# Podstawy programowania: Laboratorium nr 7

## Obsługa plików w trybie binarnym.

2017-2018

*mgr inż. Przemysław Walkowiak*
*dr inż. Michał Ciesielczyk*

# Instrukcja

W czasie pisania programu pamiętaj o:

1. dbaniu o czytelność kodu (odpowiednie formatowanie kodu, nazewnictwo zmiennych adekwatne do ich znaczenia, komentarze),
2. dbaniu o czytelność interfejsu z użytkownikiem (w sposób jawny pytaj użytkownika jakie dane ma podać oraz opisuj wyniki, które zwracasz),
3. przed fragmentem implementującym poszczególne zadania umieść komentarz: `/*Zadanie X */` oraz wypisz na ekranie analogiczny komunikat (X jest numerem zadania): `std::cout << "Zadanie X"<< std::endl;`,
4. umieszczeniu wszystkich rozwiązań w jednym pliku, chyba, że w poleceniu napisano inaczej.
5. w zadaniach wymagających udzielenia komentarza bądź odpowiedzi, należy umieścić go w kodzie programu (np. w postaci komentarza albo wydrukować na ekranie).

# Wprowadzenie

## Konstruktory

```
struct Point
{
    double x;
    double y;

    /**
     * Default constructor
     */
    Point() {
        x = 0.0;
        y = 0.0;
    }

    /**
     * Constructor
     */
    Point(double x, double y) {
        this->x = x;
        this->y = y;
    }
}
```

```
/**
 * Function adding two points.
 */
Point add(const Point& a, const Point& b) {
    return Point(a.x + b.x , a.y + b.y );
}
```

Konstruktor jest to funkcja wewnątrz struktury wywoływana w momencie tworzenia obiektu tej struktury. Konstruktor ma zawsze nazwę taką samą jak nazwa struktury i nie zwraca żadnej wartości. Wykorzystuje się ją zazwyczaj do inicjalizowania wartości pól struktury. Konstruktor domyślny (linia 9) nie posiada żadnych argumentów i jest wywoływany następująco:

```
Point punkt; // konstruktor domyślny nadaje polom x i y wartości 0
```

Konstruktory tak jak każda funkcja może mieć dowolną liczbę argumentów. Konstruktor z linii 17 posiada dwa argumenty, które są wykorzystane do inicjalizacji pól struktury. Przykład wykorzystania:

```
Point punkt(2, 5); // konstruktor nadaje x i y wartości 2 i 5
```

## Obsługa plików binarnych

Aby móc korzystać z obsługi plików binarnych z biblioteki standardowej C++ należy załączyć bibliotekę `<fstream>`, tj. dodać do programu dyrektywę: `#include <fstream>`.

### Zapis do pliku

Przykładowy zapis $n$ kolejnych liczb całkowitych do pliku binarnego:

```
string filename = "plik.txt";
ofstream output(filename, ios::binary);
if (output) {
    for (int i = 0; i < n; i++)
        output.write(reinterpret_cast<char*> (&i), sizeof(i));
}
output.close();
```

W linii 2 otwierany jest plik o nazwie `filename` do zapisu w trybie binarnym (świadczy o tym przekazywana flaga `ios::binary`). W linii 3 sprawdzane jest czy udało się utworzyć podany plik. Za zapis kolejnych liczb całkowitych odpowiada instrukcja `write` w linii 5. Funkcja `write` przyjmuje 2 argumenty:

1. binarną reprezentację zapisywanej zmiennej – w postaci tablicy bajtów.
2. rozmiar zapisywanej zmiennej – tj. liczbę bajtów jakie zajmuje jej reprezentacja.

Aby wyciągnąć binarną reprezentację podanej zmiennej można skorzystać z operatora **reinterpret_cast** pozwalającego na reinterpretacje zmiennej jako tablica bajtów (tutaj oznaczana jako **char***). Zwróć uwagę, że podawany jest tutaj adres zmiennej (poprzez operator `&`) a nie jej wartość. Uwaga, sposób ten będzie działać wyłącznie dla typów prostych (takich jak liczby, ale już nie łańcuchy znaków typu `string`).

W celu wyznaczenia liczby bajtów, jakie zajmuje w pamięci podana zmienna najlepiej skorzystać z operatora **sizeof**.

Na koniec, plik wyjściowy jest zamykany (linia 7).

### Odczyt z pliku

Przykładowy odczyt z pliku binarnego:

```
string filename = "plik.txt";
ifstream input(filename, ios::binary);
if (input) {
    int i;
5   while (input.read(reinterpret_cast<char*> (&i), sizeof(i)))
        cout << i << " ";
}
input.close();
```

W linii 2 otwierany jest plik o nazwie `filename` do odczytu w trybie binarnym (świadczy o tym przekazywana flaga `ios::binary`). W linii 3 sprawdzane jest czy udało się utworzyć podany plik. W linii 6 wczytana liczba wyświetlana jest na ekranie.

Za właściwy odczyt kolejnych liczb z pliku odpowiada instrukcja `read` w linii 5. Zwróć uwagę, że funkcja ta wykonywana jest w pętli – tak długo, aż funkcja `read` nie będzie mogła odczytać dalszych danych. Analogicznie, do funkcji `write`, funkcja `read` przyjmuje 2 argumenty – referencję na zmienną do której wczytywane będą dane oraz liczbę wczytywanych bajtów.

Na koniec, czytany plik jest zamykany (linia 8).

# Zadania

## Zadanie 1

Wygeneruj wektor $n$ pseudolosowych liczb zmiennoprzecinkowych (**float**). Liczbę $n$ wczytaj z klawiatury. Następnie, zapisz wszystkie wartości z wygenerowanej tablicy w trybie:

a) tekstowym, oraz
b) binarnym.

Porównaj wielkości plików oraz ich zawartość (korzystając np. z aplikacji notatnika) i zapisz spostrzeżenia w komentarzu.

**Wskazówka** Do generowania zmiennoprzecinkowych liczb losowych możesz wykorzystać następującą funkcję:

```
#include <random>

float random() {
    static default_random_engine e{};
    uniform_real_distribution<float> d;
    return d(e);
}
```

## Zadanie 2

Zaimplementuj wczytywanie ciągu liczb z pliku binarnego. Zadanie wykonaj w dwóch wariantach:

a) wczytywanie do wektora liczb całkowitych (**int**), oraz
b) wczytywanie do wektora liczb zmiennoprzecinkowych (**float**).

Następnie, wczytaj zawartość całego pliku binarnego utworzonego w poprzednim zadaniu (w obu wariantach). Co się stanie, gdy w taki sam sposób spróbujesz wczytać plik tekstowy z poprzedniego zadania?

## Zadanie 3

Zaprojektuj i zaimplementuj obsługę liczb zespolonych (`complex`) z wykorzystaniem struktur. Deklarację struktury oraz implementacje funkcji umieść w osobnych plikach (np. w `complex.hpp` oraz `complex.cpp`). Zdefiniuj następujące operacje arytmetyczne działające na liczbach zespolonych:

a) dodawanie: `complex add(const complex& a, const complex& b)`,
b) odejmowanie: `complex subtract(const complex& a, const complex& b)`,
c) mnożenie: `complex multiply(const complex& a, const complex& b)`, oraz
d) przyrównanie: `bool equals(const complex& a, const complex& b)`.

Zaimplementuj podstawowe konstruktory. Przetestuj swoją implementację sprawdzając wszystkie operacje na przykładowych danych.

**Wskazówka 1** Liczby zespolone mają postać $z = a + bi$, gdzie $a$ jest częścią rzeczywistą, $b$ jest częścią urojoną, a $i$ jednostką urojoną. Jednostka urojona $i$ ma wartość $i = \sqrt{-1}$, a liczby $a$ i $b$ są liczbami rzeczywistymi.

**Wskazówka 2** Działania na liczba zespolonych są określone wzorami:

$$(a + bi) \pm (c + di) = (a \pm c) + (b \pm d)i \tag{1}$$

$$(a + bi)(c + di) = ac + (bc + ad)i + bdi^2 = (ac - bd) + (bc + ad)i \tag{2}$$

**Dodatkowe informacje:**

- Liczby zespolone – `http://pl.wikipedia.org/wiki/Liczby_zespolone`
- Działania na liczbach zespolonych – `http://pl.wikipedia.org/wiki/Liczby_zespolone#Dzia.C5.82ania`

### Zadanie 4

Napisz program zapisujący do pliku binarnego 10 liczb zespolonych (możesz je wylosować lub wczytać z klawiatury).

### Zadanie 5

Napisz program odczytujący z pliku binarnego – utworzonego w poprzednim zadaniu – ciąg liczb zespolonych oraz wyświetl je na ekranie.

## Na następne zajęcia

- Przestrzenie nazw: `http://en.cppreference.com/w/cpp/language/namespace`
- Przeciążanie operatorów: `http://en.cppreference.com/w/cpp/language/operators`