Podstawy Informatyki

Katedra Telekomunikacji, **EiT**

dr inż. Jarosław Bułat

kwant@agh.edu.pl

www.agh.edu.pl

Plan prezentacji

- Programowanie obiektowe
- Klasy
- Metody
- Modyfikator dostępu
- Konstruktor i destruktor
- Inicjalizacja składowych
- Projekt (make/cmake)

www.agh.edu.pl

AGH

Kod jest zbyt skomplikowany?

zacznij myśleć obiektowo

www.agh.edu.pl

Programowanie obiektowe

- To jeden z **paradygmatów programowania**
- Program zdefiniowany obiektami, które łączą:
  - stan (dane, pola)
  - zachowanie (akcje, procedury, metody)

- Ułatwia pisanie skomplikowanych programów
- Ułatwia rozwijanie, konserwowanie i testowanie
- Ułatwia wielokrotnie użycie programów lub ich części

- Podejście zgodne z rzeczywistością (bardziej rzeczywiste niż abstrakcyjne), podobne do sposobu przetwarzania informacji przez ludzki mózg

www.agh.edu.pl

Programowanie obiektowe

- Cechy paradygmatu obiektowego:
  - Abstrakcja
  - Hermetyzacja
  - Polimorfizm
  - Dziedziczenie
- Historia: Simula 1967, potem Smalltalk, potem “wszyscy”
- czysto obiektowe: Smalltalk, Python, Ruby, Scala, Eiffel
- głównie obiektowe: Java, C++, C#, Delphi, VB.NET
- ewolucja do obiektowych: MATLAB, COBOL, Fortran, Object Pascal

Programowanie obiektowe

- Nie ma jednego “modelu programowania obiektowego”
- Różne języki programowania implementują mniej lub więcej lub inaczej pewne aspekty programowania obiektowego
- Paradygmat obiektowy może być mieszany z proceduralnym (np. C++)

Klasa

- Rozwinięcie koncepcji struktur

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

class Circle {
   public:
      float radius;
      float area() {
         return 3.14*radius*radius;
      }
};

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

www.agh.edu.pl

Klasa

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

class Circle {
   public:
      float radius;
      float area() {
         return 3.14*radius*radius;
      }
};

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

- Rozwinięcie koncepcji struktur
- Definicja klasy - zbliżona do definicji struktury

www.agh.edu.pl

Klasa

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

class Circle {
   public:
      float radius;
      float area() {
         return 3.14*radius*radius;
      }
};

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

- Rozwinięcie koncepcji struktur
- Definicja klasy - zbliżona do definicji struktury
- Obiekt klasy - **instancja danej klasy**
- Może istnieć wiele instancji tej samej klasy == wiele zmiennych typu Circle

www.agh.edu.pl

Klasa

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

class Circle {
   public:
      float radius;
      float area() {
         return 3.14*radius*radius;
      }
};

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

- Rozwinięcie koncepcji struktur
- **Składowe** klasy (ang. members) to:

www.agh.edu.pl

Klasa

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

class Circle {
   public:
      float radius;
      float area() {
         return 3.14*radius*radius;
      }
};

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

- Rozwinięcie koncepcji struktur
- **Składowe** klasy (ang. members) to:
  - dane (zmienne)

www.agh.edu.pl

Klasa

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

class Circle {
   public:
      float radius;
      float area() {
         return 3.14*radius*radius;
      }
};

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

- Rozwinięcie koncepcji struktur
- **Składowe** klasy (ang. members) to:
  - dane (zmienne)
  - metody (funkcje)

www.agh.edu.pl

Klasa

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

class Circle {
   public:
      float radius;
      float area() {
         return 3.14*radius*radius;
      }
};

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

- Rozwinięcie koncepcji struktur
- **Składowe** klasy (ang. members) to:
  - dane (zmienne)
  - metody (funkcje)
- Odwołania do składowych tak jak w strukturach:
  - operator . jeżeli zmienna
  - operator -> jeżeli wskaźnik
- Metoda jest wywoływana “na” obiekcie (w kontekście obiektu)
- Metoda ma dostęp tylko do danych z obiektu na którym została wywołana

www.agh.edu.pl

Klasa

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

class Circle {
   public:
      float radius;
      float area() {
         return 3.14*radius*radius;
      }
};

int main() {
   Circle cir, okr;
   cir.radius = 1;
   okr.radius = 7;

   cout << cir.area() << endl;
   cout << okr.area() << endl;
}
```

- Rozwinięcie koncepcji struktur
- **Składowe** klasy (ang. members) to:
  - dane (zmienne)
  - metody (funkcje)
- Obiekt jest zmienną:
  - mogę utworzyć wiele zmiennych tego samego typu
  - mogę utworzyć wiele obiektów jednej klasy
  - każdy obiekt ma własne zmienne ale wspólny kod metod

www.agh.edu.pl

quiz

PI09_OO1

socrative.com

- login
- student login

Room name:

KWANTAGH

Klasa: deklaracja, definicja

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

class Circle {
   public:
      float radius;
      float area();
};

float Circle::area() {
   return 3.14*radius*radius;
}

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

www.agh.edu.pl

Klasa: deklaracja, definicja

» Deklaracja klasy

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

class Circle {
   public:
      float radius;
      float area();
};

float Circle::area() {
   return 3.14*radius*radius;
}

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

www.agh.edu.pl

Klasa: deklaracja, definicja

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

class Circle {
   public:
      float radius;
      float area();
};

float Circle::area() {
   return 3.14*radius*radius;
}

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

- Deklaracja klasy
- Definicja składowych klasy poza miejscem deklaracji

www.agh.edu.pl

Klasa: deklaracja, definicja

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

class Circle {
   public:
      float radius;
      float area();
};

float Circle::area() {
   return 3.14*radius*radius;
}

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

- Deklaracja klasy
- Definicja składowych klasy poza miejscem deklaracji
- Operator zasięgu :: pozwala definiować składowe poza deklaracją

www.agh.edu.pl

Klasa: deklaracja, definicja

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

class Circle {
   public:
      float radius;
      float area();
};

float Circle::area() {
   return 3.14*radius*radius;
}

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

- Deklaracja klasy
- Definicja składowych klasy poza miejscem deklaracji
- Operator zasięgu :: pozwala definiować składowe poza deklaracją według schematu:

www.agh.edu.pl

Klasa: deklaracja, definicja

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

class Circle {
   public:
      float radius;
      float area();
};

float Circle::area() {
   return 3.14*radius*radius;
}

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

- Deklaracja klasy
- Definicja składowych klasy poza miejscem deklaracji
- Operator zasięgu :: pozwala definiować składowe poza deklaracją według schematu:
  - nazwa klasy

www.agh.edu.pl

Klasa: deklaracja, definicja

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

class Circle {
   public:
      float radius;
      float area();
};

float Circle::area() {
   return 3.14*radius*radius;
}

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

- Deklaracja klasy
- Definicja składowych klasy poza miejscem deklaracji
- Operator zasięgu :: pozwala definiować składowe poza deklaracją według schematu:
  - nazwa klasy
  - nazwa składowej (metody)

www.agh.edu.pl

Klasa: deklaracja, definicja

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

class Circle {
   public:
      float radius;
      float area(void);
};

float Circle::area(void) {
   return 3.14*radius*radius;
}

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

- Deklaracja klasy
- Definicja składowych klasy poza miejscem deklaracji
- Operator zasięgu :: pozwala definiować składowe poza deklaracją według schematu:
  - nazwa klasy
  - nazwa składowej (metody)
  - typ

www.agh.edu.pl

Klasa: deklaracja, definicja

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

class Circle {
   public:
      float radius;
      float area(void);
};

float Circle::area(void) {
   return 3.14*radius*radius;
}

int main() {
   Circle cir;
   cir.radius = 1;

   cout << cir.area() << endl;
}
```

- Deklaracja klasy
- Definicja składowych klasy poza miejscem deklaracji
- Operator zasięgu :: pozwala definiować składowe poza deklaracją według schematu:
  - nazwa klasy
  - nazwa składowej (metody)
  - typ
- Podział na deklarację i definicję
  - zwiększa czytelność kodu
  - ułatwia tworzenie złożonych klas

www.agh.edu.pl

Klasa: deklaracja, definicja

```
// file: ex02_main.cc
// g++ ex02_circle.cc ex02_main.cc

#include <iostream>
#include "ex02_circle.h"
using namespace std;

int main() {
    Circle cir;
    cir.radius = 1;

    cout << cir.area() << endl;
}
```

```
// file: ex02_circle.h
class Circle {
    public:
        float radius;
        float area();
};
```

```
// file: ex02_circle.cc
#include "ex02_circle.h"

float Circle::area() {
    return 3.14*radius*radius;
}
```

www.agh.edu.pl

Klasa: deklaracja, definicja

```
// file: ex02_main.cc
// g++ ex02_circle.cc ex02_main.cc

#include <iostream>
#include "ex02_circle.h"
using namespace std;

int main() {
    Circle cir;
    cir.radius = 1;

    cout << cir.area() << endl;
}
```

```
// file: ex02_circle.h
class Circle {
    public:
        float radius;
        float area();
};
```

```
// file: ex02_circle.cc
#include "ex02_circle.h"

float Circle::area() {
    return 3.14*radius*radius;
}
```

- Deklaracja klasy w pliku *.h
- Definicja klasy w pliku *.cc (definicja składowych klasy)

www.agh.edu.pl

Klasa: deklaracja, definicja

```
// file: ex02_main.cc
// g++ ex02_circle.cc ex02_main.cc

#include <iostream>
#include "ex02_circle.h"
using namespace std;

int main() {
    Circle cir;
    cir.radius = 1;

    cout << cir.area() << endl;
}
```

```
// file: ex02_circle.h
class Circle {
    public:
        float radius;
        float area();
};
```

```
// file: ex02_circle.cc
#include "ex02_circle.h"

float Circle::area() {
    return 3.14*radius*radius;
}
```

- Deklaracja klasy w pliku *.h
- Definicja klasy w pliku *.cc (definicja składowych klasy)

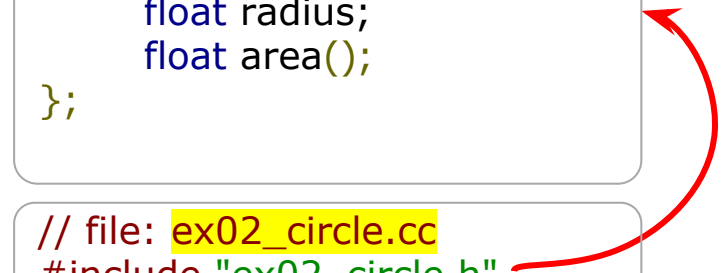
www.agh.edu.pl

Klasa: deklaracja, definicja

```
// file: ex02_main.cc
// g++ ex02_circle.cc ex02_main.cc

#include <iostream>
#include "ex02_circle.h"
using namespace std;

int main() {
    Circle cir;
    cir.radius = 1;

    cout << cir.area() << endl;
}
```

```
// file: ex02_circle.h
class Circle {
    public:
        float radius;
        float area();
};
```

```
// file: ex02_circle.cc
#include "ex02_circle.h"

float Circle::area() {
    return 3.14*radius*radius;
}
```



» Definicje składowych wymagają deklaracji klasy

www.agh.edu.pl

Klasa: deklaracja, definicja

```
// file: ex02_main.cc
// g++ ex02_circle.cc ex02_main.cc

#include <iostream>
#include "ex02_circle.h"
using namespace std;

int main() {
    Circle cir;
    cir.radius = 1;

    cout << cir.area() << endl;
}
```

```
// file: ex02_circle.h
class Circle {
    public:
        float radius;
        float area();
};
```

```
// file: ex02_circle.cc
#include "ex02_circle.h"

float Circle::area() {
    return 3.14*radius*radius;
}
```

- Definicje składowych wymagają deklaracji klasy
- Utworzenie obiektu wymaga deklaracji klasy

www.agh.edu.pl

Klasa: deklaracja, definicja

```
// file: ex02_main.cc
// g++ ex02_circle.cc ex02_main.cc

#include <iostream>
#include "ex02_circle.h"
using namespace std;

int main() {
    Circle cir;
    cir.radius = 1;

    cout << cir.area() << endl;
}
```

```
// file: ex02_circle.h
class Circle {
    public:
        float radius;
        float area();
};
```

```
// file: ex02_circle.cc
#include "ex02_circle.h"

float Circle::area() {
    return 3.14*radius*radius;
}
```

- Definicje składowych wymagają deklaracji klasy
- Utworzenie obiektu wymaga deklaracji klasy
- **Kompilacja**: g++ ex02_circle.cc ex02_main.cc -o ex02

quiz

PI09_OO2

socrative.com

- login
- student login

Room name:

KWANTAGH

Jak ukryć detale implementacji?

zadeklaruj je prywatnie

www.agh.edu.pl

Klasa: specyfikator dostępu

```
class Circle {
   private:
      float radius;

   protected:
      bool initialized;

   public:
      float area;
      float circuit() {
         return 2*3.14*radius;
      }

   private:
      float x, y;
};
```

- Specyfikatory dostępu
  - dotyczą zmiennych i metod
  - modyfikują zasięg
  - zmiana od specyfikatora do końca deklaracji lub innego specyfikatora
- private: (domyślny): tylko metody danej klasy
- public: metody + zasięg obiektu klasy
- protected: TBD
- Wszystkie pola struktur są “public”
- Indentation!!!

www.agh.edu.pl

Klasa: specyfikator dostępu

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

class Circle {
    private:
        float radius;
    public:
        float area() {
            return 3.14*radius*radius;
        }
};

int main() {
    Circle cir;
    cir.radius = 1;

    cout << cir.area() << endl;
}
```

- Składowa radius jest prywatna
- Można ją użyć tylko wewnątrz klasy
- Próba użycia na zewnątrz klasy kończy się błędem kompilacji:

```
ex03.cc: In function ‘int main()’:
ex03.cc:6:15: error: ‘float Circle::radius’ is
private
        float radius;
              ^
ex03.cc:15:9: error: within this context
    cir.radius = 1;
        ^
```

www.agh.edu.pl

Klasa: specyfikator dostępu

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

class Circle {
    private:
        float radius;
    public:
        float area() {
            return 3.14*radius*radius;
        }
};

int main() {
    Circle cir;
    cir.radius = 1;

    cout << cir.area() << endl;
}
```

- Składowa radius jest prywatna
- Można ją użyć tylko wewnątrz klasy
- Próba użycia na zewnątrz klasy kończy się błędem kompilacji:

```
ex03.cc: In function ‘int main()’:
ex03.cc:6:15: error: ‘float Circle::radius’ is
private
        float radius;
              ^
ex03.cc:15:9: error: within this context
     cir.radius = 1;
         ^
```

www.agh.edu.pl

użycie prywatnych składowych

```
class Circle {
  private:
    float radius_;
  public:
    float getRadius(void) {
      return radius_;
    }
    void setRadius(float radius) {
      radius_ = radius;
    }
};

int main() {
  Circle cir;
  cir.setRadius(0.001);

  cout << cir.getRadius() << endl;
}
```

- Publiczne metody pozwalające uzyskać dostęp do prywatnych składowych
- Kontrola dostępu, ochrona
  - tylko odczyt
  - tylko zapis
  - zapis+odczyt
- Kontrola poprawności: weryfikacja danych podczas zapisu
- Kontrolowana konwersja danych
- Powszechna technika w innych językach (np. Java)

www.agh.edu.pl

użycie prywatnych składowych

```
class Circle {
    private:
        float radius_;
    public:
        float getRadius(void) {
            return radius_;
        }
        void setRadius(float radius) {
            radius_ = radius;
        }
};

int main() {
    Circle cir;
    cir.setRadius(0.001);

    cout << cir.getRadius() << endl;
}
```

- getter, accessor
- setter, mutator
- Wsparcie w IDE (automatyzacja, refactoring)
- Wady: więcej kodu

użycie prywatnych składowych

```
class Circle {
    private:
        float radius_;
    public:
        float getRadius(void) {
            return radius_;
        }
        void setRadius(float radius) {
            radius_ = radius;
        }
};

int main() {
    Circle cir;
    cir.setRadius(0.001);

    cout << cir.getRadius() << endl;
}
```

- zmienna radius_ to częsta konwencja dla składowych zmiennych (danych)

www.agh.edu.pl

użycie prywatnych składowych

```
class Circle {
    private:
        float radius_;
    public:
        float getRadius(void) {
            return radius_;
        }
        void setRadius(float radius) {
            radius_ = radius;
        }
};

int main() {
    Circle cir;
    cir.setRadius(0.001);

    cout << cir.getRadius() << endl;
}
```

- zmienna radius_ to częsta konwencja dla składowych zmiennych (danych)
  - odróżnienie składowych od zmiennych lokalnych w metodach
  - brak konfliktu nazw w metodach*
  - nie trzeba “kombinować” z nazwami w setterze

www.agh.edu.pl

quiz

PI09_OO3

socrative.com

- login
- student login

Room name:

KWANTAGH

AGH

inicjalizacja obiektu, sprzątanie

konstruktor / destruktor

www.agh.edu.pl

Konstruktor

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

class Circle {
    private:
        float radius_;
    public:
        Circle() {
            radius_ = 0;
        }
        float getRadius(void) {
            return radius_;
        }
};

int main() {
    Circle cir;
    cout << cir.getRadius() << endl;
}
```

- Konstruktor to metoda o nazwie takiej jak nazwa klasy
- Wywoływany jest automatycznie podczas tworzenia obiekt
- musi być publiczny
- Służy do inicjalizacji obiektu (np. składowych)

- Bez konstruktora, można było wywołać getRadius() i uzyskać wartość niezainicjalizowanej zmiennej

www.agh.edu.pl

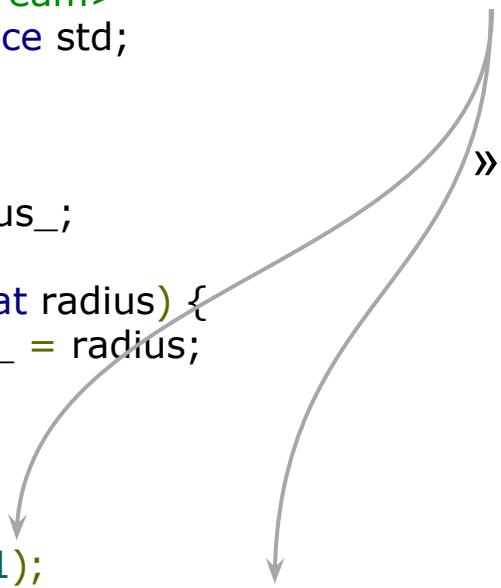
Konstruktor

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

class Circle {
    private:
        float radius_;
    public:
        Circle(float radius) {
            radius_ = radius;
        }
};

int main() {
    Circle cir(0.1);
    Circle *cir2 = new Circle(1);

    delete cir2;
}
```

- Konstruktor może mieć argumenty
- Typowy sposób inicjalizacji składowych klasy
- Podczas tworzenia obiektu można przekazać wartości
- Obiekt można tworzyć dynamicznie
- Zasady takie jak przy innych zmiennych (trzeba usunąć)
- Podczas tworzenia można przekazać wartości do konstruktora

Konstruktor

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

class Circle {
    private:
        float radius_;
    public:
        Circle(float radius) {
            radius_ = radius;
        }
};

int main() {
    Circle cir(0.1);
    Circle *cir2 = new Circle(1);
    Circle cir3;

    delete cir2;
}
```



- Podczas tworzenia obu obiektów klasy Circle uruchamiany jest konstruktor
- Próba deklaracji

  Circle cir3;

  woła konstruktor:

  Circle();

  którego nie ma więc błąd:

  error: no matching function for call to 'Circle::Circle()'
  Circle cir3;

www.agh.edu.pl

Konstruktor

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

class Circle {
    private:
        float radius_;
    public:
        Circle(float radius);
};

Circle::Circle(float radius) {
    radius_ = radius;
}
```

- Definicja konstruktora poza klasą
- Konstruktor nie zwraca typu, nie może być też void
  Definicja jako void spowoduje błąd:
  ex07.cc:11:33: error: return type specification for constructor invalid
  void Circle::Circle(float radius)
- Żaden konstruktor nie może zwrócić typu

www.agh.edu.pl

Destruktor

```
class Table {
    private:
        float *table_;
    public:
        Table(size_t size) {
            table_ = new float[size];
            cout << "1\n";
        }
        ~Table() {
            delete [] table_;
            cout << "3\n";
        }

};

int main() {
    Table tab(10);
    cout << "2\n";
}
```

- Destruktor to metoda o nazwie takiej jak nazwa klasy z “tyldą”
- Wywoływany automatycznie bezpośrednio przed zniszczeniem obiektu klasy
- Służy do “sprzątania” obiektu przed jego usunięciem:
  - zwolnienie pamięci zaalokowanej w konstruktorze
  - zwolnienie zasobów (pliki, sockety, sterowniki)
  - poinformowanie innych

www.agh.edu.pl

Destruktor

```
class Table {
    private:
        float *table_;
    public:
        Table(size_t size) {
            table_ = new float[size];
            cout << "1\n";
        }
        ~Table();
};

Table::~Table() {
    delete [] table_;
    cout << "3\n";
}
```

- Definicja destruktora poza klasą
- Destruktor nie zwraca typu
- Destruktor nie może mieć argumentów
- Destruktor musi być zadeklarowany w klasie żeby można go było zdefiniować poza klasą

quiz

PI09_OO4

socrative.com

- login
- student login

Room name:

KWANTAGH

inicjalizacja stanu obiektu, inicjalizacja składowych

www.agh.edu.pl

Inicjalizacja

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

class Complex {
    private:
        float re_, im_;
    public:
        Complex(float, float);
};

Complex::Complex(float re, float im) {
    re_ = re;
    im_ = im;
}

int main() {
    Complex x(6, 7);
}
```

- Inicjalizacja dwóch składowych prywatnych
- W deklaracji można nie podać nazw zmiennych
- W definicji przepisanie argumentów do składowych

www.agh.edu.pl

Inicjalizacja

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

class Complex {
    private:
        float re_, im_;
    public:
        Complex(float, float);
};

Complex::Complex(float re, float im)
    : re_(re),
      im_(im) {
}

int main() {
    Complex x(6, 7);
}
```

- Zalecany sposób inicjalizacji składowych
- W przypadku prostej inicjalizacji
- Przypisanie może być wyrażeniem (np. im+1)
- Skutek jest taki sam jak w poprzednim przykładzie
- Konstruktor jest pusty

www.agh.edu.pl

Inicjalizacja

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

class Complex {
   private:
      float *var_;
   public:
      Complex(float, float);
};

Complex::Complex(float re, float im)
   : var_(new float[2]) {
      var_[0] = re;
      var_[1] = im;
}

int main() {
   Complex x(6, 7);
}
```

- Pomysł: będę trzymał liczbę zespoloną w dwuelementowej tablicy
  - najpierw inicjalizacja tablicy (dynamicznie)
  - przypisanie elementów
- Jaki błąd popełniłem w implementacji klasy?

www.agh.edu.pl

Inicjalizacja zmiennych **const**

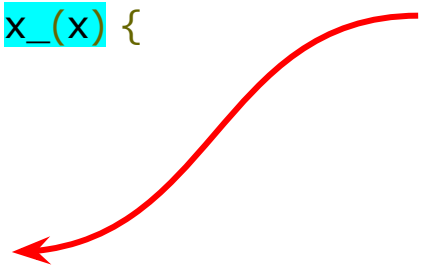
```
#include <iostream>
using namespace std;

class Foo {
    public:
        const int x_=7;
        int y_;
        Foo(int x);
};

Foo::Foo(int x) : x_(x) {
}

int main() {
    Foo obj(10);
    // obj.x_ = 0;
    obj.y_ = 0;
}
```

- Inicjalizacja zmiennych “const”
  - możliwa podczas deklaracji
  - możliwa w liście inicjalizacyjnej
- Każdy obiekt może mieć inaczej zainicjalizowaną składową const

www.agh.edu.pl

Inicjalizacja zmiennych **const**

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

class Foo {
    public:
        const int x_=7;
        int y_;
        Foo(int x);
};

Foo::Foo(int x) : x_(x) {
}

int main() {
    Foo obj(10);
    // obj.x_ = 0;
    obj.y_ = 0;
}
```



- Inicjalizacja zmiennych “const”
  - możliwa podczas deklaracji
  - możliwa w liście inicjalizacyjnej
- Każdy obiekt może mieć inaczej zainicjalizowaną składową const
- Próba zmiany zmiennej “const” spowoduje błąd:

```
ex14.cc:20:14: error: assignment of
read-only member ‘Foo::x_’
    obj.x_ = 0;
```

www.agh.edu.pl

składowe **static**

```
class Circle {
    public:
        static size_t count_;

        Circle(){count_++;}
        void printNoCircles(void){
            cout << count_ << endl;
        }
};

size_t Circle::count_ = 0;

int main() {
    Circle c0;
    c0.printNoCircles();

    Circle c1;
    c1.printNoCircles();
    c0.printNoCircles();
}
```

- Składowa typu static jest dzielona przez wszystkie obiekty tej klasy
- Musi być zdefiniowana poza klasą
- Stosowana wtedy gdy potrzebna jest wiedza o liczbie utworzonych obiektów
- Rezultat:
  1
  2
  2

www.agh.edu.pl

składowe **static**

```
class Circle {
    public:
        static size_t count_;

        Circle(){count_++;}
        static void printNoCircles(void){
            cout << count_ << endl;
        }
};

size_t Circle::count_ = 0;

int main() {
    Circle::printNoCircles();
    Circle c0;
    c0.printNoCircles();
}
```

- Metoda również może być typu static.
- Można ją wywołać nawet gdy żaden obiekt klasy nie jest utworzony.
- Rezultat:
  0
  1

www.agh.edu.pl

składowe **static**

```
class Circle {
    public:
        static size_t count_;

        Circle(){count_++;}
        static void printNoCircles(void){
            cout << count_ << endl;
        }
};

size_t Circle::count_ = 0;

int main() {
    Circle::printNoCircles();
    Circle c0;
    c0.printNoCircles();
}
```

- Metoda również może być typu static.
- Można ją wywołać nawet gdy żaden obiekt klasy nie jest utworzony.
- Rezultat:
  0
  1
- Metodę statyczną można wywołać na dwa sposoby

www.agh.edu.pl

quiz

PI09_OO5

socrative.com

- login
- student login

Room name:

KWANTAGH

AGH

jak skompilować 1000 plików?

utwórz projekt, użyj (c)make

www.agh.edu.pl

Projekt

- Prosty program: main.cc, circle.h, circle.cc
- Kompilacja: g++ main.cc, circle.cc -o main
- Nietrywialny projekt to:
  - setki/tysiące plików
  - różne docelowe platformy: ARM/x86, Win/Lin/Android
  - program pisany przez wielu deweloperów na różnych platformach: kompilacja musi się udać na Win i macOS
  - różne konfiguracje środowiska: pliki nagłówkowe i biblioteki mogą być w różnych miejscach na dysku
- Kompilator przetwarza kod źródłowy do wynikowego
- Kompilator NIE zarządza projektem

Projekt - zarządzanie

- Zależy od języka programowania (poniżej C++)
- IDE (Eclipse, CLion, VS, VSC, …) posiadają swoje narzędzia do zarządzania projektem (wzajemnie niekompatybilne!!!):
  - dodawanie/usuwanie plików do kompilacji
  - wyszukiwanie ścieżek dostępu nagłówków i bibliotek
  - uruchamianie kompilacji (multithread)
- IDE nie ma własnego kompilatora
- IDE często nie ma własnego “systemu budowania” (ang. buildsystem)
- IDE często produkuje plik tekstowy opisujący jak zbudować projekt za pomocą zewnętrznych narzędzi

Projekt - make

- **make** jest systemem budowania projektów (buildsystem)
  - wykonuje serię wywołań kompilatora
  - polecenia dla instrukcji make zapisujemy w pliku makefile

```
CPP = g++                                   # przykładowa zawartość pliku makefile
RANLIB = ar rcs
RELEASE = -c -O3                            # ← ustawienia kompilatora dla kodu release
DEBUG = -c -g -D_DEBUG                      # ← ustawienia kompilatora dla kodu debug
INCDIR = -I./stuff/include                  # ← katalogi z plikami nagłówkowymi
LIBDIR = -L./stuff/lib -L.                  # ← katalogi z bibliotekami
LIBS = -lstuff -lmystatlib -lmydynlib       # ← jakich bibliotek będzie używać program
CFLAGS = $(RELEASE)

PROGOBJS = prog1.o prog2.o prog3.o          # ← jakie pliki obiektowe będą tworzyć program

prog: main.o $(PROGOBJS) mystatlib mydynlib
    $(CC) main.o $(PROGOBJS) $(LIBDIR) $(LIBS) -o prog
debug: CFLAGS=$(DEBUG)
debug: prog
```

www.agh.edu.pl

Projekt - CMake

- **make** jest zależny od platformy (np. ścieżki dostępu), trudno jest napisać makefile

Projekt - CMake

- **make** jest zależny od platformy (np. ścieżki dostępu), trudno jest napisać makefile

- Co robi informatyk, jeżeli musi pisać makefile czyli “**nudny**” kod???

Projekt - CMake

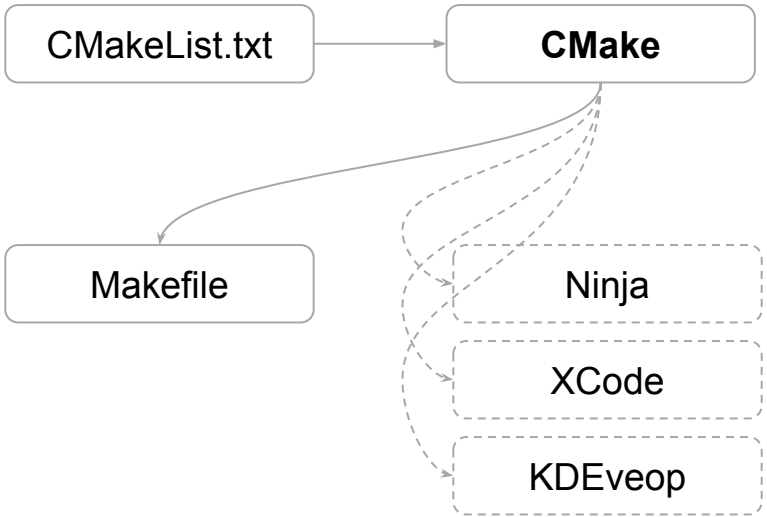
- **make** jest zależny od platformy (np. ścieżki dostępu), trudno jest napisać makefile
- Co robi informatyk, jeżeli musi pisać makefile czyli “**nudny**” kod???
- Tworzy metaprogram czyli program, który będzie pisał kod innego programu …

Projekt - CMake

- **make** jest zależny od platformy (np. ścieżki dostępu), trudno jest napisać makefile
- Co robi informatyk, jeżeli musi pisać makefile czyli “**nudny**” kod???
- Tworzy metaprogram czyli program, który będzie pisał kod innego programu …czyli:

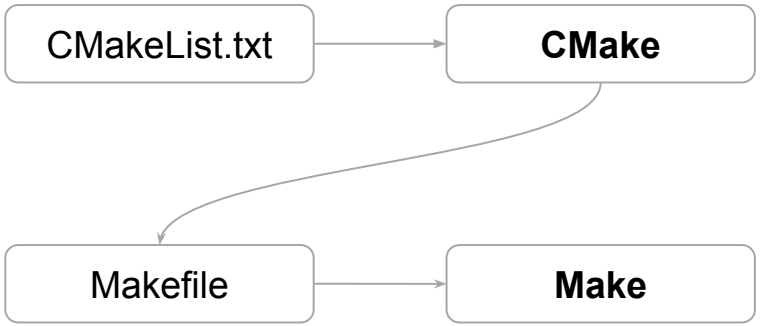
CMake: Cross-platform Make

CMake na podstawie pliku tekstowego CMakeList.txt tworzy tzw. buildsystem

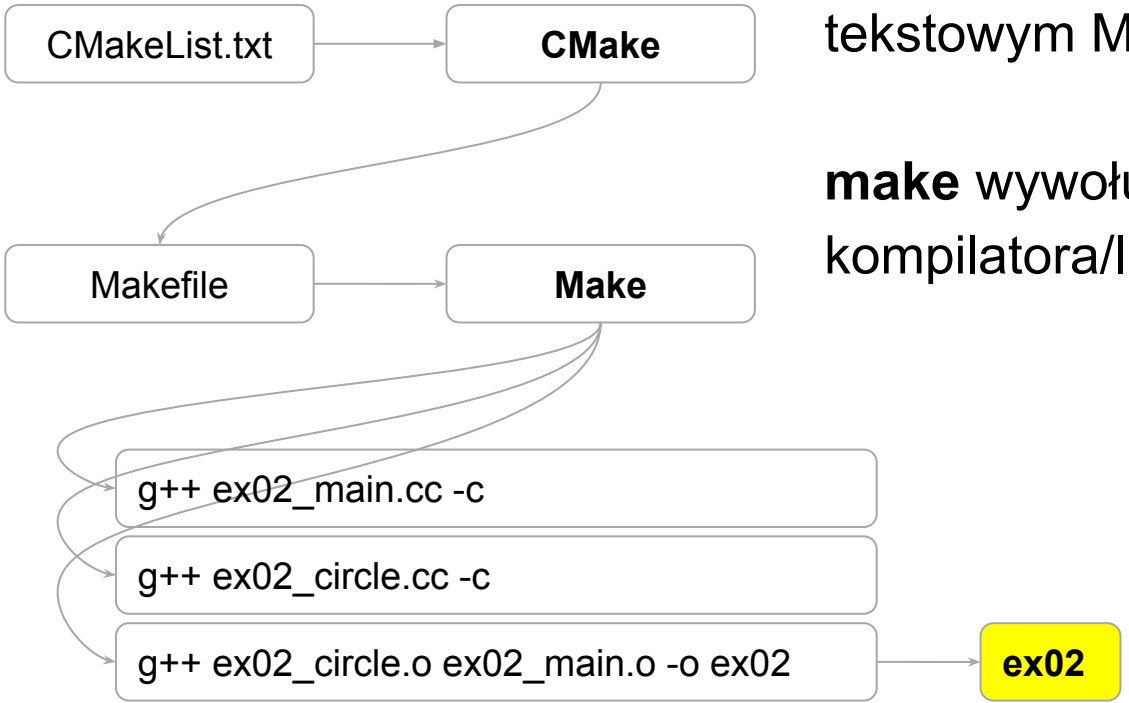


CMake na podstawie pliku tekstowego CMakeList.txt tworzy tzw. buildsystem np:

- Makefile (dla make)
- Ninja (alt. dla make)
- projekt XCode (Apple)
- projekt KDEvelop
- Visual Studio solutions



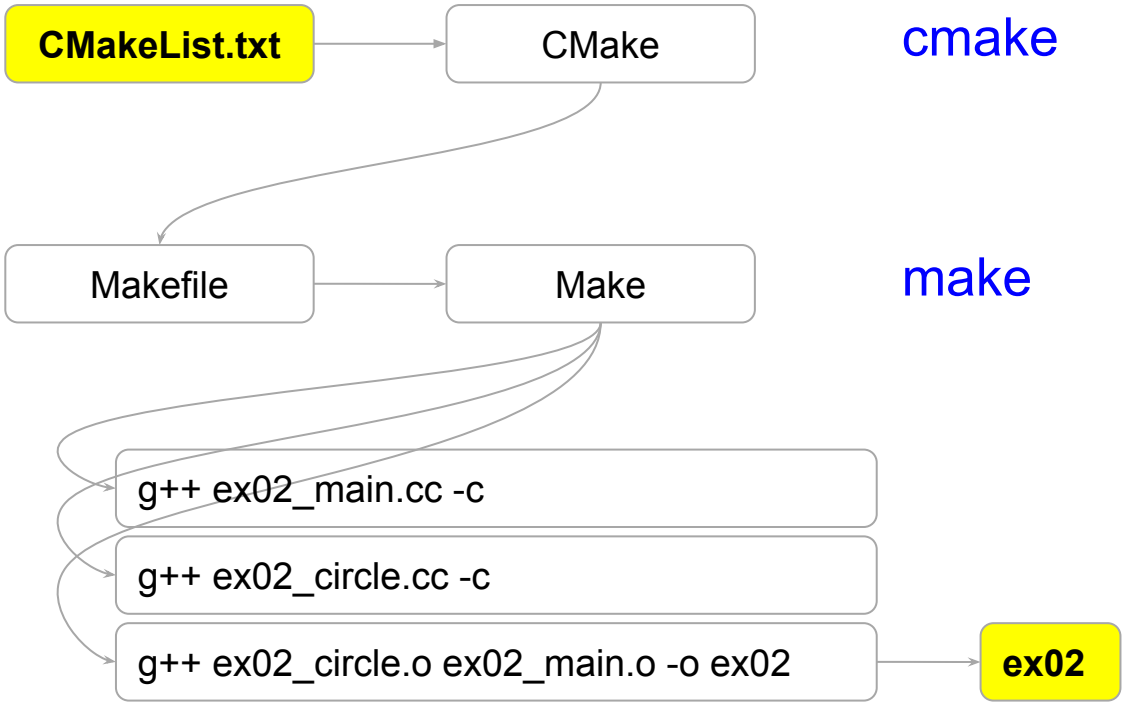
make to “buildsystem”, komendy są zapisane w pliku tekstowym Makefile



make to “buildsystem”, komendy są zapisane w pliku tekstowym Makefile

make wywołuje komendy kompilatora/linkera

polecenia (CLI)



www.agh.edu.pl

Projekt - CMake

- **CMake**
  - jest używany w bardzo wielu projektach
  - to nieformalny standard
  - jako element systemu budowania ma taką pozycję jak git w dziedzinie kontroli wersji

www.agh.edu.pl

AGH

www.agh.edu.pl

float x = 1;

czy to jest ok?

problemy z float

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

int main() {
   float x0 = 1;      // conversion int -> float
   float x1 = 1.0;    // conversion double -> float
   float x2 = 1.0f;   // ok, no conversion
   float x3 = 1f;     // syntax error !!!
}
```

- 1 typ: int
- 1.0 typ: double
- 1.0f typ: float

- W tym przypadku, konwersja w czasie kompilacji - nie boli, dla porządku powinno używać się 1.0f

www.agh.edu.pl

problemy z float

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

float foo(float x) {
   // return 2*x;     // conversion float(2)
   // return 2.0*x;   // conversion float(2.0)
   return 2.0f*x;     // ok, no conversion
}

int main() {
   float x0 = foo(1.2f);
}
```

- Pierwsze dwa sposoby powodują konwersję w run time !!!
- Wszystkie 3 sposoby zwrócą ten sam wynik
- Trzeci sposób będzie najszybszy

- Dyskusja: https://stackoverflow.com/questions/25229832/is-float-a-3-0-a-correct-statement

problemy z float

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

int main() {
   float x=2.4;

   if (2.4 == x) {
      cout << "never print\n" << endl;
   }

   cout << "first try: " <<  x << endl;
   cout << std::setprecision(9);
   cout << "second try: " << x << endl;
}
```

- Porównywanie dwóch zmiennych rzeczywistych bez uwzględnienia tolerancji to zawsze jest zły pomysł
- float(2.4) to w praktyce jest 2.400000953674 dlatego warunek nie zadziała
- if (2.4f == x) będzie ok
- Jeżeli porównujemy różne typy to lepiej: if (abs(2.4-x) <0.0001)

www.agh.edu.pl

Dziękuję