Young Programmer

www.inceptio.org.pl

Organizator:

fundacja INCEPTIO

Partnerzy:

Wydział Fizyki Politechnika Warszawska

Stowarzyszenie KLATRAT

Dofinansowanie:

Miasto Stołeczne Warszawa

Zakochaj się w Warszawie

Young Programmer

www.inceptio.org.pl

Zajęcia nr 3

Programowanie strukturalne i obiektowe

dr inż. Łukasz Graczykowski
mgr inż. Leszek Kosarzewski
Wydział Fizyki Politechniki Warszawskiej

Organizator:

fundacja INCEPTIO

Partnerzy:

Wydział Fizyki Politechnika Warszawska

Stowarzyszenie KLATRAT

Dofinansowanie:

Miasto Stołeczne Warszawa

Zakochaj się w Warszawie

Plan

- Dokończenie zadań z zajęć nr 2
- Programowanie obiektowe cz. 1
  - Struktury i klasy
  - Podział na jednostki kompilacji

Zadanie 3.a. Rzut kostką

Napisz program realizujący rzut kostką n-ścienną.

a)Wylosuj liczbę oczek.
Skorzystaj z instrukcji preprocesora umożliwiającej generację liczb pseudolosowych (`#include <stdlib.h>`).

W funkcji głównej zdeklaruj dwie zmienne typu `int sciany,rzut;` , które będą przechowywać informacje o ilości ścian kostki i wyniku rzutu kością.

Wywołaj funkcję **srand(time(0));** Ustawia ona punkt początkowy generatora liczb pseudolosowych na wartości 0. Co więcej każda generacja będzie niepowtarzalna (na miarę generatora liczb pseudolosowych).

Poproś użytkownika o podanie ilości ścian kostki i przypisz ją do zmiennej **sciany** (za pomocą komend cout i cin).

Skorzystaj z pętli do-while z warunkiem (koniec =='t'). W instrukcji pętli wylosuj liczbę oczek w kostce za pomocą komendy **rand()% sciany + 1;** . Wylosowaną wartość wypisz na ekran. Zapytaj się użytkownika, czy chce kontynuować (niech wybierze odpowiedz t/n) i pobierz odpowiedź.

Zadanie 3.b. Rzut kostką

b)Wylosuj liczbę oczek. Oblicz sumę z wszystkich losowań i wypisz ich liczbę.

Przerób poprzedni program. Instrukcję losowania przenieś do funkcji zewnętrznej, która pobiera wartość int (ilość oczek) oraz zwraca wynik losowania w formie zmiennej typu `int` (`return wartosc;`).

W funkcji main() w instrukcji pętli dopisz kilka linii. Po rzucie kostką dodaj wynik rzutu do nowej zmiennej `int suma`. Następnie zwiększ wartość nowej zmiennej `int n` o 1 (podczas definiowania zmiennej **n** nadaj jej wartość 0).

Przed zakończeniem głównej funkcji main() wypisz na ekran informację o liczbie wykonanych rzutów, liczbie ścian kostki oraz sumę rzutów.

Zadanie 3.c. Rzut kostką

c)Wylosuj liczbę oczek. Oblicz sumę z wszystkich losowań i wypisz ich liczbę. Zapamiętaj informację o wynikach w tablicy.

Przerób poprzedni program. W głównej funkcji main() stwórz jednowymiarową tablicę **rzut[]** typu int o dużym rozmiarze np. 100. Pamiętaj o nadaniu początkowych wartości wszystkim elementom 0.

Wyniki losowań wpisuj do kolejnych elementów tablicy (skorzystaj ze zmiennej **n**).

Na końcu, przed instrukcją `return 0;` wypisz wszystkie zapisane wyniki na ekran. Skorzystaj z pętli `for`, nowego iteratora `int i` oraz zmiennej **n**.

Zadanie 3.a. Rzut kostką

```
#include <iostream>
#include <stdlib.h>

using namespace std;

int main()
{
    int sciany;     // Zmienna przechowujaca informacje o ilosci scian kostki
    int rzut;         //Zmienna przechowujaca wynik rzutu kostka
    char koniec; //Zmienna przechowujaca wartosc odpowiedzi na pytanie t/n

    srand (time(0));

    cout << "Podaj ile scian ma kostka." << endl;
    cin >> sciany;

    do
    {
          rzut = rand ()% sciany + 1;
          cout << "Wynik rzutu:" << rzut << endl;
          cout << "Czy chcesz rzucic jeszcze raz? Odp t/n" << endl;
          cin >> koniec; //Pobranie do zmiennej wybor odpowiedzi na powyzsze
pytanie

    }
    while (koniec == 't');

    return 0;
}
```

Instrukcja preprocesora umożliwiająca wykorzystanie generatora liczb pseudolosowych rnd()

Ustawienie punktu początkowego generatora liczb pseudolosowych. Jeśli time(n), to n=NULL lub n=0.

Losowanie liczby całkowitej z przedziału <1, sciany>

Pętla **do-while** (pętla będzie wykonywana dopóki wartość wyrażenia () jest true

Zadanie 3.b. Rzut kostką

```
#include <iostream>
#include <stdlib.h>

using namespace std;

int losowanie(int max)
{
    return rand ()% max + 1;
}

int main()
{
    int sciany;
    int rzut, suma = 0, n = 0;
    char koniec;

    srand (time(0));

    cout << "Podaj ile scian ma kostka." << endl;
    cin >> sciany;

    do
    {
        rzut = losowanie(sciany);
        suma = suma + rzut;
        n = n + 1;
        cout << "Wynik rzutu:" << rzut << endl;
        cout << "Czy chcesz rzucic jeszcze raz? Odp t/n" << endl;
        cin >> koniec;
    }
    while (koniec == 't');

    cout<<"Rzucales "<<n<<" razy kostka o ilosci scian = "<<sciany<<". Suma rzutow =
"<<suma<<endl;

    return 0;}
```

max - zmienna zapisująca ilość ścian kostki

Losujemy wynik rzutu kostką.

Dodajemy wynik nowego rzutu do wartości sumy.

Powiększamy wartość zmiennej n o 1,
po wykonaniu rzutu i dodaniu jego wyniku do sumy.

Zadanie 3.c. Rzut kostką

```
#include <iostream>
#include <stdlib.h>

using namespace std;

int losowanie(int max)
{
    return rand ()% max + 1;
}

int main()
{
    int sciany;
    int rzut[100] = {}, suma = 0, n = 0;
    char koniec;

    srand (time(0));

    cout << "Podaj ile scian ma kostka." << endl;
    cin >> sciany;

    do
    {
        rzut[n] = losowanie(sciany);
        suma = suma + rzut[n];
        cout << "Wynik rzutu:" << rzut[n] << endl;
        n = n + 1;
        cout << "Czy chcesz rzucic jeszcze raz? Odp t/n" << endl;
        cin >> koniec;
    }
    while (koniec == 't');

    cout<<"Rzucales "<<n<<" razy kostka o ilosci scian = "<<sciany<<". Suma rzutow = "<<suma<<endl;
    cout<<"Wylosowane liczby to:"<<endl;
    for(int i=0 ; i < n ; i++)
    {
        cout<<rzut[i]<<endl;
    }
    return 0;}
```

Tworzymy dużą tablicę o wartościach początkowych =0.

Losujemy wynik rzutu kostką. Zapisanie wyniku w pierwszym wolnym elemencie tablicy.

Dodajemy wynik nowego rzutu do wartości sumy.

Wypisujemy wszystkie wyniku rzutu kostką.

Programowanie obiektowe

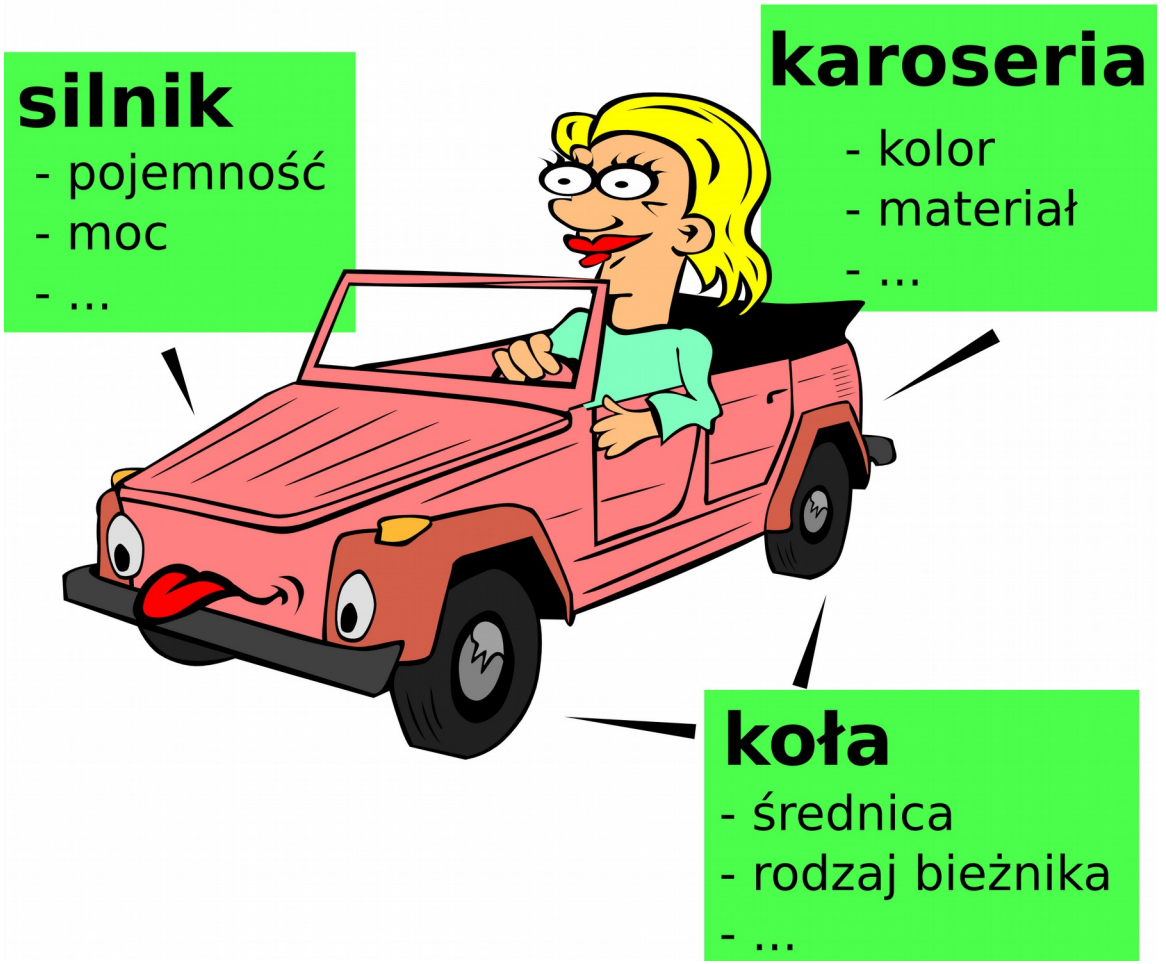
Programowanie obiektowe (ang. *object-oriented programming*) — paradygmat programowania, w którym programy definiuje się za pomocą **obiektów** — elementów łączących stan (czyli **dane**, nazywane najczęściej polami) i **zachowanie** (czyli procedury, tu: metody). Obiektowy program komputerowy wyrażony jest jako zbiór takich obiektów, komunikujących się pomiędzy sobą w celu wykonywania zadań.

Podejście to różni się od tradycyjnego programowania proceduralnego, gdzie dane i procedury nie są ze sobą bezpośrednio związane. Programowanie obiektowe ma ułatwić pisanie, konserwację i wielokrotne użycie programów lub ich fragmentów.

Żródło: https://pl.wikipedia.org/wiki/Programowanie_obiektowe

Idea “obiektu” - przykład

Samochód jako obiekt...
złożony z obiektów



Źródło wszystkich obrazków w tej prezentacji: http://openclipart.org

Struktury

Deklaracja struktury:

```
struct nazwa_struktury
{
typ1 skladowa1;
typ2 skladowa2;
//...
};
```

WAŻNY ŚREDNIK!

ALBO

```
typedef struct
{
typ1 skladowa1;
typ2 skladowa2;
//...
} nazwa_struktury;
```

WAŻNY ŚREDNIK!

WARTO WIEDZIEĆ

- Rozmiar obiektu struktury jest nie mniejszy niż suma rozmiarów elementów składowych struktury.
- Wszystkie elementy składowe struktury zajmują własne miejsce w pamięci.
- Stąd w każdej chwili mamy dostęp do wartości wszystkich elementów składowych.
- Definicja struktury jest definicją nowego typu (użytkownika), a nie konkretnego obiektu.
- Struktura może zawierać funkcje, które są dodatkowymi składowymi danej struktury.

Inicjacja struktury

- **Sposób 1:**

```
struct nazwa_struktury nazwa_obiektu = {wartość1, wartość2, …};
```

- **Sposób 2 (elementy niewymienione na liście inicjalizacyjnej nie zostaną zainicjalizowane):**

```
struct nazwa_struktury nazwa_obiektu = {.element1 = wartość1, .element3 = wartość2};
```

- **Sposób 3 – kopiowanie innego obiektu:**

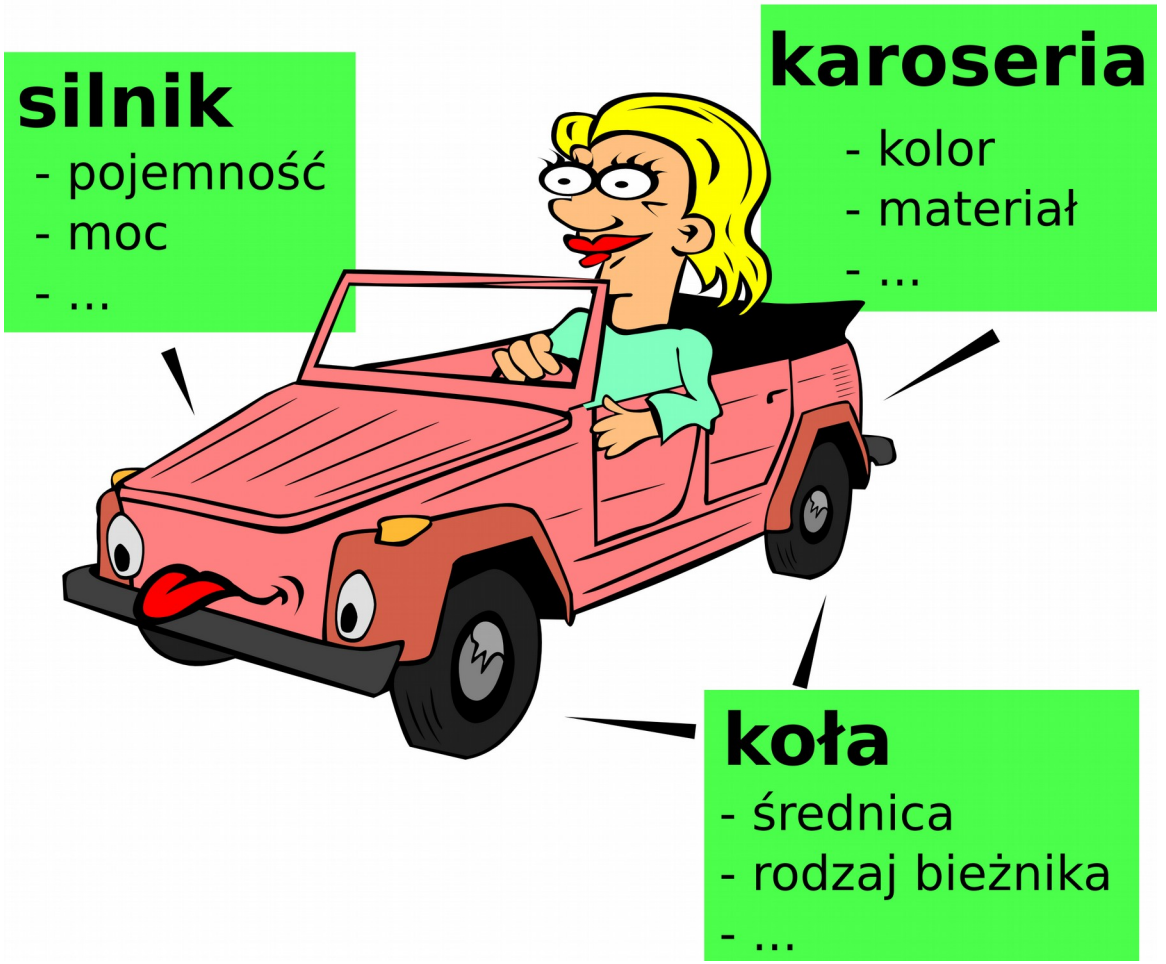
```
struct nazwa_struktury nazwa_obiektu = inny_obiekt_tej_samej_struktury;
```

Dobra rada

ZAWSZE zaczynajcie pisanie swojego programu od NAJMNIEJSZEJ możliwej kompilującej się wersji.

Idea “obiektu” - przykład

Samochód jako obiekt...
złożony z obiektów



Źródło wszystkich obrazków w tej prezentacji: http://openclipart.org

Minimalistyczny kod (struktura)

```
#include <iostream>
#include <string>

struct Silnik
{
double pojemnosc;
double moc;
};

using namespace std;

int main()
{
struct Silnik testowy_silnik; //Tworzymy silnik
testowy_silnik.moc = 100.0; //Ustawiamy moc silnika na 100 [KM]
cout << "Moc silnika: " << testowy_silnik.moc << " KM!" << endl;

return 0;
}
```

WAŻNY ŚREDNIK!

Napisaliśmy kod w wersji minimalistycznej, a teraz kompilujemy go i poprawiamy ewentualne błędy (nie powinno ich być dużo i powinny być łatwe do namierzenia).

Operator “.”

```
obiekt.skladnik
referencja.skladnik
```

Operator “.” - operator odniesienia się do składnika obiektu który znamy z nazwy lub referencji.

Przykład:

```
testowy_silnik.moc = 100.0;
testowy_silnik.pojemnosc = 4;
```

Rozbudowujemy kod

```
#include <iostream>
#include <string>

struct Silnik
{
double pojemnosc;
double moc;
};
struct Kolo
{
unsigned int srednica;
std::string typ_bieznika;
};
typedef struct
{
std::string kolor;
std::string material;
} Karoseria;

using namespace std;

int main()
{
struct Silnik testowy_silnik; //Tworzymy silnik
testowy_silnik.moc = 100.0; //Ustawiamy moc silnika na 100 [KM]
cout << "Moc silnika: " << testowy_silnik.moc << " KM!" << endl;

return 0;
}
```

WAŻNY ŚREDNIK!

WAŻNY ŚREDNIK!

Definicja struktury z użyciem **typedef**

WAŻNY ŚREDNIK!

Kompilujemy i poprawiamy ewentualne błędy...

Kod dla naszego przykładu (obiekty)

```
#include <iostream>
#include <string>

struct Silnik
{
double pojemnosc;
double moc;
};
struct Kolo
{
unsigned int srednica;
std::string typ_bieznika;
};
typedef struct
{
std::string kolor;
std::string material;
} Karoseria;

struct Samochod
{
struct Silnik silnik;
struct Kolo kolo[4];
Karoseria karoseria;
};
```

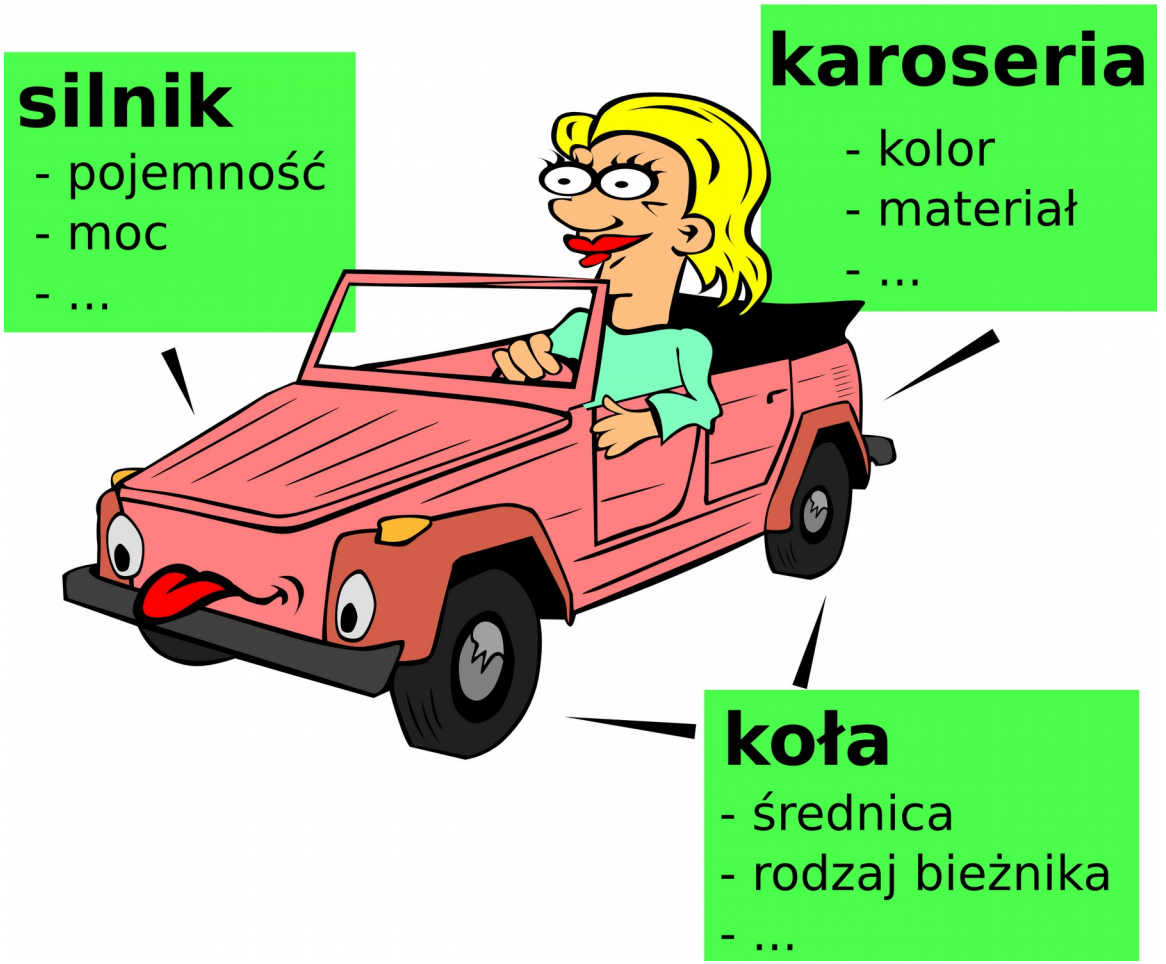
Kod dla naszego przykładu (main)

```
using namespace std;

int main()
{
struct Silnik testowy_silnik; //Tworzymy silnik
testowy_silnik.moc = 100.0; //Ustawiamy moc silnika na 100 [KM]

struct Samochod samochod; //Tworzymy samochod
samochod.silnik = testowy_silnik; //Montujemy nasz testowy silnik w samochodzie
cout << "Moc silnika wynosi az: " << (samochod.silnik).moc << " KM!" << endl;
(samochod.silnik).moc *= 2; //Podwajamy moc silnika
cout << "Moc silnika wynosi teraz: " << samochod.silnik.moc << " KM!" << endl;

//Pora na zmiane karoserii
Karoseria karoseria;
karoseria.kolor = "czerwony";
samochod.karoseria = karoseria;
cout << "Kolor karoserii to: " << samochod.karoseria.kolor << endl;

return 0;
}
```

W ten sposób doszliśmy do kodu opisującego samochód z naszego przykładu.

Idea “obiektu” - przykład

Samochód jako obiekt...
złożony z obiektów



Źródło wszystkich obrazków w tej prezentacji: http://openclipart.org

Podział na jednostki kompilacji

samochod.hpp

```
#include <iostream>
#include <string>

struct Silnik
{
double pojemnosc;
double moc;
};
struct Kolo
{
unsigned int srednica;
std::string typ_bieznika;
};
typedef struct
{
std::string kolor;
std::string material;
} Karoseria;

struct Samochod
{
struct Silnik silnik;
struct Kolo kolo[4];
Karoseria karoseria;
};
```

main.cpp

```
#include "samochod.hpp"

using namespace std;

int main()
{
struct Silnik testowy_silnik;
testowy_silnik.moc = 100.0;

struct Samochod samochod;
samochod.silnik = testowy_silnik;
cout << "Moc silnika wynosi az: " << (samochod.silnik).moc << " KM!" <<
endl;
(samochod.silnik).moc *= 2;
cout << "Moc silnika wynosi teraz: " << samochod.silnik.moc << " KM!" <<
endl;

Karoseria karoseria;
karoseria.kolor = "czerwony";
samochod.karoseria = karoseria;
cout << "Kolor karoserii to: " << samochod.karoseria.kolor << endl;

return 0;
}
```

Klasy

Deklaracja klasy:

```
class nazwa_klasy
{
public:
//konstruktory
//destruktory
//inne funkcje (metody)

private:
typ1 skladowa1;
typ2 skladowa2;
//...
};
```

WAŻNY ŚREDNIK!

WARTO WIEDZIEĆ

Klasa jest niemal identyczna ze strukturą, jednak domyślnie jej składowe i metody są **prywatne**. Oznacza to, że tylko metody tej klasy mogą mieć do nich dostęp – są ukryte przed bezpośrednią ingerencją użytkowników.

Prywatność

Jeśli chcemy wywołać poza klasą jej prywatne składowe, to musisz stworzyć specjalne funkcje, które będą publiczne np.

```
class silnik{
int pojemnosc, moc;
};
```

Jeśli chcesz wypisać zawartość obiektów silnik.pojemność lub silnik.moc, to stwórz następujące funkcje:

```
class silnik{
int pojemnosc, moc;
public:
int getPojemnosc(){return pojemnosc;}
int getMoc(){return moc;}
};
```

Jeśli chcesz zmienić wartość obiektów silnik.pojemność lub silnik.moc, to stwórz następujące funkcje publiczne: `void setPojemnosc(int pobrana_pojemnosc)` oraz `void setMoc(int pobrana_moc){moc = pobrana_moc;}`. Wtedy cała klasa:

```
class silnik{
int pojemnosc, moc;
public:
int getPojemnosc(){return pojemnosc;}
int getMoc(){return moc;}
void setPojemnosc(int pobrana_pojemnosc){pojemnosc = pobrana_pojemnosc;}
void setMoc(int pobrana_moc){moc = pobrana_moc;}
};
```

Zaprzyjaźnianie funkcji

Jeśli klasa ma obiekty albo funkcje prywatne, to nie mogą być one wywołane poza klasą – czyli żadna funkcja spoza klasy nie może się do nich dostać.
Jest możliwość, aby konkretna funkcja dostała upoważnienie dostępu do prywatnych składowych – należy ją zaprzyjaźnić z klasą.

```
class silnik
{
private: int pojemnosc, moc;
...
friend void funkcja(int);
};

void funkcja(int zmienna)
{
…
}

int main()
{   …
return 0;
}
```

Zmienne prywatne

Zaprzyjaźnienie funkcji

funkcja zaprzyjaźniona

cout<<silnik.moc<<endl;

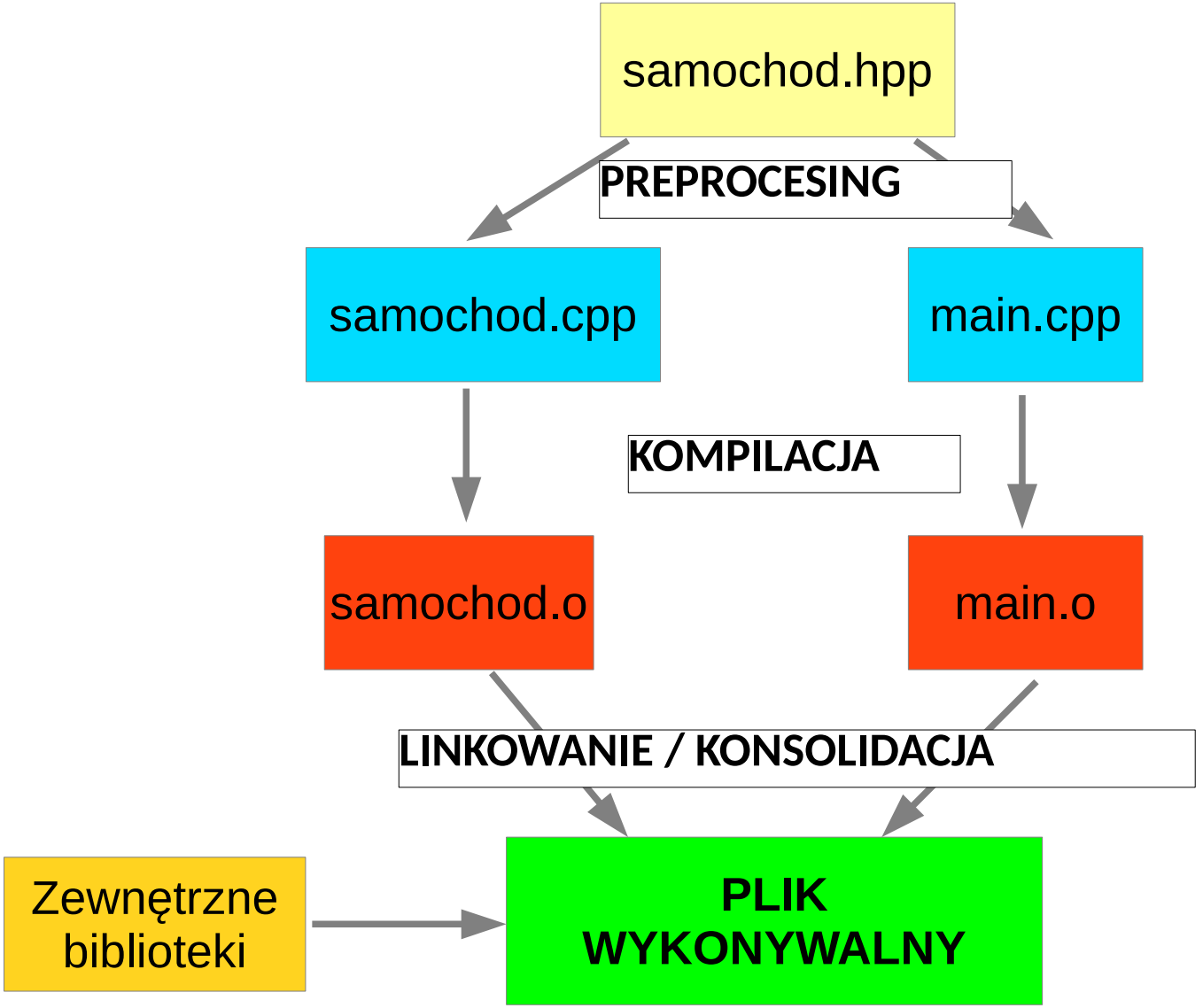
Po zaprzyjaźnieniu możemy teraz wywołać zmienne pojemnosc i moc korzystając z operatora '.'

Kompilacja wielu plików

Każdą z tych linijek trzeba wpisać/skopiować po kolei do terminala:

```
g++ -c samochod.cpp -o samochod.o
g++ -c main.cpp -o main.o
g++ samochod.o main.o -o samochod
```

Program złożony z wielu plików



Makefile

Makefile

```
all:
	g++ -c samochod.cpp -o samochod.o
	g++ -c main.cpp -o main.o
	g++ samochod.o main.o -o samochod
clean:
	rm *.o
```

I teraz wystarczy wpisać w konsoli (w katalogu z kodem i Makefilem):

make

aby wszystkie pliki się skompilowały!

Young Programmer

www.inceptio.org.pl

Zajęcia nr 3

Programowanie obiektowe cz. 1

Zadania

Organizator:

fundacja INCEPTIO

Partnerzy:

Wydział Fizyki Politechniki Warszawskiej

Stowarzyszenie KLATRAT

Dofinansowanie:

Miasto Stołeczne Warszawa

Zakochaj się w Warszawie

Zadanie 1: Policz piłki

Stwórz strukturę **pilka** zawierającą **char kolor;//'R','G','B' oraz int masa;**

Napisz kod ze zmiennymi globalnymi - ponad funkcją **main():**

```
int n=0;    // zmienne globalne są dostępne “globalnie”
Pilka tab[5];  // czyli dla wszystkich funkcji w całym programie

int main()
{…}
```

Napisz funkcję **void dodaj(char kol,int m),** która ustawia kolor i masę n-tej piłki **tab[n].kolor=kol;** oraz masę, a także zwiększy licznik **n**.

Napisz pętlę **for**, w której wczytywać będziemy kolor piłki **'R','G','B'** i masę dla 5 piłek.

Na koniec policzmy oddzielnie masę piłek zielonych, czerwonych i niebieskich. Trzeba będzie sprawdzić kolor piłki **tab[n].kolor=='R';**

Wszystko piszemy w jednym pliku!

Zadanie 2. Wielomian

Tworzymy strukturę **wielomian**.
a) Składowe struktury: **double a, b** oraz string **nazwa**.

b) W funkcji **main()** tworzymy obiekty typu wielomianil ustawiamy mu wszystkie składniki, po czym wypisujemy je na ekran

c) Zmieniamy słowo kluczowe “struct” na “class” → co się dzieje? (Dodać “public:” wewnątrz klasy przed polami

d) Tworzymy funkcję wewnątrz klasy **void Ustaw(double wsp_a, double wsp_b, string name)**.

e) Tworzymy zewnętrzną funkcję **void wypisz(wielomian)**, która za pomocą komendy cout wypisuje na ekran nazwę wielomianu (nazwa) i równanie liniowe **a**x + **b** (na ekranie widzimy np.: wielomian first: 2.5x + 5.0). Funkcja ma być zaprzyjaźniona (friend) z klasą **wielomian**.

Zadanie 3a:

Napisz program, który pobierał będzie z pliku tekstowego dane osobowe, wyświetlał je na ekranie i pozwalał na manipulowanie (zmianę, usuwanie) nimi. Reprezentacją osoby w programie ma być odpowiednio zaprojektowana klasa. “Osoby” mogą być przechowywane w tablicy (wektorze lub liście).

Zadanie 3b:

Prosty sposób na pobranie danych z pliku zawierającego dane w formie dwóch kolumn liczb zmiennoprzecinkowych:

dane.txt

expab (USD)	expba (USD)
2.29191e+11	1.74616e+11
2.78e+08	1.053e+09
3.09e+08	5.72e+08
4.474e+09	4.423e+09
6.69e+08	1.36e+09
2.353e+09	1.104e+09

```
#include <cstdlib>
#include <fstream>

//...

double col_one, col_two; //zmienne na dane z kolumn

ifstream input("dane.txt"); //strumien wejsciowy

input.ignore(1000, '\n'); //ignoruj pierwsza linie
while (!input.eof())
{
input >> col_one >> col_two;
cout << col_one << '\t' << col_two << endl;
}

//...
```

Zadanie 4a: Prosta gra RPG

Napiszmy prymitywną grę. W głównej funkcji stworzymy dwie postaci (bohatera i potworka). W grze postaci te będą walczyć. Będziemy również monitorować stan obu postac, oraz nadawać nowe parametry istniejącym postaciom. Do wszystkiego wykorzystamy klasę z odpowiednimi metodami.

Zadanie 4b: Tworzenie postaci

Tworzymy klasę **postac**.
a) Składowe **private: int zycie, obrazenia, l_ciosow** oraz **string imie**.
b) Metody **public: UstawPostac(int życie, int obrazenia, int l_ciosow, string imię); UsunPostac()**
d) Stwórz zewnętrzną funkcję zaprzyjaźnioną **void Walcz(postac &a, postac &b)**. Będzie ona odpowiedzialna za rozstrzygnięcie potyczki. Korzystając z referencji działamy bezpośrednio na naszych postaciach!

Działanie funkcji **Walcz**:
– życie postaci **a** ma być równe różnicy jej życia oraz obrażeń zadanych przez postać **b** pomnożonych przez liczbę ciosów postaci **b**.
– życie postaci **b** ma być równe różnicy jej życia oraz obrażeń zadanych przez postać **a** pomnożonych przez liczbę ciosów postaci **a**.
– wstawiamy warunek **if(a.zycie <= 0)** to ustawiamy wartość życia tej postaci na 0 i wypisujemy na ekran, że postać **a** poległa (z wykorzystaniem komendy **cout**).
– wstawiamy warunek **if(b.zycie <= 0)** to ustawiamy wartość życia tej postaci na 0 i wypisujemy na ekran, że postać **b** poległa (z wykorzystaniem komendy **cout**).

Zadanie 4c: Interakcja z graczem

a) Tworzymy funkcję **void WypiszPostac()**

Działanie funkcji:

– Na ekran wypisujemy **imie** postaci **a**.
– Wypisujemy, ile zostało jej życia.
– Wypisujemy jakie zadanie obrażenia i ile zadaje ciosów, oraz łączną wartość obrażeń na turę.

b) Przeciążamy funkcję **UstawPostac**: tworzymy nową funkcję **void UstawPostac()**, która nie przyjmuje żadnych argumentów

Działanie funkcji:

– Na ekranie wypisujemy polecenie dla użytkownika, aby podał parametry postaci.
– Niech poda **imie** postaci. (**cin >> a.imie;**)
– Niech poda początkową wartość życia.
– Niech poda jakie zadaje obrażenia i ile zadaje ciosów na turę.

Zadanie 4d: Fragment maina

```
int main()
{
	postac bohater;
	bohater.UstawPostac(100,30,2,"Zenon Waleczny"); //tworzymy bohatera
	postac mobek;
	mobek.UstawPostac(50,5,1,"bazyliszek"); //tworzymy potwora
	Walcz(bohater,mobek); //tu rozgrywa sie bitwa

	//czesc dla zadnia 4c --------->
	bohater.WypiszPostac(); // wypisujemy na ekran parametry bohatera
	mobek.WypiszPostac(); // wypisujemy na ekran parametry potwora
	Mobek.UstawPostac(); // nadajemy nowe parametry potworka
	Walcz(bohater,mobek);
	bohater.WypiszPostac();
	mobek.WypiszPostac();
	//czesc dla zadania 4c <----------

	return 0;
}
```

Zadanie 4e: Rozbudowa programu

Puść wodze fantazji!
a) Stwórz kolejne funkcje / metody klasy i nadaj nowe opcje grze np. leczenie bohatera, zdobywanie poziomu postaci (dodatkowy parametr w klasie odpowiadający za doświadczenie??) i wiele innych!
b) Wprowadź więcej tur podczas walki, np. aby walka rozgrywała się do momentu spadku **życia** postaci do poziomu <=0 . Możesz to zrealizować wstawiając do operator-pętlę do-while.
c) Wstaw całą grę w pętlę while(koniec), gdzie **bool koniec = true**. Jeśli w czasie gry chcesz ją zakończyć, to zmień wartość koniec na false.
d) Stwórz w pętli interfejs użytkownika korzystając z switch!
e) **Dla bardziej wtajemniczonych**: Temat może zostać rozwinięty o tematykę dziedziczenia. Można stworzyć nowe klasy dziedziczące po klasie postać. Sugerowane nowe klasy: bohater i potwor. Co więcej po klasie bohater mogą dziedziczyć nowe klasy np. wojownik, mag, druid, kaplan.

NIECH PRZYGODA TRWA!!