Informatyka (A1)

Laboratorium 2 : Podstawy programowania obiektowego C++

Materiały:

- Książka: Symfonia C++, Jerzy Grębosz.
- Wykład: http://ssamolej.kia.prz.edu.pl/dydaktyka/inf_1EE_ZI/inf_1EE_ZI_wyk.htm, dr inż. Sławomir Samolej.
- Zasoby WWW: http://pl.wikibooks.org/wiki/C++ , http://www.cplusplus.com.
- Wujek google.

Zrealizuj poniższe zadania → jeśli masz jakiś problem to zapytaj prowadzącego.

1. Tworzenie klasy C++ (przed funkcją main → pierwszy sposób nie zalecany dla długich programów):

Linia	Kod programu	Komentarz
1	`#include <iostream>`	• pierwszy program pokazujący w jaki sposób deklarować klasę i w jaki sposób, stworzyć obiekt klasy:
2	`using namespace std;`	
3		
4	`class kwadrat`	○ definicja klasy (klasa nic nie robi, ma jedynie charakter demonstracyjny) → w niej zamieszczamy potrzebne zmienne i funkcje
5	`{`	
6	`//pusta klasa...`	
7	`};`	
8		
9	`int main()`	○ nazwa klasy
10	`{`	
11	`kwadrat a;`	○ stworzenie obiektu klasy kwadrat **(ponieważ nie ma zdefiniowanego konstruktora, to stworzony zostaje konstruktor domyślny bezargumentowy, który nic nie robi)**
12		
13	`//system("pause");`	
14	`return 0;`	
15	`}`	

2. Pola, metody, etykiety.

➢ **Pola klasy, etykiety – program 1:**

Linia	Kod programu	Komentarz
1	`#include <iostream>`	• etykieta **(**są jeszcze **protected** oraz **private)**
2	`using namespace std;`	
3		
4	`class kwadrat`	• zmienne tzw. **pola klasy,** deklaracja zmiennych jak w zwykłym programie
5	`{`	
6	`public:`	
7	`int bok;`	
8	`float obwod;`	
9	`float pole;`	
10	`};`	• definicja klasy
11		
12	`int main()`	• wczytanie boku kwadratu w linii 18 (pola o nazwie bok) → **dostęp do pól i metod klasy poza klasą**
13	`{`	
14	`kwadrat kw;`	

```
16
17          cout << "Podaj bok: ";
18          cin >> kw.bok;
19
20          kw.pole = kw.bok * kw.bok;
21          kw.obwod = 4 * kw.bok;
22
23          cout << "Pole kwadratu: " <<
kw.pole << endl;
24          cout << "Obwod kwadratu: " <<
kw.obwod << endl;
25
26          //system("pause");
27          return 0;
28  }
29
30
```

odbywa się poprzez nazwę obiektu, kropkę i nazwę pola/zmiennej (kw.bok)

- obliczenie pola oraz obwodu kwadratu wg wzorów matematycznych
- wydrukowanie wyników

1. Dostęp do pól klasy <u>na zewnątrz klasy</u> jest możliwy **tylko i wyłącznie dzięki etykiecie public**, zamień słowo public kolejno na słowa protected, private – czy dla którejś z tych etykiet zbuduje się program?

- Gdy nie ma dziedziczenia między klasami, to etykieta public i protected ma takie samo działanie.

➢ Metody klasy, 1 sposób definicji metod klasy:

Linia	Kod programu	Komentarz

```
1   #include <iostream>
2   using namespace std;
3
4   class kwadrat
5   {
6   private:
7           float obwod;
8           float pole;
9   public:
10          int bok;
11          float ObliczObwod()
12          {
13                  obwod = 4*bok;
14                  return obwod;
16          }
17          float ObliczPole()
18          {
19                  pole = bok * bok;
20                  return pole;
21          }
21  };
22
23  int main()
24  {
25          kwadrat kw;
26
27          cout << "Podaj bok: ";
28          cin >> kw.bok;
29
30          float poleKwadratu =
kw.ObliczPole();
31          float obwodKwadratu =
kw.ObliczObwod();
32          cout << "Pole kwadratu: " <<
33  poleKwadratu << endl;
            cout << "Obwod kwadratu: " <<
34  obwodKwadratu << endl;
```

- dwa pola prywatne niedostępne poza klasą
- funkcja w klasie to tzw. **metoda klasy:**
  - metoda obliczająca i zwracająca wartość obwodu, na podstawie pola **bok (pola/metody z etykietą public są dostępne na zewnątrz i wewnątrz klasy natomiast z etykietą private/protected jedynie wewnątrz klasy**)
  - metoda obliczająca i zwracająca na wyjściu pole kwadratu, na podstawie pola **bok**
  - **dostęp do pól/metod wewnątrz klasy odbywa się poprzez nazwę - linijki: 13, 14, 19, 20**
- wczytywanie boku odbywa się tak jak wcześniej
- zmienia się sposób obliczania pola i obwodu, robi się to poprzez metody klasy
- dostęp do metod na zewnątrz klasy jest na takiej samej zasadzie jak dostęp do pól (nazwa obiektu + kropka + nazwa metody, oczywiście pole/metoda musi mieć etykietę public)
- wydrukowanie wyniku (zamiast tworzenie zmiennych obwodKwadratu, poleKwadratu, można od razu wydrukować wartość pola i obwodu

35
36
37
38
	`//system("pause");`
	`return 0;`
	`}`
		przy wywołaniu odpowiednich metod, bo zwracają wartości np.:
		`cout << "Pole kwadratu: " << kw.ObliczObwod();`

➢ Konstruktor (metoda wywoływana przy tworzeniu obiektu):

Linia	Kod programu	Komentarz

1
2
3
4
5
6
7
8
9
10
11
12
13
14
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
	`#include <iostream>`
	`using namespace std;`
	
	`class kwadrat`
	`{`
	`private:`
	`float obwod;`
	`float pole;`
	`int bok;`
	`public:`
	`kwadrat(int _bok)`
	`{`
	`bok = _bok;`
	`pole = 0;`
	`obwod = 0;`
	`}`
	`float ObliczObwod()`
	`{`
	`obwod = 4*bok;`
	`return obwod;`
	`}`
	`float ObliczPole()`
	`{`
	`pole = bok * bok;`
	`return pole;`
	`}`
	`};`
	
	`int main()`
	`{`
	`kwadrat kw(3);`
	
	`float poleKwadratu = kw.ObliczPole();`
	`float obwodKwadratu = kw.ObliczObwod();`
	
	`cout << "Pole kwadratu: " << poleKwadratu << endl;`
	`cout << "Obwod kwadratu: " << obwodKwadratu << endl;`
	
	`system("pause");`
	`return 0;`
	`}`
		• teraz zmienna bok została zdefiniowana z etykietą private i dostępna jest tylko wewnątrz klasy, jej wartość można ustawiać tylko poprzez metody
		• konstruktor to metoda klasy, która jest wywoływana w momencie definicji obiektu
		
		• w poprzednich programach, pomimo iż konstruktor nie był tworzony przez programistę to był tworzony przez kompilator niejawny konstruktor, który nic nie robi: kwadrat() {}
		• skoro konstruktor ma jeden parametr (linia 11) do zainicjowania pola bok, to należy tworzyć obiekt z podaniem jednego parametru (linia 32)
		• konstruktor służy do nadania początkowych wartości dla pól klasy
		• wartość w linijce 32 podanej jako parametr konstruktora można by równie dobrze wczytać
		
		1. Spróbuj zamienić linię 32 na:
		`kwadrat kw;`
		Co się stanie przy budowaniu programu?
		
		• skoro został stworzony konstruktor przez programistę, to kompilator nie tworzy już pustego konstruktora bezargumentowego tzw. **niejawnego**, dlatego pojawia się błąd
		
		Konstruktor:
		• nie zwraca wartości
		• inicjuje wartości pól klasy
		• wywoływany tylko przy tworzeniu obiektu, nie można z niego korzystać

		jak z normalnej metody • można go przeciążać

➢ **A co jeśli w poprzednim programie, będzie potrzeba obliczenia pola/obwodu kwadratu dla kilku wartości boków oraz będzie potrzeba tworzenia obiektu klasy kwadrat bez argumentów (trzeba stworzyć metodę zmieniającą bok kwadratu oraz przeciążyć konstruktor):**

Linia	Kod programu	Komentarz
1	`#include <iostream>`	• **przeciążenie konstruktorów**: ○ bezagrumentowy → wywołany w linii 50, przy tworzeniu obiektu ○ argumentowy (jeden argument) → wywołany w linii 49
2	`using namespace std;`	
3		
4	`class kwadrat`	
5	`{`	
6	`private:`	
7	`float obwod;`	
8	`float pole;`	
9	`int bok;`	
10	`public:`	
11	`kwadrat()`	
12	`{`	
13	`bok = pole = obwod = 0;`	• metoda do ustawiania pola ***bok*** (**tzw. setter**), może być używana wielokrotnie w przeciwieństwie do konstruktora
14	`}`	
15	`kwadrat(int _bok)`	
16	`{`	
17	`bok = _bok;`	
18	`pole = 0;`	
19	`obwod = 0;`	
20	`}`	
21	`void UstawBok(int _bok)`	
22	`{`	
23	`bok = _bok;`	
24	`}`	
25	`int ZwrocBok()`	• metoda zwracająca wartość pola ***bok*** (**tzw. getter**), może być używana na zewnątrz klasy → metoda dodana dla celów demonstracyjnych, mogłaby być potrzebna przy bardziej rozbudowanych programach
26	`{`	
27	`return;`	
28	`}`	
29	`float ObliczObwod()`	
30	`{`	
31	`obwod = 4*bok;`	
32	`return obwod;`	
33	`}`	
34	`float ObliczPole()`	
35	`{`	
36	`pole = bok * bok;`	
37	`return pole;`	
38	`}`	
39	`};`	
40		
41	`int main()`	
42	`{`	
43	`int bok1, bok2;`	
44	`cout << "Podaj bok 1 kwadratu: ";`	
45	`cin >> bok1;`	• wczytanie boków bok1, bok2 → linijki: 45, 47
46	`cout << endl << "Podaj bok 2 kwadratu: ";`	
47	`cin >> bok2;`	
48		
49	`kwadrat kw1(bok1);`	
50	`kwadrat kw2;`	
51	`kw2.UstawBok(bok2);`	

```
52
53          cout << endl << "Pole kwadratu 1: " << kw1.ObliczPole() << endl;
54          cout << "Obwod kwadratu 1: " << kw1.ObliczObwod() << endl;
55
56          cout << endl << "Pole kwadratu 2: " << kw2.ObliczPole() << endl;
57          cout << "Obwod kwadratu 2: " << kw2.ObliczObwod() << endl;
58
59          system("pause");
60          return 0;
61  }
```

- utworzenie obiektu z parametrem → konstruktor argumentowy
- utworzenie obiektu bez parametru → konstruktor bezargumentowy, trzeba ustawić bok później za pomocą **odpowiedniej metody**

3. Zmienne/metody statyczne oraz destruktor (metoda wywoływana przy usuwaniu obiektu):

Kod:

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

using namespace std;

class Klasa
{
private:
        static int liczbaInstancji;
public:
        Klasa()
        {
                ++liczbaInstancji;
        }
        ~Klasa()
        {
                --liczbaInstancji;
        }
        static int Liczba()
        {
                return liczbaInstancji;
        }
};
```

Komentarz

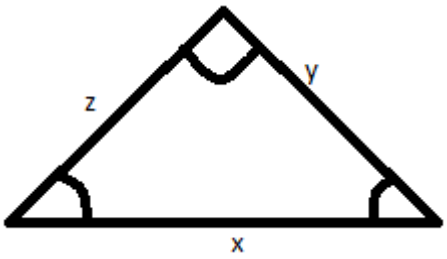
- zdefiniowanie przykładowej klasy o nazwie *Klasa*
- klasa posiada:
  - prywatne pole statyczne `liczbaInstancji`,
  - konstruktor
  - destruktor (nazwa klasy plus znak '~')
  - publiczną metodę statyczną `Liczba()`, która zwraca wartość pola `liczbaInstancji` (**tzw. getter**)

```
int Klasa::liczbaInstancji = 0;

int main()
{
        Klasa k;
        cout << Klasa::Liczba() << endl;
        {
                Klasa k1;
                cout << Klasa::Liczba() << endl;
        }

        cout << Klasa::Liczba() << endl;

        system("pause");
        return 0;
}
```

- zmienną statyczną należy zdefiniować przed funkcją main poza ciałem klasy : `int Klasa::liczbaInstancji = 0;`
- wywołanie metody statycznej odbywa się poprzez nazwę klasy, dwukropek oraz nazwę metody
- za każdym razem, gdy jest tworzony nowy obiekt klasy, automatycznie wywoływany jest konstruktor, który zwiększa zmienną statyczną `liczbaInstancji` o 1

	• analogicznie dzieje się z destruktorem, tylko wtedy wartość zmiennej zmniejszana jest o 1 • ciężko pokazać usunięcie obiektu k, ponieważ będzie on usuwany na końcu programu w momencie instrukcji return 0 • natomiast obiekt k1 zdefiniowany jako zmienna lokalna w bloku {} jest usuwany na końcu bloku, dlatego pozwala pokazać, że destruktor jest wywoływany przy usuwaniu obiektu

4. Utwórz klasę trójkąt .



- **Klasa powinna zawierać pola:**
  - x, y, z – prywatne, które przechowują boki trójkąta
  - obw2 – prywatne oznaczające połowę obwodu trójkąta
  - pole – oznaczające pole trójkąta
    Wszystkie pola typu **float.**
- **Klasa powinna zawierać metody:**
  - konstruktor bez argumentów, konstruktor kopiujący – wszystkie konstruktory mogą być zdefiniowane w deklaracji klasy i mają na celu inicjalizację pól x,y,z (w przypadku bezargumentowego konstruktora x, y, z mają być ustawiane na 0)
  - SprawdzXYZ() - publiczna metoda statyczna (static bool SprawdzXYZ(....) -> zwracająca wartości typu bool: true lub false), sprawdza czy trzy argumenty podane na wejście funkcji są > 0 oraz spełniają warunek boków trójkąta (czyli, że można z nich zbudować trójkąt), jeśli są poprawne to metoda zwraca true (return true), jeśli nie to false (return false)
  - UstawXYZ- publiczna metoda, która posiada 3 argumenty, służące do ustawienia pól

klasy x, y ,z, przed ustawieniem powinna być wywołana metoda SprawdzXYZ z nowymi wartościami, jeśli ta funkcja zwróci true to tylko wtedy nowe watości pola mogą być ustawiane, jeśli nie to to nowe wartości pól nie są ustawiane,

- ObliczObw2- prywatna metoda void, obliczająca połowę obwodu i zapisująca tą wartość w polu obw2,
- ObliczPole- publiczna metoda obliczająca i zwracająca pole trójkąta, zapisująca jego wartość w polu klasy o nazwie pole, najłatwiej ze wzoru Herona:

$$P=\sqrt[2]{obw2*(obw2-x)*(obw2-y)*(obw2-z)}$$ (jako pierwiastek można użyć funkcji sqrt z biblioteki math.h)

- ZwrócX – zwraca wartość x, (analogicznie dla y, z, alfa, beta, gamma).

➢ **Przykładowe rozwiązanie z pokazanym drugim sposobem na definiowanie metod poza klasą (sposób bardziej zalecany z punktu widzenia programistycznego, mniej zalecany do pisania na kartce na kolokwium → zajmuje więcej czasu i miejsca)**

Linia	Kod programu	Komentarz
1	`#include <iostream>`	• zadeklarowanie metod w klasie, definiowane są poza klasą
2	`#include <math.h>`	
3	`using namespace std;`	
4		
5	`class Trojkat`	
6	`{`	
7	`private:`	
8	`float x, y, z;`	
9	`float obw2;`	
10	`float pole;`	
11	`public:`	
12	`Trojkat();`	
13	`Trojkat(Trojkat& trojkat);`	
14	`static bool SprawdzXYZ(float _x, float _y,`	
15	`float _z);`	
	`void UstawXYZ(float _x, float _y, float _z);`	
16	`private:`	
17	`void ObliczObw2();`	
18	`public:`	
19	`float ObliczPole();`	
20	`float ZwrocX();`	
21	`// dokończyć samemu ZwrocY, ZwrocZ,`	
	`ZwrocPole, ZwrocObw2`	
22	`};`	
23	`int main()`	• utworzenie obiektu z wykorzystaniem konstruktora bezargumentowego • utworzenie obiektu z wykorzystaniem konstruktora kopiującego
24	`{`	
25	`Trojkat t1;`	
26	`t1.UstawXYZ(3,4,5);`	
27	`cout << "Pole trojkata wynosi: " <<`	
	`t1.ObliczPole() << endl;`	
28		
29	`t1.UstawXYZ(1,1,1);`	
30		
31	`Trojkat t2(t1);`	
32	`cout << "Pole trojkata wynosi: " <<`	
	`t2.ObliczPole() << endl;`	

```
33
34          system("pause");
35          return 0;
36  }
37
38  Trojkat::Trojkat()
39  {
40          x = y = z = obw2 = pole = 0;
41  }
42  Trojkat::Trojkat(Trojkat& t)
43  {
44          x = t.x;
45          y = t.y;
46          z = t.z;
47          obw2 = t.obw2;
48          pole = t.pole;
49  }
50  bool Trojkat::SprawdzXYZ(float _x, float _y, float
    _z)
51  {
52          if(_x > 0 && _y > 0 && _z > 0)
53          {
54                  if((_x + _y > _z) && (_y + _z > _x)
    && (_x + _z > _y))
55                  {
56                          return true;
57                  }
58          }
59          return false;
60  }
61  void Trojkat::UstawXYZ(float _x, float _y, float
    _z)
62  {
63          if(SprawdzXYZ(_x, _y, _z))
64          {
65                  x = _x;
66                  y = _y;
67                  z = _z;
68          }
69  }
70  void Trojkat::ObliczObw2()
71  {
72          obw2 = (x + y + z) / 2;
73  }
74  float Trojkat::ObliczPole()
75  {
76          ObliczObw2();
77          pole = sqrt(obw2*(obw2 – x) * (obw2 - y) *
    (obw2 - z));
78          return pole;
79  }
80  float Trojkat::ZwrocX()
81  {
82          return x;
83  }
```

- definicja metod poza klasą
- do nagłówka metody definiowanej poza klasą dodawana jest nazwa klasy oraz podwójny dwukorpek :: przed nazwą metody
- ciało metody definiowanej poza klasą jest takie samo jak w przypadku, gdy metoda była definiowana wewnątrz klasy
- **konstruktor kopiujący (obiekt klasy trójkąt może posłużyć do utworzenia nowego obiektu tej klasy)**
- warunek trójkąta

Zamień w klasie metodę:

- ObliczObw2 na ObliczObw tak, metoda miała etykietę public i zwracała na wyjściu wartość obliczonego obwodu, pamiętaj o koniecznych zmianach w metodzie ObliczPole
- Dodaj do klasy Trojkat metodę DrukujPole tak, aby zamiast linijki:
  `cout << "Pole trojkata wynosi: " << t2.ObliczPole() << endl;`
  używać linijki: t2.DrukujPole();
- Dodaj do klasy trójkąt alfa, beta, gamma – pola prywatne oznaczające kąty trojkąta oraz

metodę:
ObliczKaty- publiczna metoda obliczająca kąty trójkąta – pola klasy: alfa, beta i gamma, wiedząc, że pole trójkąta można obliczyć ze wzoru → 0.5 razy dwa dowolne boki trojkąta i sinus kąta między nimi zawarty np. 0.5 * y * z * sin(alfa) – mając pole trojkąta, y oraz z (ich wartości są w polach klasy) można obliczyć wartość sin(alfa) i przypisać np. do zmiennej lokalnej sinAlfa, następnie funkcja asin(sinAlfa) (arcus sinus jest zdefiniowana w bibliotece math.h) zwróci wartość kąta alfa w radianach np. dla trójkąta równobocznego(wszystkie kąty 60 st.,). asin(sinAlfa) zwróci ok. 1.0472.

- Dodaj metodę ZwrocAlfa, ZwrocBeta, ZwrocGama.

5. Napisz klasę trójkąt prostokątny o nazwie *TrojkatPr.*

➢ **Klasa posiada następujące pola i metody:**

a) pola prywatne**:**

- ***a, b*** → pole oznaczające boki trójkąta typu zmiennoprzecinkowego (są to przyprostokątne),
- ***poleTr*** → pole oznaczające wartość pola trójkąta typu zmiennoprzecinkowego,

b) metody publiczne**:**

- **konstruktor dwuargumentowy:** argumenty ustawiają pola ***a*** i ***b,*** jeśli dany argument będzie **< 0** to bok mu odpowiadający ma być ustawiony na **0, *poleTr*** ustawiane jest na **0,**
- **gettery** i **settery** dla pól ***a*** i ***b*** → ***setter*** ma zmieniać wartość boku tylko jeśli jego argument będzie **> *0,***
- **getter** pola ***poleTr,***
- ***ObliczPole*** → metoda bezargumentowa zapisującą wartość pola trójkąta do pola ***poleTr*** oraz zwracająca wartość pola na wyjściu ze wzoru: **½(ab),**
- ***Wypisz*** → wypisującą tekst: ”Trójkąt prostkątny o bokach: ” … ” i ”...” posiada wartość pola wynoszącą ” …

  W miejsce kropek należy wstawić odpowiednie pola lub metody.

➢ **Napisz funkcję główną oraz dołącz odpowiednie biblioteki, w której wykonaj następujące kroki z użyciem utworzonych klas w zad. 1 i 2 (zakłada się, że klasy z zad. 1 i 2 są umiejscowione przed funkcją main, więc nie trzeba już się tym przejmować):**

- Utwórz obiekt klasy ***TrojkatPr***,
- Oblicz pole trójkąta i je wypisz w następujący sposób:
- ”Pole trójkąta wynosi” ...
- Zmień długość jednego boku trójkąta,
- Oblicz pole trójkąta,
- Użyj metody ***Wypisz***.

6. Pytania na wejściówkę:

1. Napisz definicję klasy (bez funkcji main i includowania bibliotek) o nazwie A, która posiada:

- pole o nazwie a1 typu całkowitego (etykieta public)
- pole o nazwie a2 typu float (etykieta private)
- konstruktor bez argumentów (ustawia a1 i a2 na 0) → zdefiniować wewnątrz klasy
- konstruktor z dwoma parametrami do ustawiania a1 i a2 → zdefiniować wewnątrz klasy

2. Napisz definicję klasy o nazwie B, która posiada:

- pole o nazwie b1 typu string (etykieta public)
- pole o nazwie b2 typu string (etykieta protected)
- konstruktor bezargumentowy, ustawia oba pola na pusty tekst → ""
- Odpowiedz jednym słowem (tak lub nie) czy możliwy jest zapis poza klasą w funkcji main (jeśli obiektB jest instancją klasy B):

```
obiektB.b1 = "roman";
```

- Odpowiedz jednym słowem (tak lub nie) czy możliwy jest zapis poza klasą w funkcji main (jeśli obiektB jest instancją klasy B):

```
obiektB.b2 = "roman";
```

3. Dla poniższej klasy C uzupełnij konstruktory oraz napisz dwa przykładowe utworzenia obiektu klasy C (dla każdego konstruktora):

```
class C
{
public:
	C(int _b1, int _b2)
	{
		//inicjalizacja pól klasy od podania parametrów
	}
	C()
	{
		//inicjalizacja pól klasy na 0.
	}
private:
	int b1;
	int b2;

};
```

4. Napisz 4 podstawowe zasady dotyczące konstruktora (zwracana wartość, nazwa konstruktora, ile konstruktorów, kiedy jest wywoływany, jaką pełni funkcję):

a) …

b) ...

c) ...

d) …

e) ...

5. Napisz jednym zdaniem:
   - czym różni się obiekt od klasy,
   - co to jest metoda klasy,
   - co to jest etykieta klasy,
   - co to jest pole klasy.