespace** std**;**

int main**()** **{**

std**::**locale**::**global**(**std**::**locale**(**""**));**

int x **=** 25**;**

int**\*** ptr **=** **&**x**;** // Inicjalizacja wskaźnika ptr adresem zmiennej x

int y **=** **\***ptr**;** // Pobranie wartości spod adresu przechowywanego przez ptr

cout **<<** "Wartość zmiennej x: " **<<** x **<<** endl**;**

cout **<<** "Wartość pobrana ze wskaźnika: " **<<** y **<<** endl**;**

**return** 0**;**

**}**

Wskaźniki pozwalają na dynamiczne alokowanie pamięci, którą przydzielamy za pomocą operatora new i zwalniamy za pomocą operatora delete.

#include <iostream>

int main**()** **{**

std**::**locale**::**global**(**std**::**locale**(**""**));**

// Użycie wskaźnika do alokacji pamięci dla pojedynczej zmiennej int

int**\*** ptr **=** **new** int**(**5**);**

// Wyświetlenie wartości zmiennej int za pomocą wskaźnika

std**::**cout **<<** "Wartość zaalokowanej zmiennej: " **<<** **\***ptr **<<** std**::**endl**;**

// Użycie wskaźnika do alokacji pamięci dla tablicy int o rozmiarze 5

int**\*** tablica **=** **new** int**[**5**];**

// Wypełnienie tablicy wartościami

**for** **(**int i **=** 0**;** i **<** 5**;** **++**i**)** **{**

tablica**[**i**]** **=** i**;**

**}**

// Wyświetlenie wartości tablicy za pomocą wskaźnika

std**::**cout **<<** "Zawartość tablicy: "**;**

**for** **(**int i **=** 0**;** i **<** 5**;** **++**i**)** **{**

std**::**cout **<<** tablica**[**i**]** **<<** " "**;**

**}**

std**::**cout **<<** std**::**endl**;**

// Zwolnienie zaalokowanej pamięci dla pojedynczej zmiennej

**delete** ptr**;**

// Zwolnienie zaalokowanej pamięci dla tablicy

**delete[]** tablica**;**

**return** 0**;**

**}**

## Struktury

Struktura jest typem danych, który pozwala na grupowanie różnych zmiennych różnych typów w jedną całość. Struktury w C++ są przydatne do tworzenia niestandardowych typów danych, które mogą przechowywać różne typy zmiennych oraz funkcje operujące na tych danych. Są stosowane w programowaniu obiektowym, a także jako narzędzie do uporządkowania danych w programach.

#include <iostream>

#include <string>

// Definicja struktury

struct Osoba **{**

std**::**string imie**;**

int wiek**;**

double wzrost**;**

**};**

int main**()** **{**

// Inicjowanie struktury

Osoba o1**;**

o1**.**imie **=** "Jan"**;**

o1**.**wiek **=** 30**;**

o1**.**wzrost **=** 180.5**;**

// Dostęp do składowych struktury

std**::**cout **<<** o1**.**imie **<<** " ma " **<<** o1**.**wiek **<<** " lat." **<<** std**::**endl**;**

**return** 0**;**

**}**

Na ogół są kojarzone z grupowaniem zmiennych, ale mogą mieć też własne funkcje, konstruktory (funkcja wywoływaną przy tworzeniu struktury), destruktory (wywoływana automatycznie przy zwalnianiu pamięci).

#include <iostream>

#include <string>

// Definicja struktury

struct Osoba **{**

std**::**string imie**;**

int wiek**;**

double wzrost**;**

// Funkcja w strukturze

void wypiszInfo**()** **{**

std**::**cout **<<** imie **<<** " ma " **<<** wiek **<<** " lat i wzrost " **<<** wzrost **<<** " cm." **<<** std**::**endl**;**

**}**

**};**

int main**()** **{**

// Inicjowanie struktury

Osoba o1 **=** **{** "Jan"**,** 30**,** 180.5 **};**

// Wywołanie funkcji w strukturze

o1**.**wypiszInfo**();**

**return** 0**;**

**}**

#include <iostream>

#include <string>

// Definicja struktury Osoba

struct Osoba **{**

std**::**string imie**;**

int wiek**;**

// Konstruktor

Osoba**(**std**::**string i**,** int w**)** **:** imie**(**i**),** wiek**(**w**)** **{**

std**::**cout **<<** "Konstruktor: " **<<** imie **<<** " ma " **<<** wiek **<<** " lat." **<<** std**::**endl**;**

**}**

// Destruktor

**~**Osoba**()** **{**

std**::**cout **<<** "Destruktor: " **<<** imie **<<** " zostaje usunięty." **<<** std**::**endl**;**

**}**

**};**

int main**()** **{**

// Tworzenie obiektu osoba1 za pomocą konstruktora

Osoba osoba1**(**"Jan"**,** 30**);**

// Kod programu

**return** 0**;**

**}**

#include <iostream>

#include <string>

// Definicja struktury Samochod

struct Samochod **{**

std**::**string model**;**

int rokProdukcji**;**

// Konstruktor

Samochod**(**std**::**string m**,** int r**)** **:** model**(**m**),** rokProdukcji**(**r**)** **{**

std**::**cout **<<** "Konstruktor: Samochod " **<<** model **<<** " z roku " **<<** rokProdukcji **<<** " zostal stworzony." **<<** std**::**endl**;**

**}**

// Destruktor

**~**Samochod**()** **{**

std**::**cout **<<** "Destruktor: Samochod " **<<** model **<<** " z roku " **<<** rokProdukcji **<<** " zostal usuniety." **<<** std**::**endl**;**

**}**

**};**

int main**()** **{**

// Tworzenie obiektu samochod1 za pomocą konstruktora

Samochod samochod1**(**"BMW"**,** 2020**);**

// Kod programu

**return** 0**;**

**}**

Oczywiście struktury możemy alokować także dynamicznie

#include <iostream>

#include <string>

// Definicja struktury Osoba

struct Osoba **{**

std**::**string imie**;**

int wiek**;**

// Konstruktor

Osoba**(**std**::**string i**,** int w**)** **:** imie**(**i**),** wiek**(**w**)** **{**

std**::**cout **<<** "Konstruktor: " **<<** imie **<<** " ma " **<<** wiek **<<** " lat." **<<** std**::**endl**;**

**}**

// Destruktor

**~**Osoba**()** **{**

std**::**cout **<<** "Destruktor: " **<<** imie **<<** " zostaje usunięty." **<<** std**::**endl**;**

**}**

**};**

int main**()** **{**

// Użycie zmiennych typu Osoba w funkcji main()

Osoba osoba1**(**"Jan"**,** 30**);**

// Użycie operatora new i delete

Osoba**\*** osoba2 **=** **new** Osoba**(**"Anna"**,** 25**);**

// Odwołanie się do zmiennej imie dla osoba1

std**::**cout **<<** "Imie osoby 1: " **<<** osoba1**.**imie **<<** std**::**endl**;**

// Odwołanie się do zmiennej imie dla osoba2

std**::**cout **<<** "Imie osoby 2: " **<<** osoba2**->**imie **<<** std**::**endl**;**

// Zwolnienie pamięci zaalokowanej dla osoba2

**delete** osoba2**;**

**return** 0**;**

**}**

#include <iostream>

#include <string>

// Definicja struktury Samochod

struct Samochod **{**

std**::**string model**;**

int rokProdukcji**;**

// Konstruktor

Samochod**(**std**::**string m**,** int r**)** **:** model**(**m**),** rokProdukcji**(**r**)** **{**

std**::**cout **<<** "Konstruktor: Samochod " **<<** model **<<** " z roku " **<<** rokProdukcji **<<** " zostal stworzony." **<<** std**::**endl**;**

**}**

// Destruktor

**~**Samochod**()** **{**

std**::**cout **<<** "Destruktor: Samochod " **<<** model **<<** " z roku " **<<** rokProdukcji **<<** " zostal usuniety." **<<** std**::**endl**;**

**}**

**};**

int main**()** **{**

// Użycie zmiennych typu Samochod w funkcji main()

Samochod samochod1**(**"BMW"**,** 2020**);**

// Użycie operatora new i delete

Samochod**\*** samochod2 **=** **new** Samochod**(**"Audi"**,** 2019**);**

// Odwołanie się do zmiennej model dla samochod1

std**::**cout **<<** "Model samochodu 1: " **<<** samochod1**.**model **<<** std**::**endl**;**

// Odwołanie się do zmiennej model dla samochod2

std**::**cout **<<** "Model samochodu 2: " **<<** samochod2**->**model **<<** std**::**endl**;**

// Zwolnienie pamięci zaalokowanej dla samochod2

**delete** samochod2**;**

**return** 0**;**

**}**

Możemy mieć również całe tablice takich struktur.

#include <iostream>

#include <string>

// Definicja struktury Osoba

struct Osoba **{**

std**::**string imie**;**

int wiek**;**

// Konstruktor

Osoba**(**std**::**string i**,** int w**)** **:** imie**(**i**),** wiek**(**w**)** **{**

std**::**cout **<<** "Konstruktor: " **<<** imie **<<** " ma " **<<** wiek **<<** " lat." **<<** std**::**endl**;**

**}**

// Destruktor

**~**Osoba**()** **{**

std**::**cout **<<** "Destruktor: " **<<** imie **<<** " zostaje usunięty." **<<** std**::**endl**;**

**}**

**};**

int main**()** **{**

std**::**locale**::**global**(**std**::**locale**(**""**));**

// Tablica statyczna struktur Osoba

Osoba osoby\_statyczna**[**3**]** **=** **{** Osoba**(**"Anna"**,** 25**),** Osoba**(**"Jan"**,** 30**),** Osoba**(**"Ewa"**,** 35**)** **};**

// Pętla operująca na tablicy statycznej

**for** **(**const auto**&** osoba **:** osoby\_statyczna**)** **{**

std**::**cout **<<** osoba**.**imie **<<** " ma " **<<** osoba**.**wiek **<<** " lat." **<<** std**::**endl**;**

**}**

// Tablica dynamiczna struktur Osoba

Osoba**\*** osoby\_dynamiczna **=** **new** Osoba**[**3**]{** Osoba**(**"Adam"**,** 28**),** Osoba**(**"Kasia"**,** 27**),** Osoba**(**"Piotr"**,** 33**)** **};**

// Pętla operująca na tablicy dynamicznej

**for** **(**int i **=** 0**;** i **<** 3**;** **++**i**)** **{**

std**::**cout **<<** osoby\_dynamiczna**[**i**].**imie **<<** " ma " **<<** osoby\_dynamiczna**[**i**].**wiek **<<** " lat." **<<** std**::**endl**;**

**}**

// Zwolnienie pamięci zaalokowanej dynamicznie

**delete[]** osoby\_dynamiczna**;**

**return** 0**;**

**}**

#include <iostream>

#include <string>

// Definicja struktury Samochod

struct Samochod **{**

std**::**string model**;**

int rokProdukcji**;**

// Konstruktor

Samochod**(**std**::**string m**,** int r**)** **:** model**(**m**),** rokProdukcji**(**r**)** **{**

std**::**cout **<<** "Konstruktor: Samochod " **<<** model **<<** " z roku " **<<** rokProdukcji **<<** " zostal stworzony." **<<** std**::**endl**;**

**}**

// Destruktor

**~**Samochod**()** **{**

std**::**cout **<<** "Destruktor: Samochod " **<<** model **<<** " z roku " **<<** rokProdukcji **<<** " zostal usuniety." **<<** std**::**endl**;**

**}**

**};**

int main**()** **{**

// Tablica statyczna struktur Samochod

Samochod samochody\_statyczna**[**2**]** **=** **{** Samochod**(**"BMW"**,** 2020**),** Samochod**(**"Audi"**,** 2019**)** **};**

// Pętla operująca na tablicy statycznej

**for** **(**const auto**&** samochod **:** samochody\_statyczna**)** **{**

std**::**cout **<<** "Samochod " **<<** samochod**.**model **<<** " z roku " **<<** samochod**.**rokProdukcji **<<** std**::**endl**;**

**}**

// Tablica dynamiczna struktur Samochod

Samochod**\*** samochody\_dynamiczna **=** **new** Samochod**[**2**]{** Samochod**(**"Ford"**,** 2022**),** Samochod**(**"Mercedes"**,** 2021**)** **};**

// Pętla operująca na tablicy dynamicznej

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

std**::**cout **<<** "Samochod " **<<** samochody\_dynamiczna**[**i**].**model **<<** " z roku " **<<** samochody\_dynamiczna**[**i**].**rokProdukcji **<<** std**::**endl**;**

**}**

// Zwolnienie pamięci zaalokowanej dynamicznie

**delete[]** samochody\_dynamiczna**;**

**return** 0**;**

**}**

## Enum

Enumy w C++ są używane do definiowania zbioru nazwanych stałych numerycznych, co zapewnia czytelność kodu i ułatwia jego utrzymanie. Mogą one być używane w wielu sytuacjach, w których potrzebne są stałe wartości zrozumiałe dla ludzi i kompilatora.

#include <iostream>

enum Kolor **{** CZERWONY**,** ZIELONY**,** NIEBIESKI **};**

int main**()** **{**

std**::**locale**::**global**(**std**::**locale**(**""**));**

Kolor mojKolor **=** CZERWONY**;**

**switch** **(**mojKolor**)** **{**

**case** CZERWONY**:**

std**::**cout **<<** "Twój kolor to czerwony." **<<** std**::**endl**;**

**break;**

**case** ZIELONY**:**

std**::**cout **<<** "Twój kolor to zielony." **<<** std**::**endl**;**

**break;**

**case** NIEBIESKI**:**

std**::**cout **<<** "Twój kolor to niebieski." **<<** std**::**endl**;**

**break;**

**}**

**return** 0**;**

**}**

#include <iostream>

enum Kolor **{** CZERWONY **=** 5**,** ZIELONY **=** 10**,** NIEBIESKI **=** 15 **};**

int main**()** **{**

Kolor mojKolor **=** NIEBIESKI**;**

std**::**cout **<<** "Wartość mojego koloru: " **<<** mojKolor **<<** std**::**endl**;**

**return** 0**;**

**}**

#include <iostream>

enum class DniTygodnia **{** PONIEDZIALEK**,** WTOREK**,** SRODA**,** CZWARTEK**,** PIATEK**,** SOBOTA**,** NIEDZIELA **};**

int main**()** **{**

std**::**locale**::**global**(**std**::**locale**(**""**));**

DniTygodnia dzis **=** DniTygodnia**::**PIATEK**;**

// Przykład użycia w instrukcji switch

**switch** **(**dzis**)** **{**

**case** DniTygodnia**::**PONIEDZIALEK**:**

std**::**cout **<<** "Dzisiaj jest poniedziałek." **<<** std**::**endl**;**

**break;**

**case** DniTygodnia**::**WTOREK**:**

std**::**cout **<<** "Dzisiaj jest wtorek." **<<** std**::**endl**;**

**break;**

// ... pozostałe dni tygodnia ...

**default:**

std**::**cout **<<** "Dzisiaj jest jakiś dzień." **<<** std**::**endl**;**

**break;**

**}**

**return** 0**;**

**}**

#include <iostream>

enum Kolor **{** CZERWONY**,** ZIELONY**,** NIEBIESKI **};**

void wyswietlKolor**(**Kolor kolor**)** **{**

**switch** **(**kolor**)** **{**

**case** CZERWONY**:**

std**::**cout **<<** "Wybrany kolor: Czerwony" **<<** std**::**endl**;**

**break;**

**case** ZIELONY**:**

std**::**cout **<<** "Wybrany kolor: Zielony" **<<** std**::**endl**;**

**break;**

**case** NIEBIESKI**:**

std**::**cout **<<** "Wybrany kolor: Niebieski" **<<** std**::**endl**;**

**break;**

**}**

**}**

int main**()** **{**

Kolor wybranyKolor **=** ZIELONY**;**

wyswietlKolor**(**wybranyKolor**);**

**return** 0**;**

**}**

## Unie

Unie w C++ to specjalny typ danych, który pozwala przechowywać różne typy danych w tej samej przestrzeni pamięci.

#include <iostream>

#include <iomanip>

// Enum reprezentujący różne jednostki pomiarowe

enum Jednostka **{**

JEDNOSTKA\_INT**,**

JEDNOSTKA\_FLOAT**,**

JEDNOSTKA\_DOUBLE

**};**

// Struktura przechowująca wartość i jej jednostkę

struct Wartosc **{**

Jednostka jednostka**;**

union **{**

int wartosc\_int**;**

float wartosc\_float**;**

double wartosc\_double**;**

**};**

**};**

int main**()** **{**

Wartosc wartosc1**,** wartosc2**;**

// Ustawienie wartości i jednostki dla wartosci1

wartosc1**.**jednostka **=** JEDNOSTKA\_INT**;**

wartosc1**.**wartosc\_int **=** 10**;**

// Ustawienie wartości i jednostki dla wartosci2

wartosc2**.**jednostka **=** JEDNOSTKA\_FLOAT**;**

wartosc2**.**wartosc\_float **=** 3.14f**;**

// Wyświetlenie wartości w zależności od jednostki

**switch** **(**wartosc1**.**jednostka**)** **{**

**case** JEDNOSTKA\_INT**:**

std**::**cout **<<** "Wartosc 1: " **<<** wartosc1**.**wartosc\_int **<<** std**::**endl**;**

**break;**

**case** JEDNOSTKA\_FLOAT**:**

std**::**cout **<<** "Wartosc 1: " **<<** wartosc1**.**wartosc\_float **<<** std**::**endl**;**

**break;**

**case** JEDNOSTKA\_DOUBLE**:**

std**::**cout **<<** "Wartosc 1: " **<<** std**::**setprecision**(**15**)** **<<** wartosc1**.**wartosc\_double **<<** std**::**endl**;**

**break;**

**}**

**switch** **(**wartosc2**.**jednostka**)** **{**

**case** JEDNOSTKA\_INT**:**

std**::**cout **<<** "Wartosc 2: " **<<** wartosc2**.**wartosc\_int **<<** std**::**endl**;**

**break;**

**case** JEDNOSTKA\_FLOAT**:**

std**::**cout **<<** "Wartosc 2: " **<<** wartosc2**.**wartosc\_float **<<** std**::**endl**;**

**break;**

**case** JEDNOSTKA\_DOUBLE**:**

std**::**cout **<<** "Wartosc 2: " **<<** std**::**setprecision**(**15**)** **<<** wartosc2**.**wartosc\_double **<<** std**::**endl**;**

**break;**

**}**

**return** 0**;**

**}**

# Zadania

1. Uruchom i przeanalizuj poniższy kod i napisz:
   1. Objaśniający komentarz do każdej linijki kodu (1 punkt)
   2. Napisz czym się różnią te funkcje i jakie są różnice między zwykłą zmienną, referencją a wskaźnikiem (1 punkt)

#include <iostream>

**using** **namespace** std**;**

void potegaPrzezZmienna**(**int x**)** **{**

x **=** x **\*** x**;**

cout **<<** "Wynik (przez zmienną): " **<<** x **<<** endl**;**

**}**

void potegaPrzezReferencje**(**int**&** x**)** **{**

x **=** x **\*** x**;**

cout **<<** "Wynik (przez referencję): " **<<** x **<<** endl**;**

**}**

void potegaPrzezWskaznik**(**int**\*** x**)** **{**

**\***x **=** **(\***x**)** **\*** **(\***x**);**

cout **<<** "Wynik (przez wskaźnik): " **<<** **\***x **<<** endl**;**

**}**

int main**()** **{**

std**::**locale**::**global**(**std**::**locale**(**""**));**

int liczba **=** 5**;**

cout **<<** "Początkowa wartość: " **<<** liczba **<<** endl**;**

potegaPrzezZmienna**(**liczba**);**

cout **<<** "Wartość po funkcji: " **<<** liczba **<<** endl **<<** endl**;**

liczba **=** 5**;**

cout **<<** "Początkowa wartość: " **<<** liczba **<<** endl**;**

potegaPrzezReferencje**(**liczba**);**

cout **<<** "Wartość po funkcji: " **<<** liczba **<<** endl **<<** endl**;**

liczba **=** 5**;**

cout **<<** "Początkowa wartość: " **<<** liczba **<<** endl**;**

potegaPrzezWskaznik**(&**liczba**);**

cout **<<** "Wartość po funkcji: " **<<** liczba **<<** endl**;**

**return** 0**;**

**}**

1. Uruchom i przeanalizuj poniższy kod. Poszukaj informacji o arytmetyce wskaźników. Napisz w komentarzu linijka po linijce co robi kod oraz napisz czym jest arytmetyka wskaźników

#include <iostream>

**using** **namespace** std**;**

int main**()** **{**

std**::**locale**::**global**(**std**::**locale**(**""**));**

int tablica**[]** **=** **{** 10**,** 20**,** 30**,** 40**,** 50 **};**

int**\*** ptr **=** tablica**;**

cout **<<** "Wartość: " **<<** **\***ptr **<<** endl**;**

ptr**++;**

cout **<<** "Wartość po przesunięciu: " **<<** **\***ptr **<<** endl**;**

ptr **+=** 2**;**

cout **<<** "Wartość po przesunięciu: " **<<** **\***ptr **<<** endl**;**

**--**ptr**;**

cout **<<** "Wartość po przesunięciu: " **<<** **\***ptr **<<** endl**;**

cout **<<** "Wartość: " **<<** **\*(**ptr**+**2**)** **<<** endl**;**

cout **<<** "Wartość: " **<<** **\***ptr **<<** endl**;**

**return** 0**;**

**}**

1. Napisz dwie funkcje, które będą zamieniać wartości dwóch podanych w parametrach funkcji zmiennych. Jedna powinna użyć do tego wskaźników a druga referencji.
2. Napisz czym się różni styczne i dynamiczne alokowanie pamięci.
3. Napisz program, który przyjmie liczbę całkowitą jako argument wywołania programu a w przypadku jego braku zapyta o nią użytkownika. Następnie program stworzy tablicę o wielkości podanej przez użytkownika, zapełni ją losowymi wartościami a następnie posortuje ją, obliczy sumę, medianę oraz średnią a wyniki wypisze na konsoli. (2 punkty)
4. Napisz przykładowy program z własną strukturą. Program powinien mieć przykład zarówno statycznej jak i dynamicznej alokacji tej struktury oraz pracować ze zmiennymi wewnątrz tej struktury. (2 punkty)
5. Napisz własny przykład z enum i unią. W komentarzach do kodu wyjaśnij czym one są.