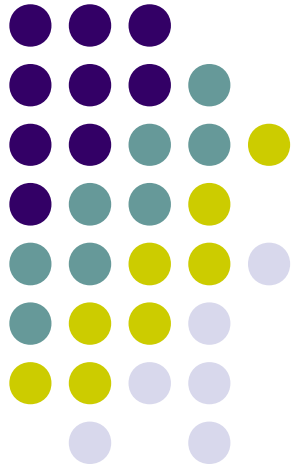
Średniozaawansowane programowanie w C++

„Szpaki” 2016/2017

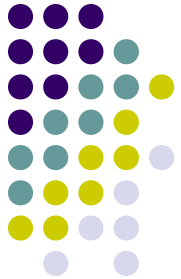
Czwartek

Wykład: 16:15–17:00

Konsultacje/projekt: 17:00–18:30



Harmonogram zajęć



1. Kontenery i algorytmy STL, sprytne wskaźniki, bindowanie funkcji
2. Wyjątki, stringi, wyrażenia regularne
3. Wzorce projektowe
4. Wielowątkowość
5. Grafika 2D
6. Qt (1)
7. Qt (2)
8. Programowanie generyczne, traity
9. Tworzenie dokumentacji, praca w zespołach projektowych

* harmonogram zajęć może ulec zmianie

Zasady zaliczania

7 x program (500 zł)
1 x projekt końcowy (1500 zł)

W sumie do „zarobienia” 5000 zł

Uwaga!

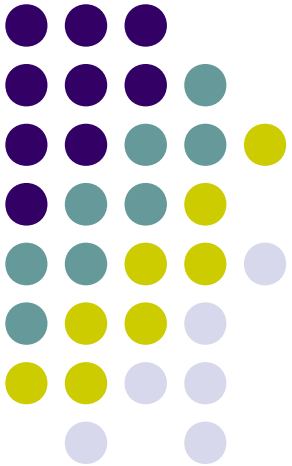
1. Oddanie programu tydzień po terminie: –100 zł
2. Program, którego nie można zbudować może być wyceniony maksymalnie na 100 zł

Zalecana literatura

1. Bjarne Stroustrup: *Programowanie. Teoria i praktyka z wykorzystaniem C++*. Helion, 2010.
2. Bjarne Stroustrup: *Język C++*. WNT.
3. Robert Nowak, Andrzej Pająk: *Język C++. Mechanizmy, wzorce, biblioteki*. BTC, 2010.
4. H. Sutter, A. Alexandrescu: *Język C++. Standardy kodowania*. Helion, 2005.
5. E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Wzorce projektowe*. WNT, 2005.
6. B. Karlsson: *Więcej niż C++. Wprowadzenie do bibliotek Boost*. Helion, 2006.

Siedem „grzechów” głównych

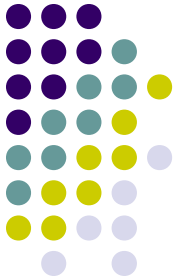
które popełniają młodzi adepci programowania w C/C++



1. Wyrzucanie rezultatów do „kosza”

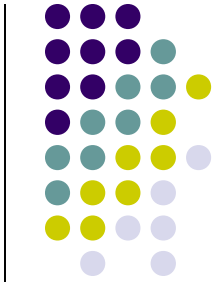
```
double rob_trudne_obliczenia (const double *tab)
{
        double cos_trudnego = 0.0;
        for (int x = 0; x < N; ++x)
        {
                // robi bardzo trudne obliczenia
        }
        return cos_trudnego;
}


int main ()
{
        double cos_trudnego;
        double nasza_tablica [333];
        rob_trudne_obliczenia (nasza_tablica);  // ŹLE
        cos_trudnego = rob_trudne_obliczenia (nasza_tablica);   // DOBRZE
        std::cout << cos_trudnego << std::endl;
        return 0;
}
```

2. Błędne pisanie nagłówków funkcji i metod



```
class Jablko
{
        public:
                void wyjmijPestki (int n);
                friend void wyjmij_pestki_z_jablka (Jablko &j);
};

void wyjmijPestki (int n)       // ŹLE: to powinna być metoda!
{
        // wyjmuje pestki
}

void Jablko::wyjmijPestki (double n)    // ŹLE: skąd to double?!
{
        // wyjmuje pestki
}

void wyjmij_pestki_z_jablka (Jablko &j) // DOBRZE: funkcja zaprzyjaźniona
{
        // wyjmuje pestki z j
}
```

3. Przykrywanie parametrów i składowych klas

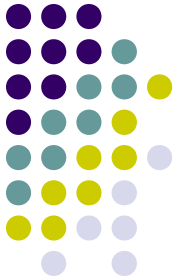


```
class Kaczuszka
{
        private:
                int liczba_nozek;
                int wiek;
                int liczba_potomstwa;
        public:
                void dodajPotomstwo (int n);
                void postarz ();
};

void Kaczuszka::dodajPotomstwo (int n)
{
        int n;  // ŹLE: przykrycie parametru
        wiek += n;
}

void Kaczuszka::postarz ()
{
        int wiek;       // ŹLE: przykrycie składowej klasy
        wiek++;
}
```

4. Permanentne nieuctwo stałych „chwytów” i popularnych algorytmów

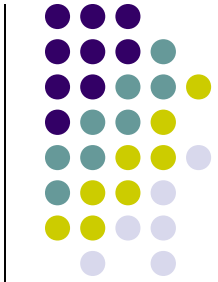
```
int main (int argc, char *arg[])        // obsługa parametrów programu
{
        if (argc < 2)
        {
                std::cerr << "Za mało parametrów!" << std::endl;
                return (1);
        }

        std::vector<double> tab;
        double srednia = 0.0;
        // (…) wczytanie do tablicy

        for (std::vector<double>::const_iterator it = tab.begin();
                it != tab.end(); ++it)
                srednia += *it;

        srednia/= tab.size ();
        std::cout << "Srednia liczb wynosi:" << srednia << std::endl;
        return 0;
}
```

5. Nieumiejętne używanie słówka *const*



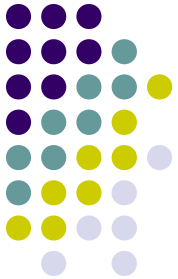
```
const int LICZBA_JEGO_IMIENIA = 44;

class PrzechowalniaBagazu
{
        private:
                std::vector<Bagaz> bagaze_;
        public:
                PrzechowalniaBagazu (const PrzechowalniaBagazu &pb);
                int liczBagaz () const;
                double masaCalkowita () const;
};

int PrzechowalniaBagazu::liczBagaz () const
{
        return bagaze_.size ();
}

double PrzechowalniaBagazu::masaCalkowita () const
{
        double masa = 0.0;
        For (std::vector<Bagaz>::const_iterator; (…))
                        masa += it->masa ();
        return masa;
}
```

6. Mylenie operatorów porównania i przypisania



```
int licznosc_druzyny;
// (…)

if (licznosc_druzyny = 11)      // ŹLE: przypisanie zamiast porównania
        graj_w_pilke ();
if (licznosc_druzyny == 11)     // DOBRZE: porównanie!
        graj_w_pilke ();


for (int i = 0; i < 100; ++i)
{
        suma = suma + tab[i];   // oldschoolowo
        suma += tab[i];         // nowocześnie :)
}

long silnia (int n)
{
        long wynik = 1;
        for (int i = 2; i <= n; ++i)
                wynik *= i;     // nowocześnie :)
        return wynik;
};
```

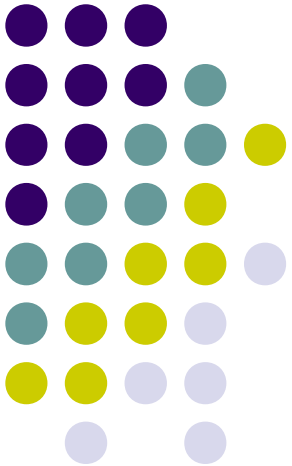
7. Tablica obiektów czy tablica w obiekcie?



```
class Billing
{
        Private:
                std::vector <Rozmowa> polaczenia_;
        Public:
                void dodajPolaczenie (const Rozmowa &r)
                {
                        polaczenia_.push_back (r);
                }
                // reszta interfejsu
};

int main ()
{
        std::vector <Jablon> sad;
        Jablon antonowka;
        sad.push_back (antonowka);      // sadzimy jabłko w sadzie
}
```

Styl kodowania

JEDNOLITOŚĆ i HARMONIJNOŚĆ



Styl kodowania i nazewnictwa

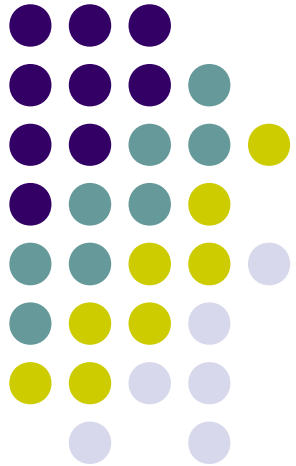
```
class MojaKlasa
{
        public:
                MojaKlasa ();
                void funkcjaPierwsza (int parametr_pierwszy);

                enum Kolor {BLACK, DARK_GREEN, BLUE, WHITE};


        private:
                Kolor kolor_;
};
```

Budowanie programu z wielu jednostek kompilacji

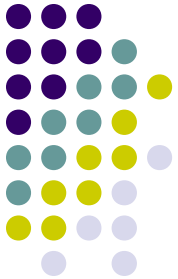
*.hpp

*.cpp

Makefile

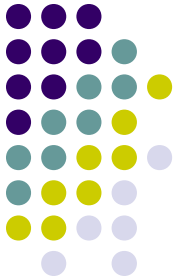


Pliki nagłówkowe *.h / *.hpp



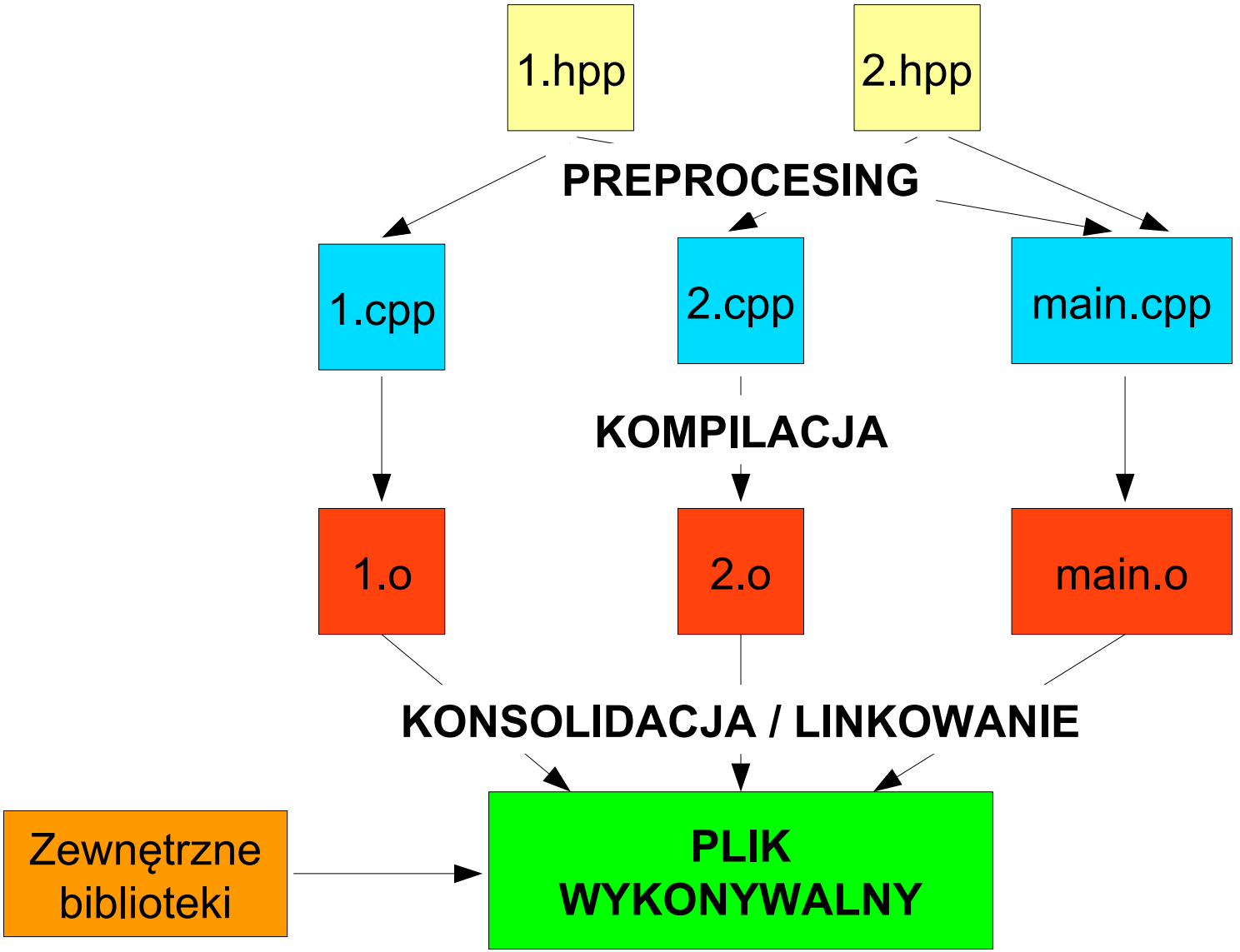
```
// oporniki.h

#ifndef _oporniki_h_
#define _oporniki_h_

#include <vector>
#include <complex>

typedef bool typ_polaczenia;
extern const typ_polaczenia SZEREGOWE;
extern const typ_polaczenia ROWNOLEGLE;

class Impedancja
{
        public:
                virtual std::complex<double> operator() (double omega) const = 0;
};

class Opornik : public Impedancja
{
        public:
                Opornik (double r = 0.0);
                virtual std::complex<double> operator() (double omega) const;
        private:
                double r_;
};

// definicje innych klas

#endif
```

Pliki źródłowe *.cpp

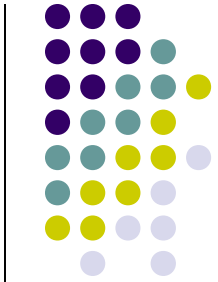


```
// oporniki.cpp

#include "oporniki.h"

const typ_polaczenia SZEREGOWE = false;
const typ_polaczenia ROWNOLEGLE = true;

Opornik::Opornik (double r)
{
        r_ = r;
}

std::complex<double> Opornik::operator() (double omega) const
{
        return r_;
}
```

Struktura programu



Makefile



```
.SILENT:        # nie „wypluwa” komend na ekran

CXX = g++
CXXFLAGS = -Wall -pedantic -O2
LIBRARIES = `allegro-config -libs` -lboost_thread


zad:    1.o 2.o main.o
                echo Buduję program...
                $(CXX) -o zad 1.o 2.o main.o $(CXXFLAGS) $(LIBRARIES)
1.o:    1.cpp 1.hpp
                echo Kompiluję plik 1.cpp...
                $(CXX) -o 1.o -c 1.cpp $(CXXFLAGS)
2.o:    2.cpp 2.hpp
                echo Kompiluję plik 2.cpp...
                $(CXX) -o 2.o -c 2.cpp $(CXXFLAGS)
main.o: main.cpp 1.hpp 2.hpp
                echo Kompiluję plik main.cpp...
                $(CXX) -o main.o -c main.cpp $(CXXFLAGS)
clean:
                rm -f *.o
```

Biblioteki rozszerzające

1. **Allegro** 4.x (http://www.allegro.cc/)
   – biblioteka multimedialna (grafika, muzyka, klawiatura, mysz itp.)

2. **Boost** (http://www.boost.org/)
   – rozszerzenie standardu C++ (m.in. sprytne wskaźniki, wątki, funkcje matematyczne)

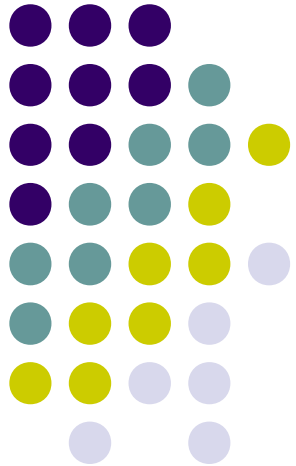
3. **Ncurses** (http://www.gnu.org/software/ncurses/)
   – biblioteka „graficzna” pod konsolę

4. **Qt** (https://www.qt.io)

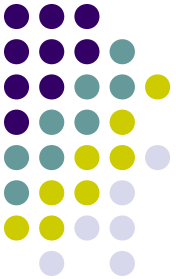
Standard C++11

Nie wszystko złoto

co się świeci z góry...



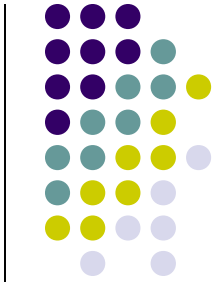
Standard C++11



1. Początek prac ok. 2002 r. (C++0x)
2. Pierwszy szkic standardu – wrzesień 2008 r.
3. Zatwierdzenie standardu – sierpień 2011 r.
4. Eksperymentalne wsparcie standardu C++0x od GCC 4.3 do 4.6
5. Pełne (nadal eksperymentalne) wsparcie C++11 od GCC 4.7
6. Pełne wsparcie C++11 od GCC 4.8.1
7. C++11 wprowadza wiele nowych fantastycznych elementów języka!

Więcej: https://gcc.gnu.org/projects/cxx-status.html#cxx11

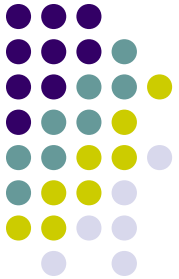
Standard C++11 – nowości



1. Referencje do r-wartości (&&)
2. Uogólnione wyrażenia stałe (`constexpr`)
3. Szablony zewnętrzne (`extern template`)
4. Listy inicjujące (`std::initializer_list`)
5. Automatyczne określenie typu (auto)
6. Pętla *for* oparta na zakresie
7. Funkcje i wyrażenia *lambda*
8. Ustawianie metod jako *default* lub *delete*
9. Ułatwienie używania wątków
10. Typy krotkowe (`tuple`)
11. Wyrażenia regularne
12. Sprytne wskaźniki
13. Itp., itd....

Więcej: http://pl.wikipedia.org/wiki/C%2B%2B11

Standard C++11 – GCC



GCC 4.3 – 4.6:

```
-std=c++0x
```

GCC 4.7:

```
-std=c++11
```

Więcej: http://gcc.gnu.org/projects/cxx0x.html

Programowanie jest fantastyczne!!!