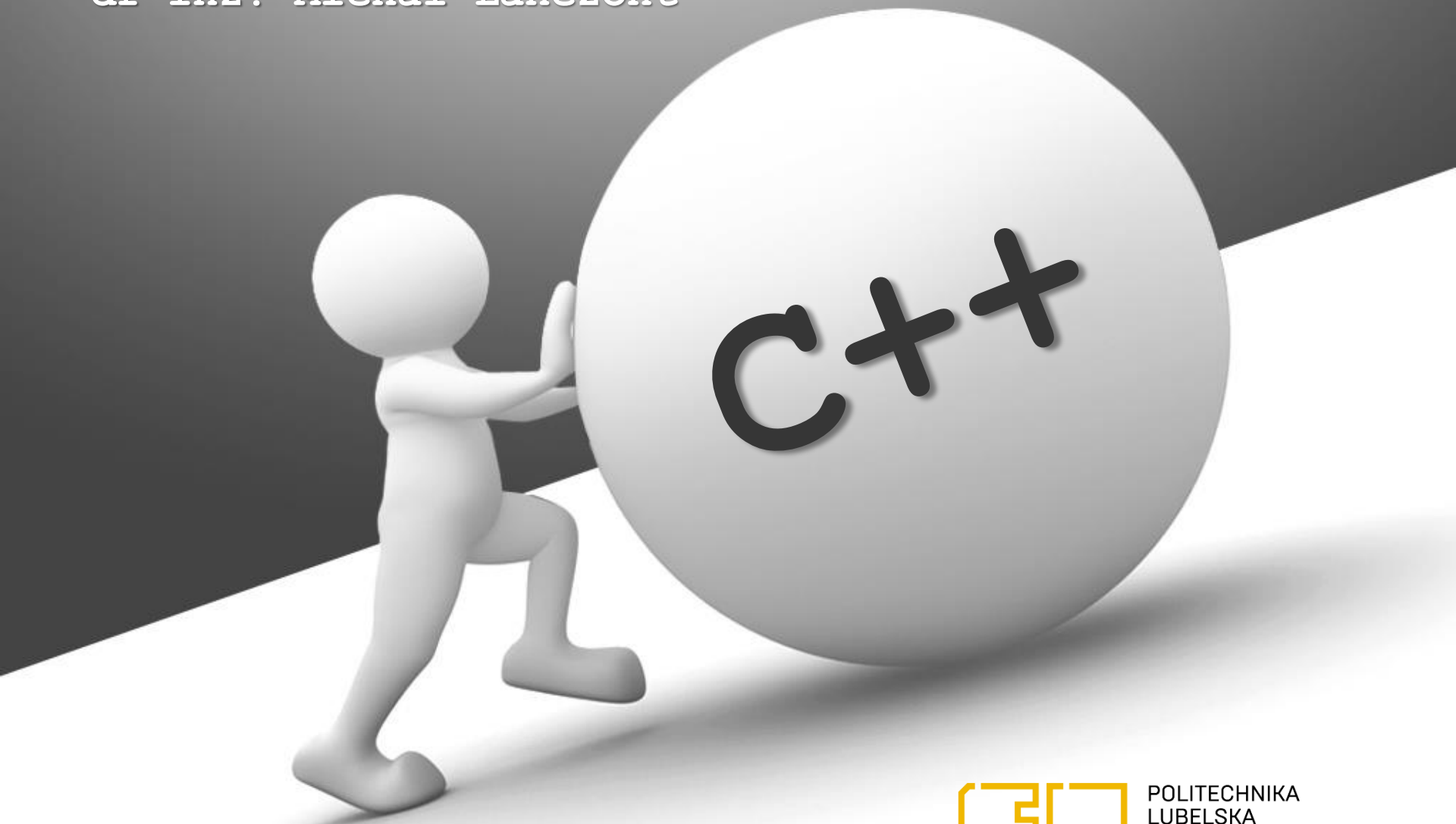
Informatyka II

dr inż. Michał Łanczont



POLITECHNIKA
LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI
I INFORMATYKI

Zakres wykładu

- Stan strumienia wejściowego – błędy
- Wyjątki
- Programowanie obiektowe – podstawowe pojęcia
- Wskaźnik, referencja
- Klasa – obiekt
- Wskaźnik *this*

Stan strumienia

Strumienie cin i cout zawierają daną składową bitową (dziedziczoną z ios_base) opisującą stan strumienia:

- eofbit – ustawia wartość kiedy cin napotka koniec pliku
- badbit – ustawia wartość gdy strumień z jakiegoś powodu został uszkodzony (błąd odczytu danych z pliku)
- failbit - ustawia wartość kiedy metody cin lub cout wykonane zostaną z błędem (błędny typ danych, brak prawa dostępu do pliku itp.)

Stan strumienia

Metody umożliwiające monitorowanie stanu strumienia oraz modyfikacje bitów stanu:

- good() – zwraca i … strumień nadaje się do użytku
- eof() – wraca tru… …niec pliku (ustawiony eofbit)
- .rdstate() – …aca true jeżeli ustawiony jest badbit
- fail() – zwraca true jeżeli ustawiony jest badbit lub …
- rdstate() zwraca … …enia
- clear() – czyszczenie bitów stanu

0 – stan OK

1 – eofbit

2 - failbit

4 - badbit

Stan strumienia

- Próba wprowadzenia niekompatybilnej z zmienną danej powoduje że metoda `cin` zwraca wartość `false`
- Próba ponownego odczytu danych ze strumienia wejściowego będzie powodowała błąd, strumień pozostanie zamknięty do czasu wyzerowania bitu błędu oraz opróżnienia bufora metody
- Bit błędu można wyzerować wywołując funkcję `clear()`

Stan strumienia

- **Opróżnienie bufora cin wymaga odczytania przechowywanych w nim znaków**

```
cin.clear();
while (!isspace(cin.get()))
   continue;
```

lub

```
cin.clear();
while (cin.get() != '\n')
    continue;
```

Stan strumienia

```
18  void czyszcz
19  {
20      std::cou
21      std::cin
22      while (s
23          cont
24      std::cou
25      test();
26  }
27  int suma()
28  {
29      int sum=
30      while(st
31          sum+
32      return s
33  }
34  void test()
35  {
36      std::cou
37      std::cou
38      std::cou
39      std::cou
40      if(std::
41          czys
42  }
```

```
ndl;
endl;
td::endl;
dl;
;
dl;
```

```
C:\Windows\system32\cmd.exe
Dane wejściowe: 1 2 3 4 5 a
Suma=15
Stan strumienia: 4
Test .good(): false
Test .bad(): false
Test .fail(): true
Czyszczenie strumienia wejściowego
Ponowny test strumienia wejściowego
Stan strumienia: 0
Test .good(): true
Test .bad(): false
Test .fail(): false
Nowe dane wejściowe: 1 2 3 4 5 0
Suma=15
Stan strumienia: 0
Test .good(): true
Test .bad(): false
Test .fail(): false

Press any key to continue . . .
```

Metody klasy iostream

- Metoda .get() pobiera ze strumienia wejściowego dane i przypisuje je do wskazanej zmiennej.
- Metoda umożliwia pobranie pojedynczego znaku lub ciągu znaków
- Metoda poza znakami pobiera także znak pusty (spacje, tabulacji) czy znak przejścia do nowego wiersza

Metody klasy iostream

- Metoda może być wywoływana na dwa sposoby:

```
cin.get(zmienna);
zmienna = cin.get();
```

- Metoda pozwala na pobranie danych ze strumienia wejściowego i przypisanie ich do zmiennej typu **char**

10:40

Metody klasy iostream

```
1  #include <iostream>
```

```
C:\Windows\system32\cmd.exe
Test metody: cin.get(char)
Wprowadź znaki: Testowy zestaw danych.
Testowy zestaw danych.

Test metody: char=cin.get()
Wprowadź znaki: Testowy zestaw danych.
Testowy zestaw danych.

Test standardowego wprowadzania danych
Wprowadź znaki: Testowy zestaw danych.
Testowyzestawdanych.

Press any key to continue . . .
```

Metody klasy iostream

- Metoda .get() pozwala na pobranie z strumienia wejściowego ciągu znakowego i przypisanie go do tablicy

```
cin.get(tablica,limit);
cin.get(tablica,limit,znak);
```

- Metoda przypisuje do tablicy ciągu znaków pobrany z strumienia wejściowego i kończy go znakiem '\0'

Metody klasy iostream

```
1  #include <iostream>
2  void drukujPoZnaku(char*,int);
3  void znak(char*,int);
4  int main()
5  {
6      char c[50];
7      znak(c,50);
8      std::cout << "Wprowadź ciąg znakowy: ";
9          std::cin.get(c,50,'.');
10             std::cout << "Wprowadzony ciag:" << c << std::endl;
```

```
C:\Windows\system32\cmd.exe
Wprowadź ciąg znakowy: Testowy ciąg znakowy.
Wprowadzony ciag:Testowy ciąg znakowy
Drukowanie: Testowy ciąg znakowy\0_____________________________
Wprowadź ciąg znakowy: Wprowadzony ciag:
Drukowanie: \0_____________________________________________

Press any key to continue . . .
```

```
31         for(int i=0;i<limit;i++)
32             tab[i]='_';
33     }
```

10:40

Metody klasy iostream

```
1  #include <iostream>
2  void czytanie(char*);
3  int main()
4  {
5
6
7
8
9
10
11
12
13
14     std::cout << ":>";
15     std::cin >> tab;
16     std::cin.ignore(1024,'\n');
17 }
```

C:\WINDOWS\system32\cmd.exe

```
Podaj imię:
:>Jakub Adam Rupert
Podaj nazwisko:
:>Kowalski vel Wąsik
Witaj Jakub Kowalski
Press any key to continue . . .
```

do

10:40 Błędy w działaniu programu

- Każda pisana aplikacja jest narażona na błędy
  - Niewłaściwe dane wejściowe
  - Błędy przy przetwarzaniu danych
  - Kontrolowane przerwanie działania kodu
  - Nieprzewidziane sytuacje
- Reakcję programu na sytuacje nieprzewidziane w algorytmie należy zaprogramować w kodzie

Reakcja na błędy

- Reakcję na każdą z sytuacji „awaryjnych” można opisać za pomocą odpowiednio zbudowanej konstrukcji warunkowej
- Testując instrukcją `if()` przedmiot analizy (zmienna, funkcja …) dla wyniku `false` opisujemy działanie na sytuację awaryjną i przerywa się działanie funkcji instrukcją `return`

Przykład

kod2

```
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 41
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 32
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 23
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 14
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 5
Brak wolnych kartek
Brak wolnych kartek
Brak wolnych kartek
Brak wolnych kartek
Brak wolnych kartek

Press any key to continue...
```

Wyjątki

W języku C++ dostępny jest mechanizm wyjątków realizowany przez funkcje:

- `try{testowany kod}` – testowania kodu
- `throw wyjątek;`– wysłanie kodu wyjątku – dowolna wartość dowolnego typu
- `catch(const typ){}` – przechwycenie wyjątku
- `catch(…)` – uniwersalne przechwycenie dowolnego typu

Przykład

```
#include <iostream>
```

kod3

```
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 41
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 32
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 23
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 14
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 5
Brak wolnych kartek

Press any key to continue...
```

```
21      else
22          throw "Brak wolnych kartek";
23      page-=use;
24      cout << "Zasobnik papieru: " << page << endl;
25  }
```

Przykład

kod3

```
Zasobnik papieru: 40
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 34
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 28
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 22
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 16
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 10
Testowanie działania wyjątków w języku C++
Zasobnik papieru: 4
Brak wolnych kartek

Press any key to continue...
```

Wyjątki – strumień cin

- Metody obsługi strumieni wejściowego i wyjściowego we współpracy z metodami z biblioteki <exception> dostarczają narzędzi detekcji bledów
- Po aktywacji metodą .exceptions() metoda catch() pozwala na przechwycenie błędu
- Metoda .what() pozwala na wyświetlenie informacji o przechwyconym wyjątku

Wyjątki

- Z wykorzystaniem biblioteki exception i instrukcji try catch można zaprogramować reakcję na „wyjątek” wywołany pojawieniem się błędu obsługi strumienia
- Identyfikacja błędu jest możliwa dzięki funkcji what()

Przykład

```
23  void czyszczenie()
```

kod4

```
Dane wejściowe: 10 20 30 40 a
Wyjątek:basic_ios::clear: iostream error
Suma=0
Stan strumienia: 4
Test .good(): false
Test .fail(): true
Czyszczenie strumienia wejściowego
Ponowny test strumienia wejściowego
Stan strumienia: 0
Test .good(): true
Test .fail(): false

Press any key to continue...
```

```
45      cout << "Test .fail(): " << cin.fail() << endl;
46      if(cin.rdstate())
47          czyszczenie();
48  }
```

Złożone typy danych

- W języku C++ można zdefiniować własny (złożony) typ danych zaadoptowany dla konkretnego algorytmu
- Struktura – `struct` – deklaratywny typ danych
- Struktura budowana jest z wykorzystaniem:
  - Typów podstawowych – `int`, `float`, `char` …
  - Tablic – `int[]`, `char[]`, …
  - Struktur – `struct name`, `struct name[]`
  - Unii – `union name`, `union name[]`
- Tablica strukturalna

Przykład

C:\Windows\system32\cmd.exe

```
Podaj dane pracownika:
Imię (imiona) pracowniak: Adam Antonii
Nazwisko pracownika: Kwiatkowski
Numer telefonu pracownika: 555666777
Zarobki: 7689.34
----------------------------
Pan/Pani Adam Antonii Kwiatkowski
Nr. kontaktowy: +48555666777
Pensja: 7689.34 zł

Press any key to continue . . .
```

```
36      cout.precision(2); cout.setf(ios_base::fixed,ios_base::floatfield);
37      cout << x.zarobki << " zł" << endl;
38  }
```

Wskaźniki i referencje

- **Wskaźnik jest typem zmiennej przechowującym adres obiektu tego samego typu**

```
typ *nazwa;
```

- **Operator *referencji* zwraca adres w pamięci pod którym zmienna przechowuje swoją zawartość**

```
nazwa = & zmienna;
```

- **Operator *dereferencji* (*wyłuskania*) umożliwia zwrócenie przez wskaźnik zawartości zmiennej którą wskazuje**

```
typ zmienna2 = *nazwa;
```

Wskaźnik pusty – nullptr

- Literał nullptr reprezentuje wskaźnik pusty, adresujący na wartość 0
- Literał można przypisać tylko do wskaźnika, dowolnego typu

```
int *A = nullptr;
```

- Wyłuskanie wartości wskazywanej przez wskaźnik o wartości nullptr zwraca wartość pustą

Przykład

```cpp
#include <iostream>
using namespace std;
int main()
{
    int x = 10;
    int *A = &x;
    cout << "x = " << x << endl;
    cout << "A = " << A << endl;
    cout << "*A = " << *A << endl;
    A = nullptr;
    cout << "A = " << A << endl;
    cout << "*A = " << *A << endl;
}
```

```
C:\WINDOWS\system32\cmd.exe
x = 10
A = 0x57571ffcc4
*A = 10
A = 0
*A =
Press any key to continue . . .
```

Wskaźnik a typ char

- **Wskaźnik na ciąg tekstowy**

```
char *nazwa;
nazwa = ”zadany ciąg tekstowy”;
```

- **Inicjacja wskaźnika znakowego ciągiem tekstowym**
- `char *nazwa = ”zadany ciąg tekstowy.”;`
- **Nie można zainicjować wskaźnika na znak**

~~`char *nazwa = 'a';`~~

Przykład

```
 1  #include <iostream>
 2  using namespace std;
 3
 4  int main()
```

C:\WINDOWS\system32\cmd.exe

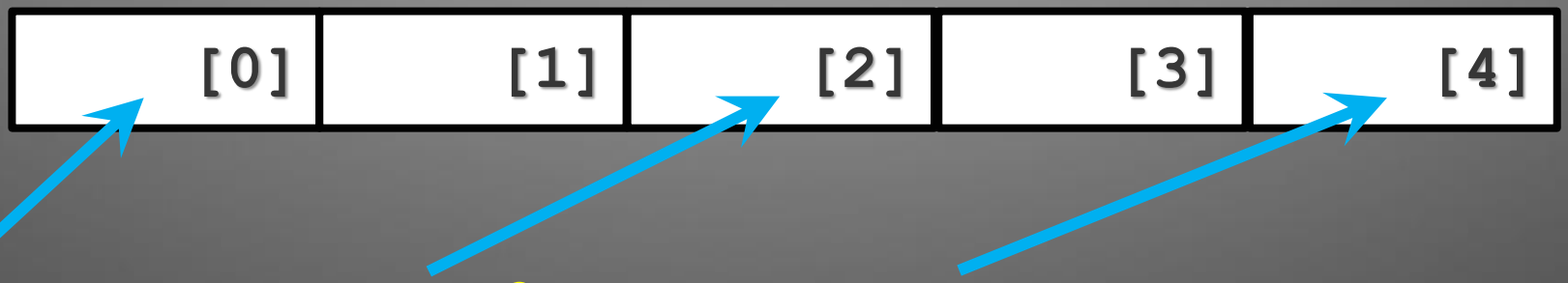
```
 &nazwa      : *nazwa     : nazwa
0xa73a9ff808: A          : Alfa i omega.
0xa73a9ff800: A          : Alfa i omega.
To samo.

 &nazwa      : *nazwa     : nazwa
0xa73a9ff808: A          : A teraz inny tekst o zadanej długości.
0xa73a9ff800: A          : Alfa i omega.
Co innego.

Press any key to continue . . .
```

```
18      cout << &tekst1 << ": " << *tekst1 << "\t\t: " << tekst1 << endl;
19      cout << &tekst2 << ": " << *tekst2 << "\t\t: " << tekst2 << endl;
20      if(tekst1 == tekst2)
21          cout << "To samo." << endl;
22      else
23          cout << "Co innego." << endl;
24  }
```

Wskaźnik a tablica

- Przechowuje określoną liczbę wartości wybranego typu

```
int nazwa[5];
```




```
nazwa;  nazwa[2];   nazwa+4;
```

- Nazwa tablicy jest wskaźnikiem na początek obszaru pamięci przydzielony dla tablicy

Przykład

```
C:\WINDOWS\system32\cmd.exe
tab      :0xfd6e7ff960
*tab     :12
tab[3]   :1
tab+4    :0xfd6e7ff970
*tab+4   :16
*(tab+4):786

Press any key to continue . . .
```

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

int main()
{
    int tab[] = {12,34,54,1,786};
    cout << "tab\t:" << tab << endl;
    cout << "*tab\t:" << *tab << endl;
    cout << "tab[3]\t:" << tab[3] << endl;
    cout << "tab+4\t:" << tab+4 << endl;
    cout << "*tab+4\t:" << *tab+4 << endl;
    cout << "*(tab+4):" << *(tab+4) << endl;
}
```

Funkcja a wskaźnik

- Argumentem funkcji może być zmienna, referencja lub wskaźnik
- Przekazanie do funkcji adresu w pamięci powoduje że wewnątrz funkcji można modyfikować wskazana wartość
- Tablica jest zawsze przekazywana do funkcji jak wskaźnik

10:40

Przykład

```
C:\WINDOWS\system32\cmd.exe
A: 10
A: 10
A: 35
A: 60

Press any key to continue . . .
```

```cpp
#include <iostream>
using namespace std;
void test1(int x) {x+=25;}
void test2(int &x) {x+=25;}
void test3(int *x) {(*x)+=25;}
int main()
{
    int A=10;
    cout << "A: " << A << endl;
    test1(A);
    cout << "A: " << A << endl;
    test2(A);
    cout << "A: " << A << endl;
    test3(&A);
    cout << "A: " << A << endl;
}
```

Klasa

- C
  - Definicja struktury
  - Zmienna strukturalna
  - Funkcje operujące na zmienne
- C++
  - Definicja klasy
    - Cechy - dane
    - Metody – funkcje
  - Obiekt na bazie klasy

Programowanie obiektowe

- Styl programowania w którym programy definiuje się za pomocą **obiektów** – elementów łączących *stan* (**dane**, pola) i *zachowanie* (**procedury**, metody).
- Obiektowy program komputerowy wyrażony jest jako zbiór takich obiektów, komunikujących się pomiędzy sobą w celu wykonywania zadań.
- Każdy z obiektów utworzony jest zgodnie z zasadami zdefiniowanymi w danej klasie

Programowanie obiektowe

- Obiekt – każdy element rzeczywisty i abstrakcyjny:
  - rzeczy, osoby, zjawiska itp
  - myśli, uczucia, wrażenia itp
- Każdy obiekt ma cechy opisujące go:
  - waga, wymiar, właściciel, treść itp
- Wybór cech i ich wartości zależy od decyzji osoby implementującej ją w kodzie

Programowanie obiektowe

Główne zalety programowania obiektowego:

- Ponowne użycie kodu
- Łatwa lokalizacja błędu
- Lepsze odwzorowanie rzeczywistości w aplikacji
- Hermetyzacja danych
- Skalowalność kodu
- Łatwy podział prac
- Mechanizm dziedziczenia i polimorfizm
- Lepsze wykorzystanie pamięci w programie

Programowanie obiektowe

- Język C++ powstał jako rozwinięcie języka C poprzez dodanie do niego klas
- Klasa jest definicją formatu danych wraz z metodami z nią powiązanymi
- Klasa jest rozwinięciem znanego z języka C deklaratywnego typu danych jakim jest struktura

Programowanie obiektowe

- **Klasa** (class) – zbiór informacji na temat czym jest obiekt
- Jak jest tworzony i jakie ma cechy w programie (symulacji)
- Klasa jest pojęciem abstrakcyjnym
- **Obiekt** jest definiowany na podstawie **cech** zdefiniowanych w **klasie** po przypisaniu im konkretnych **wartości**

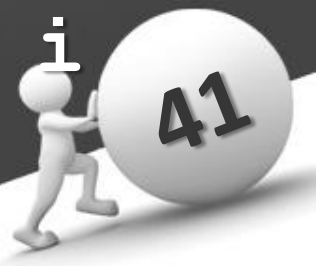
Programowanie obiektowe

- **Klasy** poza definicją **cech** mogą zawierać także **funkcje**
- Funkcja zdefiniowana w ramach klasy określana jest jako **metoda**, niekiedy **zachowanie**
- Na podstawie zdefiniowanej klasy można utworzyć wiele obiektów

Klasy

Najważniejsze właściwości klas:

1. Jest TYPEM zdefiniowanym przez użytkownika
2. Zawiera zestaw składowych – dane i funkcje
3. Funkcje składowe mogą – definiować sposób inicjacji (tworzenia), kopiowania, przenoszenia i usuwania obiektów
4. Dostęp do składowych uzyskuje się przy użyciu kropki (obiekty) i operatora -> (wskaźniki)



Klasy

Najważniejsze właściwości klas:

5. Można dla niej zdefiniować operatory +, ! i []
6. Klasa to przestrzeń nazw obejmująca swoje składowe
7. Składowe publiczne określają interfejs klasy, a prywatne jej implementacje

Klasy

- Klasa umożliwia zdefiniowanie w języku programowania obiektów rzeczywistych i abstrakcyjnych
- W ramach klasy można zdefiniować:
  - **cechy** - opisujące dany obiekt
  - **metody** – definiujące działania wykonywane przez dany obiekt

Klasy

- Dostęp do elementów projektowanej klasy jest ściśle określony
  - Publiczny – public:
  - Prywatny – private:
  - Chroniony – protected:
- W strukturze wszystkie elementy mają nadany domyślnie poziom **publiczny** - niemodyfikowalny
- W klasie wszystkie elementy domyślnie mają nadany poziom **prywatny**

Klasy

- W obszarze private deklaruje się:
  - Cechy będące danymi klasy
  - Metody wewnętrzne
- W obszarze public deklaruje się:
  - Interfejs klasy:
    - Cechy które mają być widoczne na zewnątrz
    - Metody interfejsu (wejścia, wyjścia, działania)

Klasy

- Poza cechami (komponentami) w klasie można zdefiniować metody (interfejs), czyli funkcje operujące na cechach (zmiennych) zdefiniowanych w klasie.
- W wielu sytuacjach najpierw określa się interfejs, czyli jakie działania ma wykonywać dana klasa, dopiero na podstawie takiej listy określa się komponenty (cechy) jakie są konieczne do wykonania przewidywanych zadań.

Przykład

```
1  #include <iostream>
2  #include <string>
```

C:\Windows\system32\cmd.exe

```
Podaj dane pracownika:
Imię (imiona) pracowniak: Adam Alfred
Nazwisko pracownika: Kowalski
Numer telefonu pracownika: 444555323
Zarobki: 8766.13
Pan/Pani Adam Alfred Kowalski
Nr. kontaktowy: +48444555323
Pensja: 8766.13 zł

Press any key to continue . . .
```

drukuj()

```
21          cout << zarobki << " zł" << endl;
22      }
23  };
```

Klasa

- Raz zdefiniowana klasa może być wielokrotnie używana w kodach różnych programów
- Kod klasy w osobnym pliku źródłowym i nagłówkowym
- Zapewnia to łatwość ponownego wykorzystania już napisanego kodu w wielu projektach
- Dla klasy lub grupy powiązanych klas deklaracje zapisuje się w pliku nagłówkowym (*.hpp) a kod metod w pliku źródłowym (*.cpp)

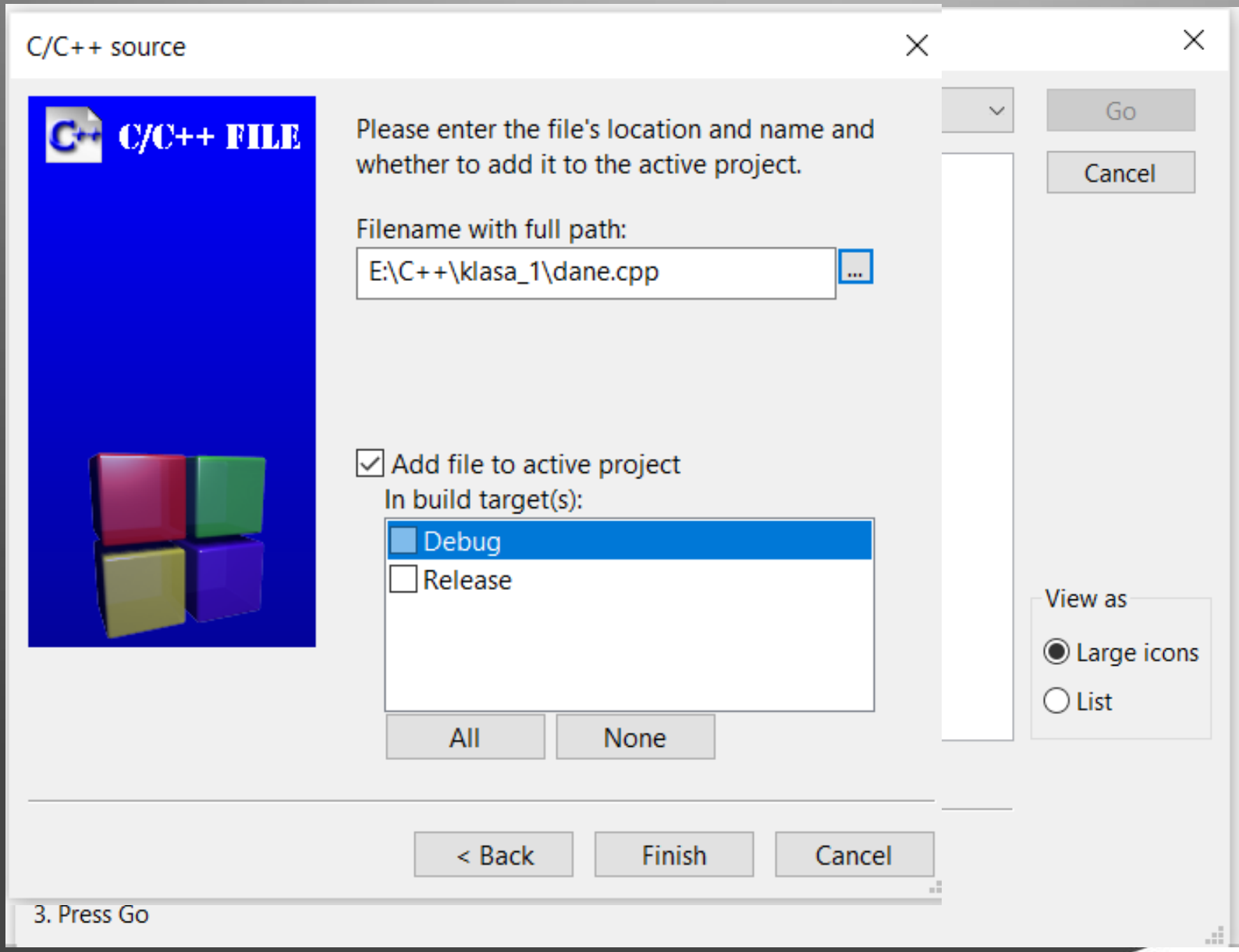
Przykład

```
1  #include "dane.hpp"
2
3  void dane::wczytaj()
4  {
5      cout << "Podaj dane pracownika:" << endl;
6      cout << "Imię (imiona) pracowniak: "; getline(cin,imie);
7      cout << "Nazwisko pracownika: "; getline(cin,nazwisko);
8      cout << "Numer telefonu pracownika: "; cin >> telefon;
9      cout << "Zarobki: "; cin >> zarobki;
10 }
11 void dane::drukuj()
12 {
13     cout << "Pan/Pani " << imie << " " << nazwisko << endl;
14     cout << "Nr. kontaktowy: +48" << telefon << endl;
15     cout << "Pensja: ";
16     cout.precision(2); cout.setf(ios_base::fixed,ios_base::floatfield);
17     cout << zarobki << " zł" << endl;
18 }
13  };
14  #endif
```

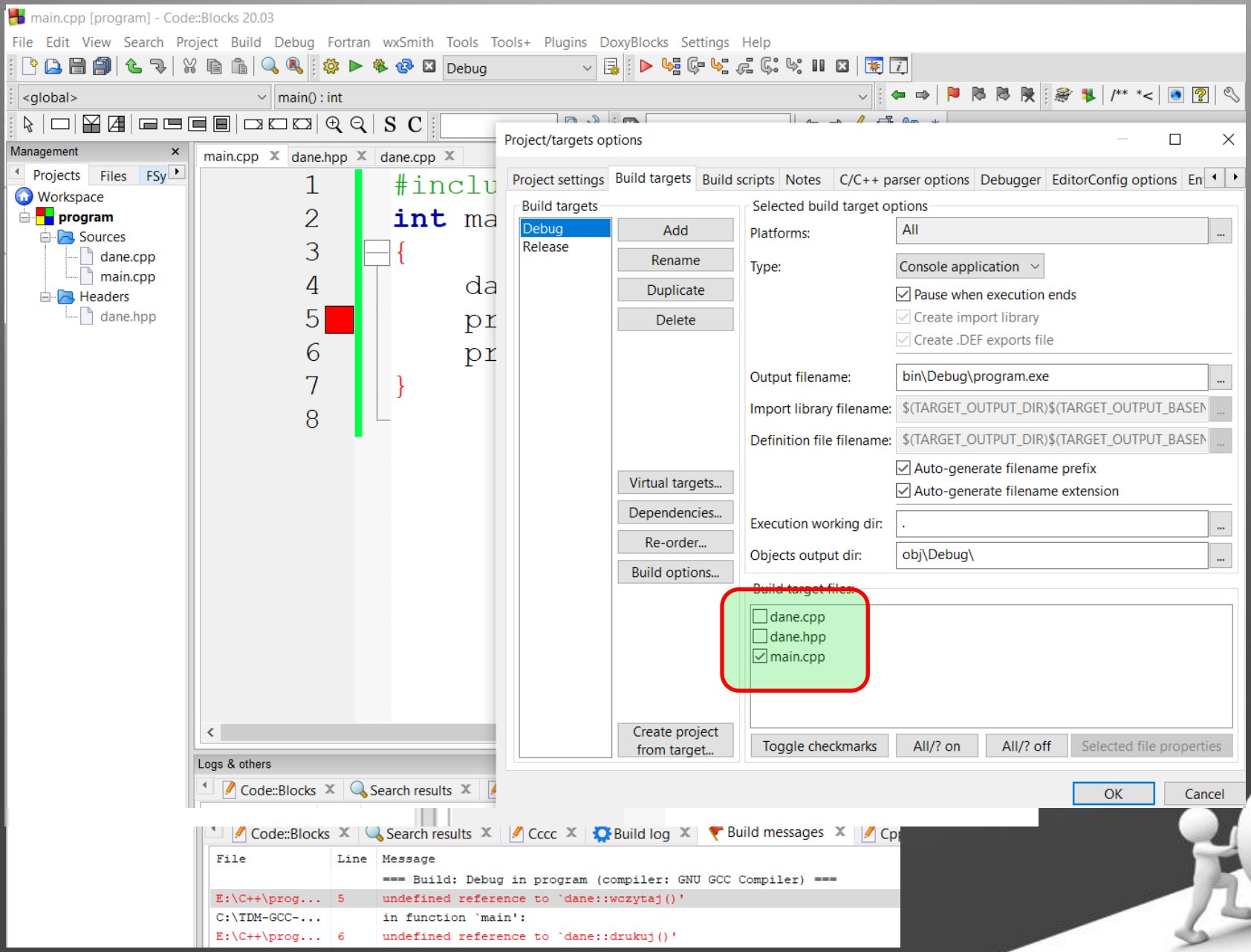
CODE::BLOCKS - klasa

- Dedykowane środowiska programistyczne wspierają (automatyzują) dodawanie plików klasy do projektu
  - Dodanie pliku nagłówkowego i źródłowego
  - Kreator dodawania klasy

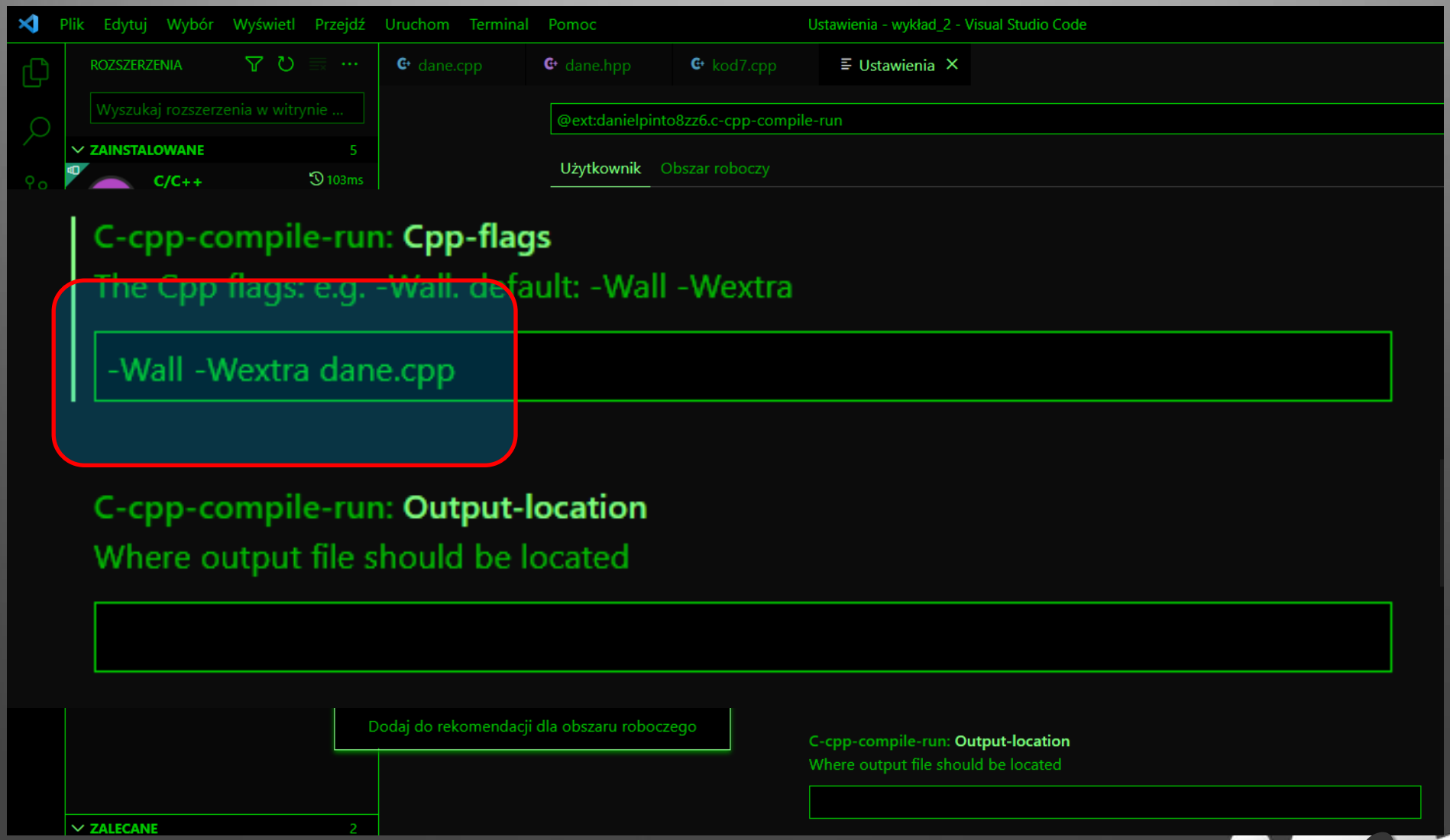
CODE::BLOCKS - klasa



CODE::BLOCKS - klasa



10:40 Visual Studio Code - klasa



CODE::BLOCKS - klasa

include\dane.hpp [program_2] - Code::Blocks 20.03

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings H

Debug

Management

Projects Files FSymb

- Workspace
- program_2
  - Sources
    - src
      - dane.cpp
      - main.cpp
  - Headers
    - include
      - dane.hpp

main.cpp | include\dane.hpp | src\dane.cpp

```
1   #ifndef DANE_HPP
2   #define DANE_HPP
3
4
5   class dane
6   {
7       public:
8           dane();
9
10      protected:
11
12      private:
13          string imie;
14          string nazwisko;
15          int telefon;
16          float zarobki;
17  };
18
19  #endif // DANE_HPP
20
```

Przykład

- Zaproponować klasę „portfel” pozwalająca na przechowywanie informacji o posiadanej kwocie
- Klasa umożliwia zarządzanie zasobami tylko za pomocą metod
- Interfejs zapewnia
  - Inicjację portfela
  - Wpłaty
  - Wypłaty
  - Informacje o stanie
  - Informacje statystyczne:
    - Liczba transakcji
    - Stosunek wpłat do wypłat (kwoty, transakcje)

Przykład

```
1  #ifndef PORTFEL_HPP
2  #define PORTFEL_HPP
3  #include <iostream>
4  using namespace std;
5
6  class portfel
7  {
8      float kwota;
9      unsigned int nWplata;
10     unsigned int nWyplata;
11     float kwWplata;
12     float kwWyplata;
13
14     public:
15         void inicjuj();
16         void wplata(float);
17         void wyplata(float);
18         void stan();
19         void statystyki();
20 };
21 #endif // PORTFEL_HPP
```

Przykład

```
#include "portfel.hpp"
void portfel::stan()
{
    cout << "Stan konta: " << kwota << " zl" << endl;
    system("pause>null");
}
void portfel::statystyki()
{
    cout << "Stan konta: " << kwota << " zl" << endl;
    cout << "Liczba wplat: " << nWplata;
    cout << ", kwota wplat: " << kwWplata << " zl" << endl;
    cout << "Liczba wyplat: " << nWyplata;
    cout << ", kwota wyplat: " << kwWyplata << " zl" << endl;
    cout.precision(3);
    cout << "Stosunek wplat/wyplat: " << kwWplata/kwWyplata << endl;
    cout << "Stosunek liczby transakcji: ";
    cout << nWyplata/((float)nWplata) << endl;
}
        kwWyplata+=suma;
    }
}
```

Przykład

```
        if(x==1){
            cin >> kwota;
            kasa.wplata(kwota);
        }
        else if(x==2){
            cin >> kwota;
            kasa.wyplata(kwota);
        }
        else
            kasa.stan();
    }
    system("cls");
    cout << "Koniec transakcji." << endl;
    cout << "Statystyki sesji:" << endl;
    kasa.statystyki();
    return 0;
}
```

Przykład

"G:\M˜j dysk\DYDAKTYKA\Informatyka II\Nowe Wyk| ady\Kody\wyk| ad_2\portfel\bin\Debug\portfel.exe"

```
Koniec transakcji.
Statystyki sesji:
Stan konta: 2563 zl
Liczba wplat: 5, kwota wplat: 10709 zl
Liczba wyplat: 4, kwota wyplat: 8146 zl
Stosunek wplat/wyplat: 1.31
Stosunek liczby transakcji: 0.8

Process returned 0 (0x0)   execution time : 152.947 s
Press any key to continue.

Mozna wyplacic do: 226 zl
```

Wskaźnik *this*

- W języku C++ dostępny jest specjalny wskaźnik this wskazujący na obiekt dla którego wywołana metodę
  - Metoda może zwrócić obiekt jako wynik działania
  - Odróżnienie pól klasy od zmiennych lokalnych metody – jeżeli nazywają się tak samo
  - Skasowanie obiektu - niebezpieczne

Przykład 1

```
14      void print()
15      {
16          cout << "Punkt (" << nazwa << ")->["
17               << x << "," << y << "]" << endl;
18      }
```

```
C:\Windows\system32\cmd.exe
Punkt (Lublin)->[10,10]

Press any key to continue . . .
```

```
24      A.print();
25  }
```

Przykład 2

- Napisać klasę wektor do przechowywania trójwymiarowych wektorów
- Klasa powinna posiadać trzy metody publiczne:
  - Wypisz – wyświetla na standardowym wyjściu wartość wektora
  - Wczytaj – umożliwia ustawienie wartości wektora
  - Dodaj – dodaje do wektora wektor podany na wejściu
- Wszystkie pola (cechy) zadeklarowane jako prywatne

Przykład 2

C:\Windows\system32\cmd.exe

```
Alfa(10,20,20)
Beta(-20,-10,30)
Alfa(-10,10,50)
Alfa(-10,10,50)

Press any key to continue . . .
```

```
 9        B.wypisz();
10        A.dodaj(B);
11        A.wypisz();
12        B=A;
13        B.wypisz();
14    }
19        z = z + A.z;
20    }
```