Obiekty dynamiczne, klasy i funkcje zaprzyjaźnione, przeciążanie operatorów

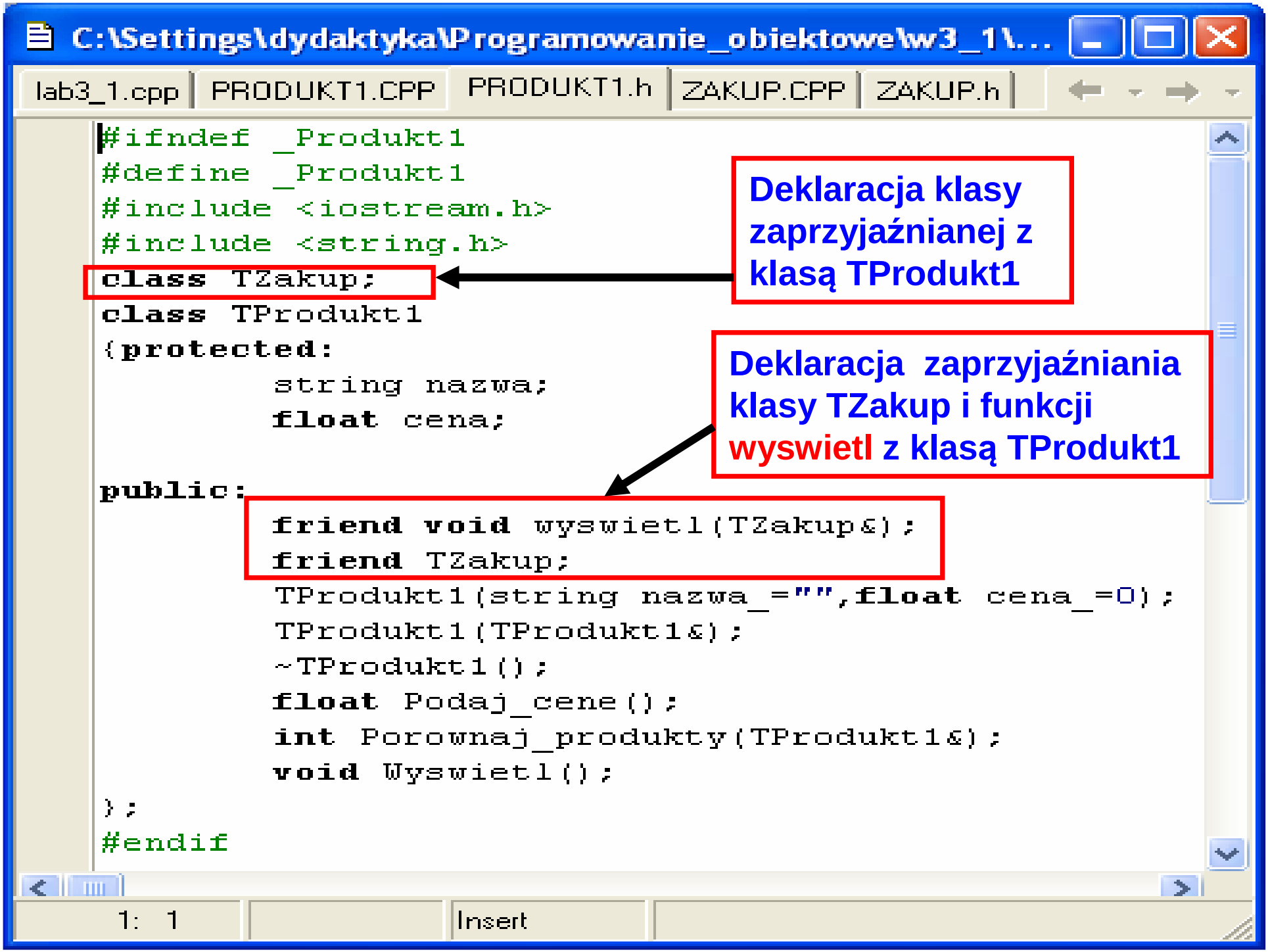
1. Klasy zaprzyjaźnione i funkcje zaprzyjaźnione
2. Przeciążanie operatorów
3. Obiekty dynamiczne
4. Obiekty dynamiczne jako składowe klas

Obiekty dynamiczne, klasy i funkcje zaprzyjaźnione, przeciążanie operatorów

1. Klasy zaprzyjaźnione i funkcje zaprzyjaźnione

Klasy zaprzyjaźnione i funkcje zaprzyjaźnione

1. Klasy zaprzyjaźnione mają dostęp do składowych prywatnych i chronionych klasy, z którą jest zaprzyjaźniona tzn. gdzie wystąpiła deklaracja zaprzyjaźnienia danej klasy z inną.
2. Niezależna *funkcja zaprzyjaźniona* z jedną klasą ma dostęp do składowych prywatnych i chronionych klasy. Sama funkcja nie nabywa własności składowej klasy.
3. Niezależna funkcja zaprzyjaźniona z kilkoma klasami ma dostęp do wszystkich składowych prywatnych i chronionych tych klas. Sama funkcja nie nabywa jednak własności składowej tych klas. Deklaracja zaprzyjaźnienia powinna wystąpić w każdej z tych klas, natomiast definicja tylko raz.

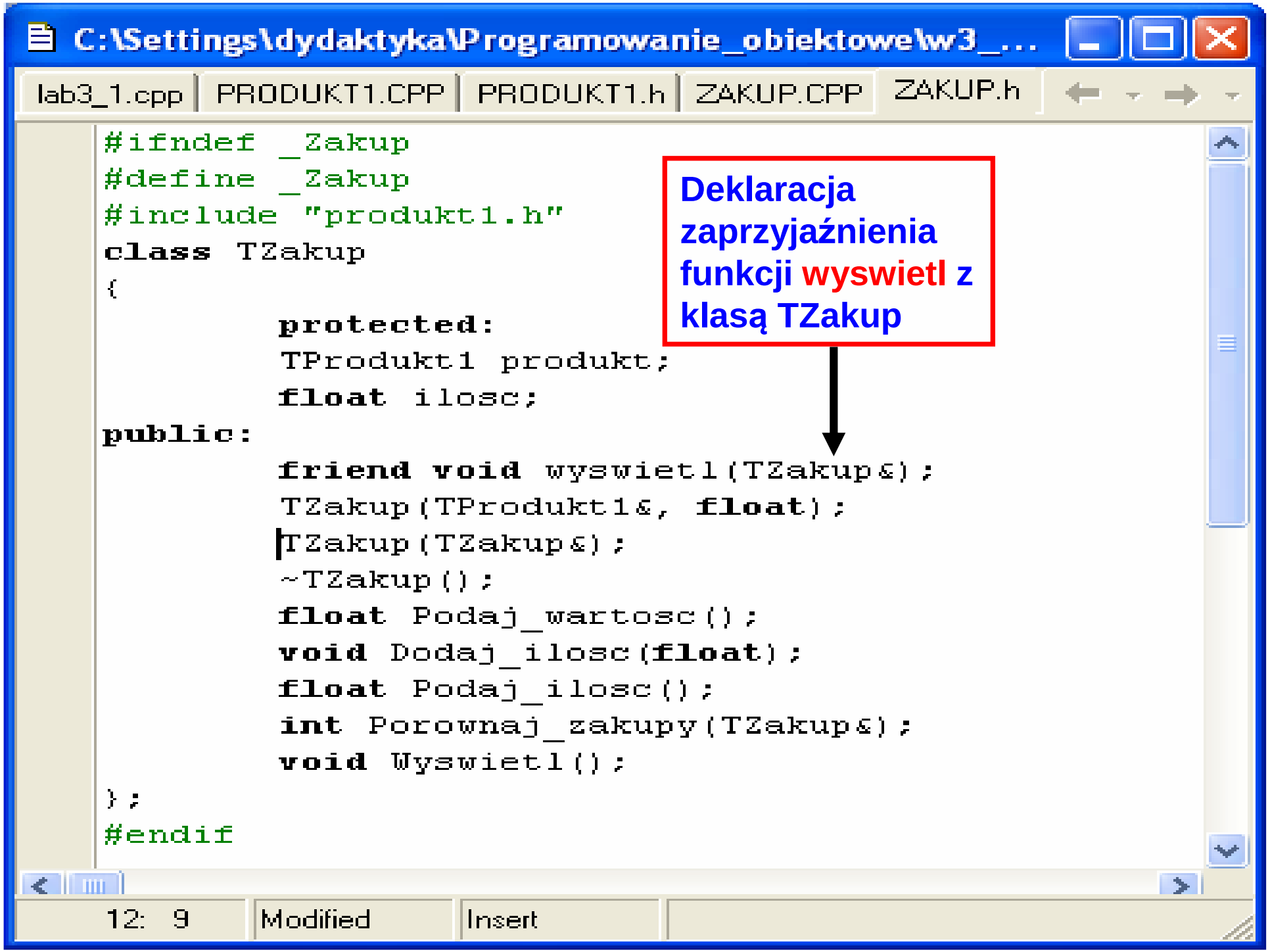

C:\Settings\dydaktyka\Programowanie_obiektowe\w3_1\...
lab3_1.cpp
PRODUKT1.CPP
PRODUKT1.h
ZAKUP.CPP
ZAKUP.h
#ifndef _Produkt1
#define _Produkt1
#include <iostream.h>
#include <string.h>
class TZakup;
class TProdukt1
{protected:
string nazwa;
float cena;
public:
friend void wyswietl(TZakup&);
friend TZakup;
TProdukt1(string nazwa_="",float cena_=0);
TProdukt1(TProdukt1&);
~TProdukt1();
float Podaj_cene();
int Porownaj_produkty(TProdukt1&);
void Wyswietl();
};
#endif
Deklaracja klasy zaprzyjaźnianej z klasą TProdukt1
Deklaracja zaprzyjaźniania klasy TZakup i funkcji wyswietl z klasą TProdukt1
1: 1
Insert

C:\Settings\dydaktyka\Programowanie_obiektowe\w3_1\PRODUKT1.CPP

lab3_1.cpp | PRODUKT1.CPP | PRODUKT1.h | ZAKUP.CPP | ZAKUP.h

```
#include "produkt1.h"
TProdukt1::TProdukt1(string nazwa_,float cena_)
 { cout<<"Konstruktor zwykly z parametrami klasy TProdukt1"<<endl;
   nazwa=nazwa_;
   cena=cena_;}
TProdukt1::TProdukt1(TProdukt1& p)
 { cout<<"Konstruktor kopiujacy klasy TProdukt1"<<endl;
   nazwa = p.nazwa;
   cena=p.cena; }
TProdukt1::~TProdukt1()
 { cout<<"Destruktor klasy TProdukt1"<<endl; }
float TProdukt1::Podaj_cene()
 { return cena; }
int TProdukt1::Porownaj_produkty(TProdukt1& p)
 { return nazwa==p.nazwa && Podaj_cene()==p.Podaj_cene(); }
void TProdukt1::Wyswietl()
 { cout<<"Cena produktu: "<<cena<<", Nazwa produktu: "<<nazwa<<endl; }
```

1: 1 | Insert


C:\Settings\dydaktyka\Programowanie_obiektowe\w3_...
lab3_1.cpp
PRODUKT1.CPP
PRODUKT1.h
ZAKUP.CPP
ZAKUP.h
#ifndef _Zakup
#define _Zakup
#include "produkt1.h"
class TZakup
{
protected:
TProdukt1 produkt;
float ilosc;
public:
friend void wyswietl(TZakup&);
TZakup(TProdukt1&, float);
TZakup(TZakup&);
~TZakup();
float Podaj_wartosc();
void Dodaj_ilosc(float);
float Podaj_ilosc();
int Porownaj_zakupy(TZakup&);
void Wyswietl();
};
#endif
Deklaracja zaprzyjaźnienia funkcji wyswietl z klasą TZakup
12: 9
Modified
Insert

C:\Settings\dydaktyka\Programowanie_obiektowe\w3_1\ZAKUP.CPP

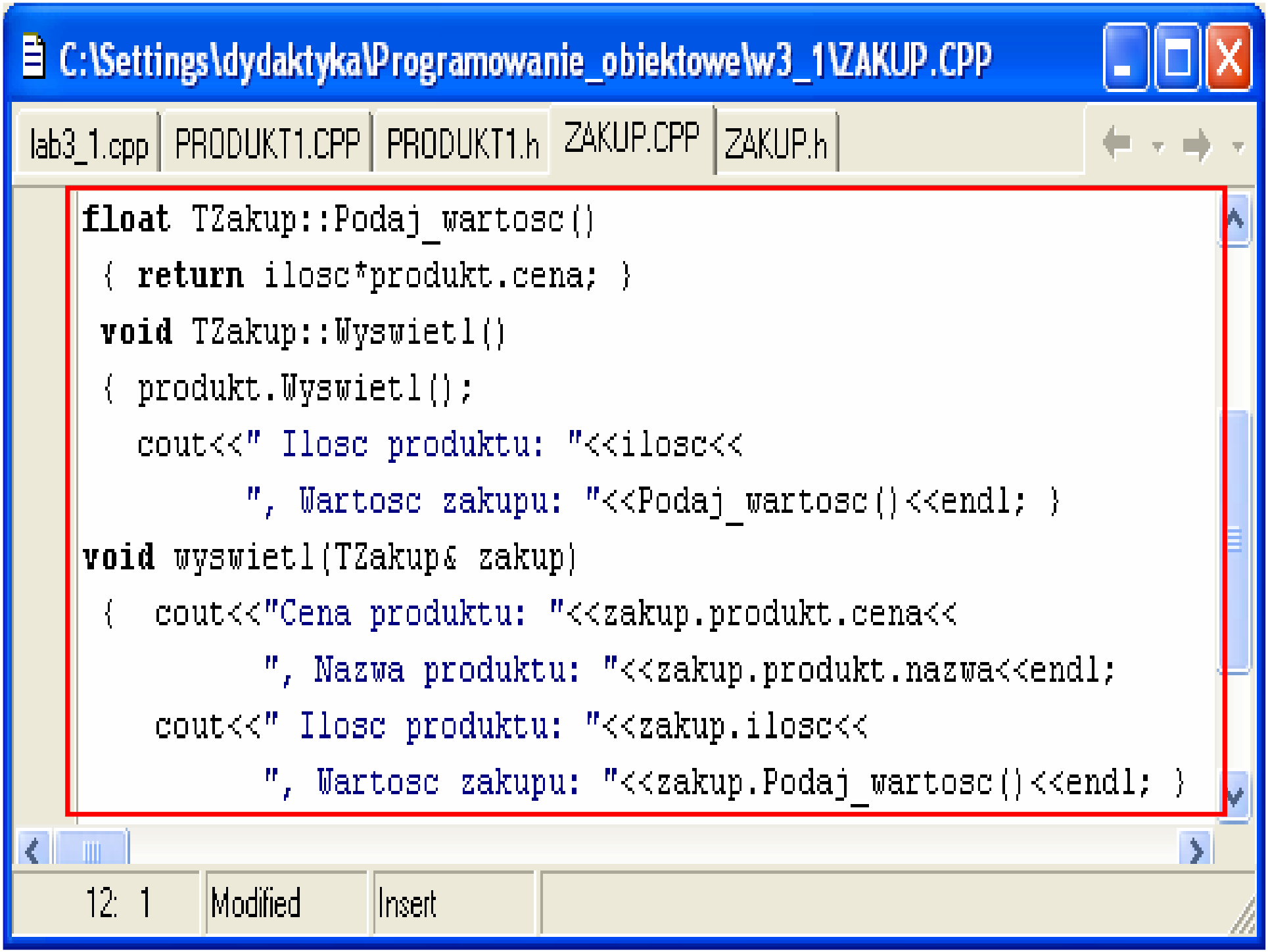
lab3_1.cpp | PRODUKT1.CPP | PRODUKT1.h | ZAKUP.CPP | ZAKUP.h

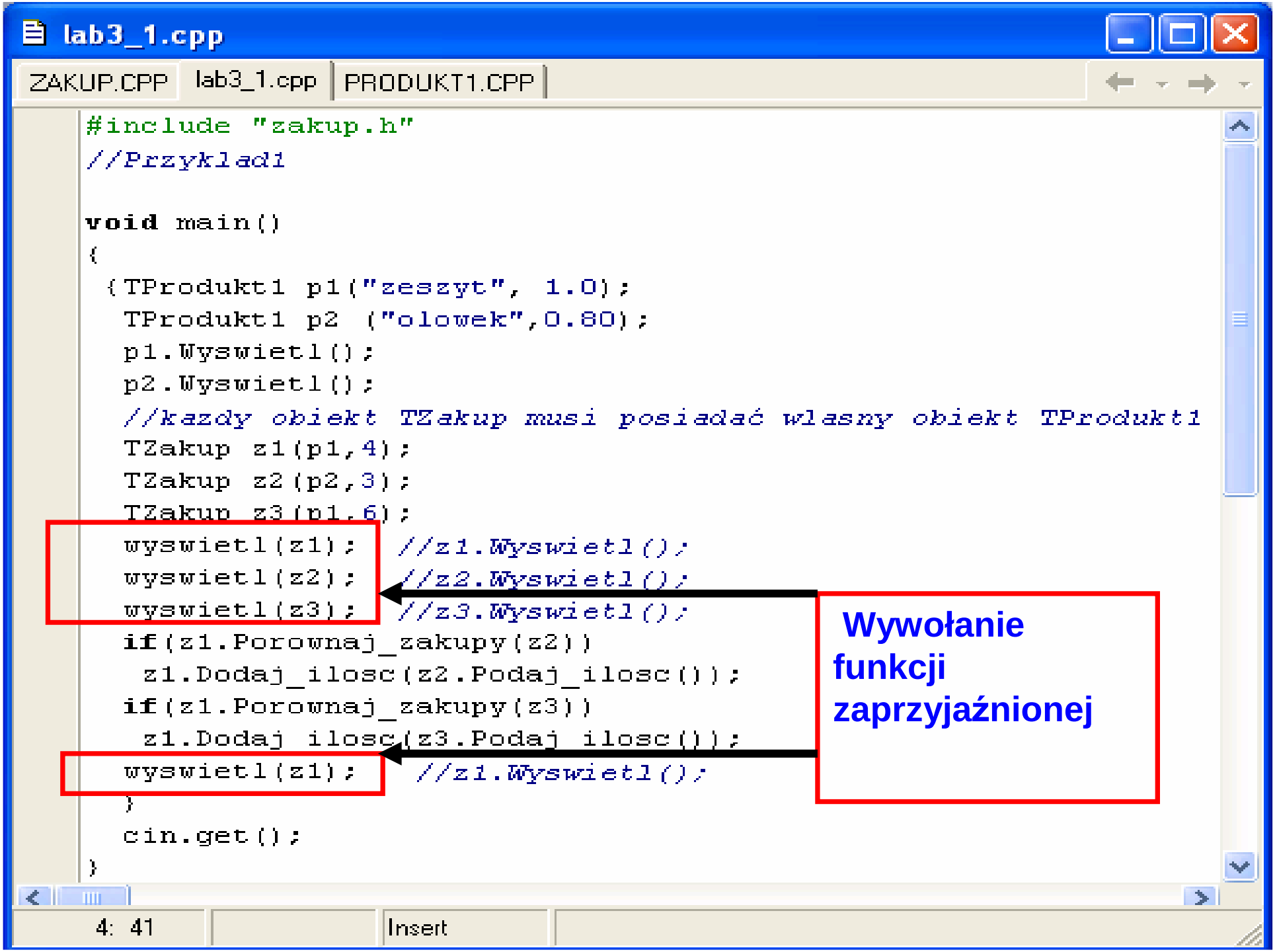
```
#include "zakup.h"
TZakup::TZakup(TProdukt1& produkt_,float ilosc_)
 { cout<<"Konstruktor zwykly z parametrami klasy TZakup"<<endl;
   produkt=produkt_;
   ilosc=ilosc_; }
TZakup::TZakup(TZakup& p)
 { cout<<"Konstruktor kopiujacy klasy TZakup"<<endl;
   produkt=p.produkt;
   ilosc=p.ilosc; }
TZakup::~TZakup()
 { cout<<"Destruktor klasy TZakup"<<endl; }
float TZakup::Podaj_wartosc()
 { return ilosc*produkt.cena; }
void TZakup::Dodaj_ilosc(float ile)
 { ilosc+=ile; }
float TZakup::Podaj_ilosc()
 { return ilosc; }
int TZakup::Porownaj_zakupy(TZakup& zakup)
 { return  produkt.Porownaj_produkty(zakup.produkt); }
void TZakup::Wyswietl()
 { produkt.Wyswietl();
   cout<<" Ilosc produktu: "<<ilosc<<
         ", Wartosc zakupu: "<<Podaj_wartosc()<<endl; }
void wyswietl(TZakup& zakup)
 {  cout<<"Cena produktu: "<<zakup.produkt.cena<<
          ", Nazwa produktu: "<<zakup.produkt.nazwa<<endl;
    cout<<" Ilosc produktu: "<<zakup.ilosc<<
          ", Wartosc zakupu: "<<zakup.Podaj_wartosc()<<endl; }
```

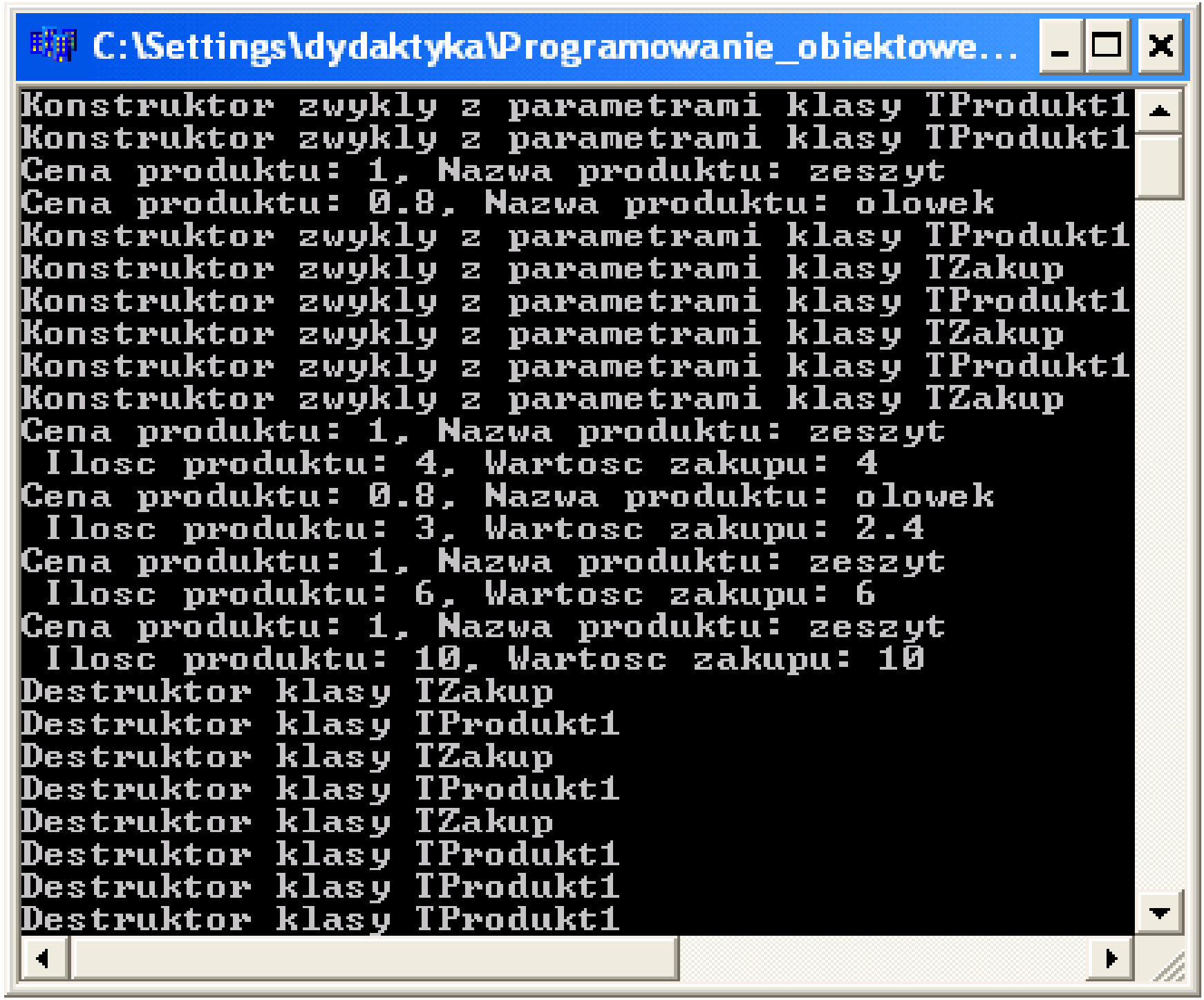
Klasa TZakup ma bezpośredni dostęp do składowych chronionych klasy TProdukt1

Definicja funkcji wyswietl zaprzyjaźnionej z klasami TProdukt i TZakup

20: 1 | Modified | Insert


C:\Settings\dydaktyka\Programowanie_obiektowe\w3_1\ZAKUP.CPP
lab3_1.cpp
PRODUKT1.CPP
PRODUKT1.h
ZAKUP.CPP
ZAKUP.h
float TZakup::Podaj_wartosc()
{ return ilosc*produkt.cena; }
void TZakup::Wyswietl()
{ produkt.Wyswietl();
cout<<" Ilosc produktu: "<<ilosc<<
", Wartosc zakupu: "<<Podaj_wartosc()<<endl; }
void wyswietl(TZakup& zakup)
{ cout<<"Cena produktu: "<<zakup.produkt.cena<<
", Nazwa produktu: "<<zakup.produkt.nazwa<<endl;
cout<<" Ilosc produktu: "<<zakup.ilosc<<
", Wartosc zakupu: "<<zakup.Podaj_wartosc()<<endl; }
12: 1
Modified
Insert


lab3_1.cpp
ZAKUP.CPP
lab3_1.cpp
PRODUKT1.CPP
#include "zakup.h"
//Przyklad1
void main()
{
{TProdukt1 p1("zeszyt", 1.0);
TProdukt1 p2 ("olowek",0.80);
p1.Wyswietl();
p2.Wyswietl();
//kazdy obiekt TZakup musi posiadać wlasny obiekt TProdukt1
TZakup z1(p1,4);
TZakup z2(p2,3);
TZakup z3(p1,6);
wyswietl(z1); //z1.Wyswietl();
wyswietl(z2); //z2.Wyswietl();
wyswietl(z3); //z3.Wyswietl();
if(z1.Porownaj_zakupy(z2))
z1.Dodaj_ilosc(z2.Podaj_ilosc());
if(z1.Porownaj_zakupy(z3))
z1.Dodaj_ilosc(z3.Podaj_ilosc());
wyswietl(z1); //z1.Wyswietl();
}
cin.get();
}
Wywołanie funkcji zaprzyjaźnionej
4: 41
Insert


C:\Settings\dydaktyka\Programowanie_obiektowe...
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TProdukt1
Cena produktu: 1, Nazwa produktu: zeszyt
Cena produktu: 0.8, Nazwa produktu: olowek
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TZakup
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TZakup
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TZakup
Cena produktu: 1, Nazwa produktu: zeszyt
Ilosc produktu: 4, Wartosc zakupu: 4
Cena produktu: 0.8, Nazwa produktu: olowek
Ilosc produktu: 3, Wartosc zakupu: 2.4
Cena produktu: 1, Nazwa produktu: zeszyt
Ilosc produktu: 6, Wartosc zakupu: 6
Cena produktu: 1, Nazwa produktu: zeszyt
Ilosc produktu: 10, Wartosc zakupu: 10
Destruktor klasy TZakup
Destruktor klasy TProdukt1
Destruktor klasy TZakup
Destruktor klasy TProdukt1
Destruktor klasy TZakup
Destruktor klasy TProdukt1
Destruktor klasy TProdukt1
Destruktor klasy TProdukt1

Obiekty dynamiczne, klasy i funkcje zaprzyjaźnione, przeciążanie operatorów

1. **Klasy zaprzyjaźnione i funkcje zaprzyjaźnione**
2. **Przeciążanie operatorów**

Nie przeciąża się operatorów: **. .* :: ?: sizeof**
Przeciążanie operatorów za pomocą funkcji

operator op (*lista_argumentów*)

1. jako niezależnej funkcji, zaprzyjaźnionej z jedną lub kilkoma klasami
2. jako metody - jeden z argumentów jest niejawny (***this***)
3. w przypadku przeciążania operatorów dwuargumentowych nie można wykorzystać metody, lecz funkcji zaprzyjaźnionej, jeżeli lewy argument nie należy do klasy, dla której jest przeciążany operator. Przykładem jest operator wyjścia <<
4. należy przeciążać istniejące operatory przy zachowaniu odpowiedniej liczby argumentów. Tak przeciążone operatory zachowują też typową dla nich łączność i priorytet, lecz ich znaczenie może być inne niż zwyczajowe
5. przeciążony operator musi ***mieć obiekt jako jeden z argumentów***, stąd nie można zmienić znaczenia operatorów dla standardowych typów

Liczba argumentów	Operatory	Łączność
2	()[2] [][2] →[2] →*	lewostronna
1	+ - ++[4] --[4] ! ~ * &[1,2] new[3] delete[3] (typ)	prawostronna
2	* / %	lewostronna
2	+ -	lewostronna
2	<< >>	lewostronna
2	< <= > >=	lewostronna
2	= = !=	lewostronna
2	&	lewostronna
2	∧	lewostronna
2	‖	lewostronna
2	&&	lewostronna
2	|	lewostronna
2	=[1, 2] += -= *= /= %= ^= &= |= <<= >>=	prawostronna
2	,[1]	lewostronna

1. Jeżeli nie jest przeciążony, ma znaczenie domniemane
2. Musi być zdefiniowany jako metoda niestatyczna
3. Mogą mieć znaczenie globalne, natomiast przeciążone na rzecz klasy muszą być metodami typu **static** (deklarowane również niejawnie) o następujących prototypach:
   void * new (size_t)
   //zwraca adres przydzielonego obiektu
   void delete (typ *)
   *//funkcja zwalnia obiekt o adresie typ**
   Operatory domniemane są osiągalne za pomocą odwołań: **::new**, **::delete**
4. Przyrostkowe operatory definiuje się z jednym fikcyjnym argumentem **int.**

C:\Settings\dydaktyka\Programowanie_obiektowe\w3_2\PRODUKT1.h

ZAKUP.CPP | ZAKUP.h | lab3_2.cpp | PRODUKT1.CPP | PRODUKT1.h

```
#ifndef _Produkt1
#define _Produkt1
#include <iostream.h>
#include <string.h>
class TProdukt1
{protected:
        string nazwa;
        float cena;

public:
        TProdukt1(string nazwa_="",float cena_=0);
        TProdukt1(TProdukt1&);
        ~TProdukt1();
        float Podaj cene();
        int operator==(TProdukt1&);
        friend ostream& operator<< (ostream &, TProdukt1 &);
};
#endif
```

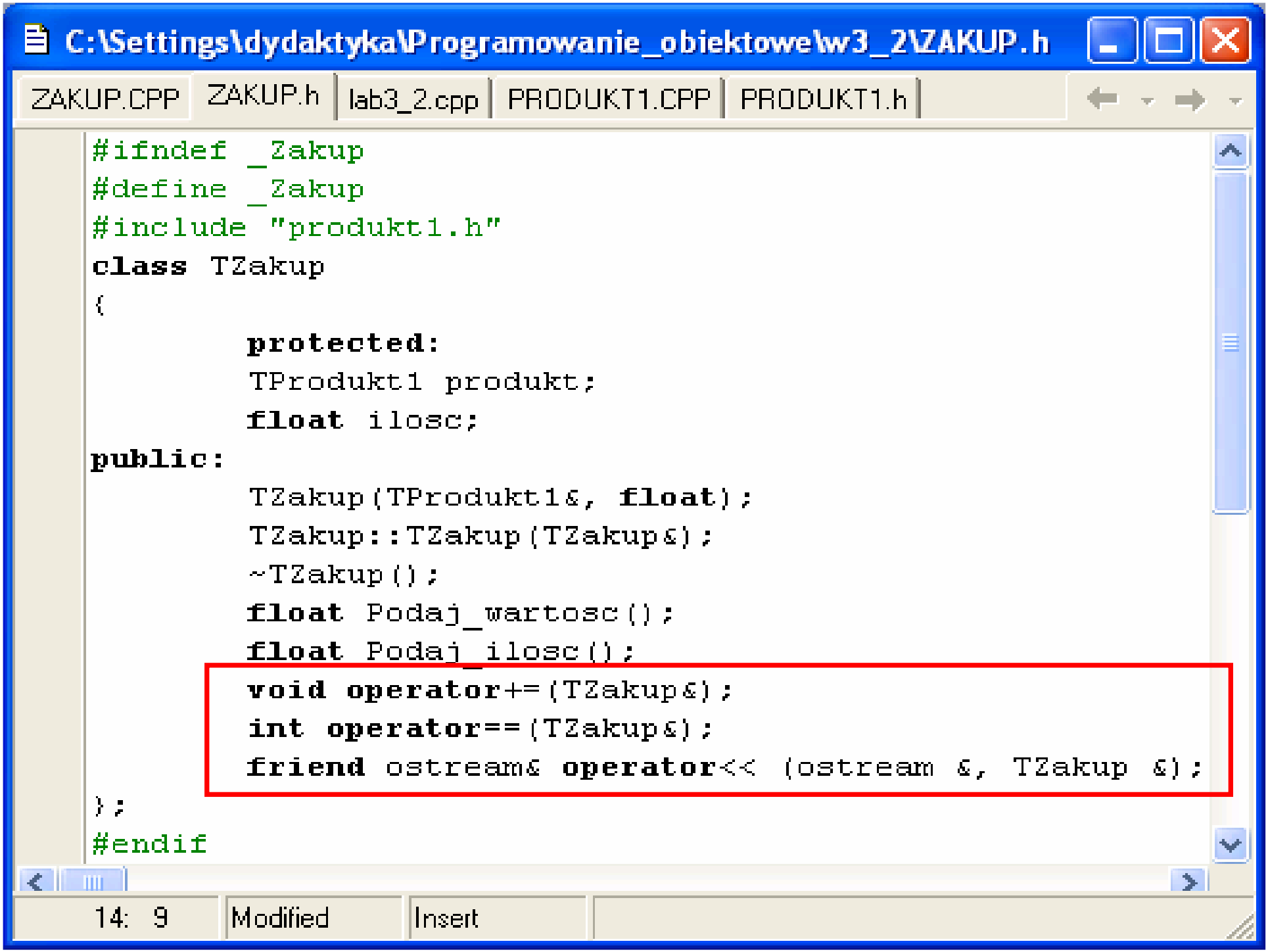
13: 73 | Modified | Insert

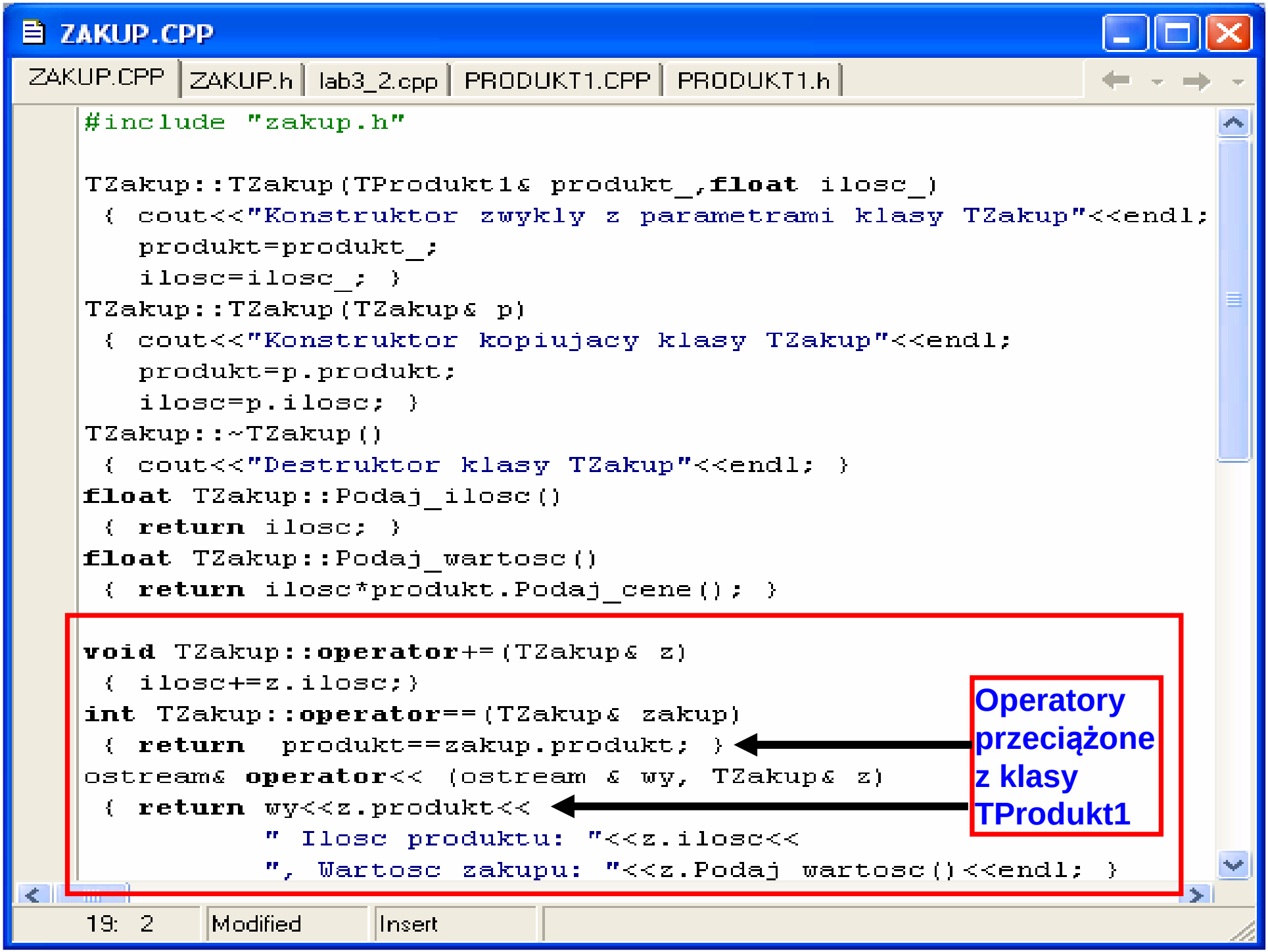
PRODUKT1.CPP

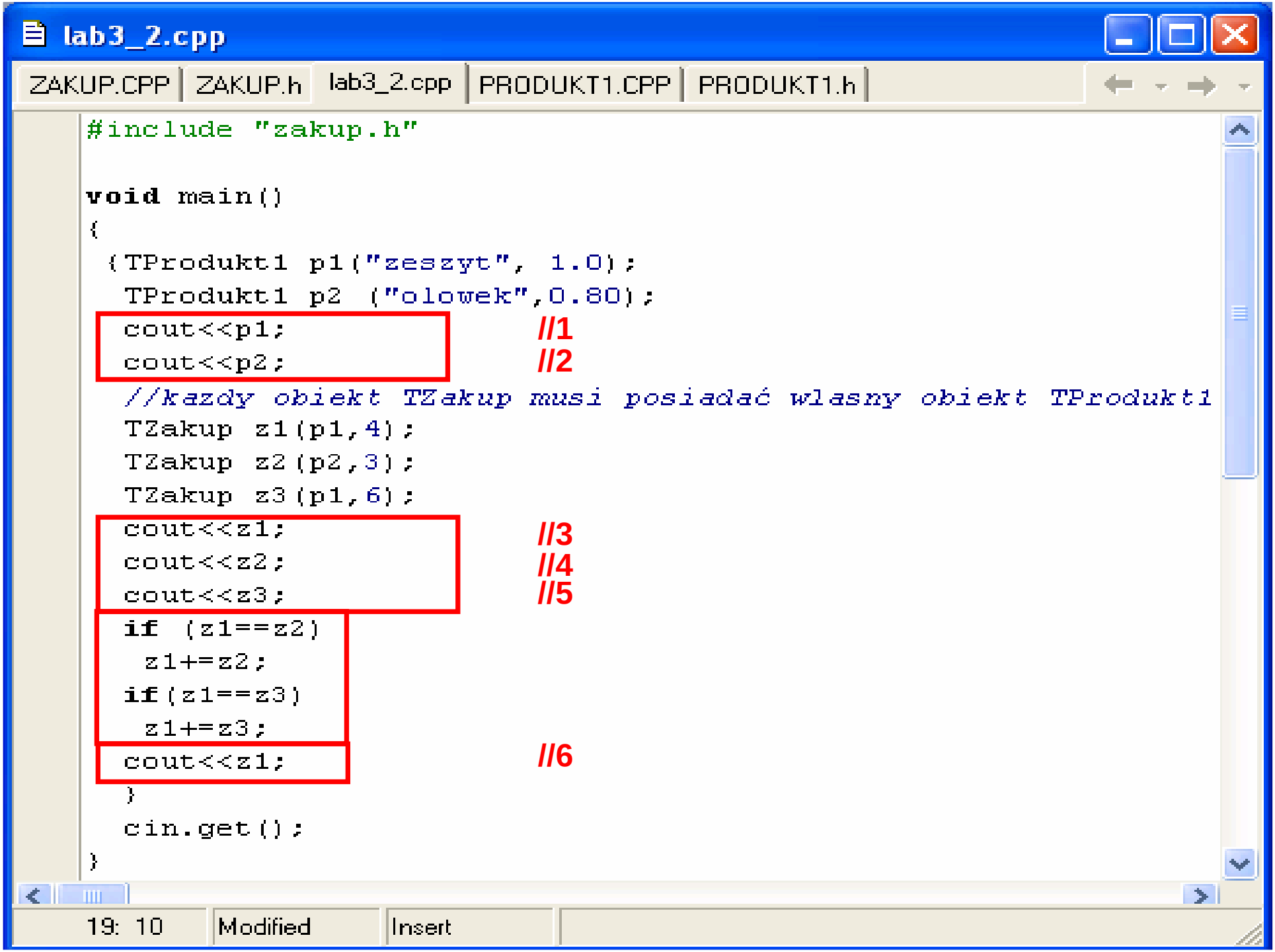
ZAKUP.CPP | ZAKUP.h | lab3_2.cpp | PRODUKT1.CPP | PRODUKT1.h

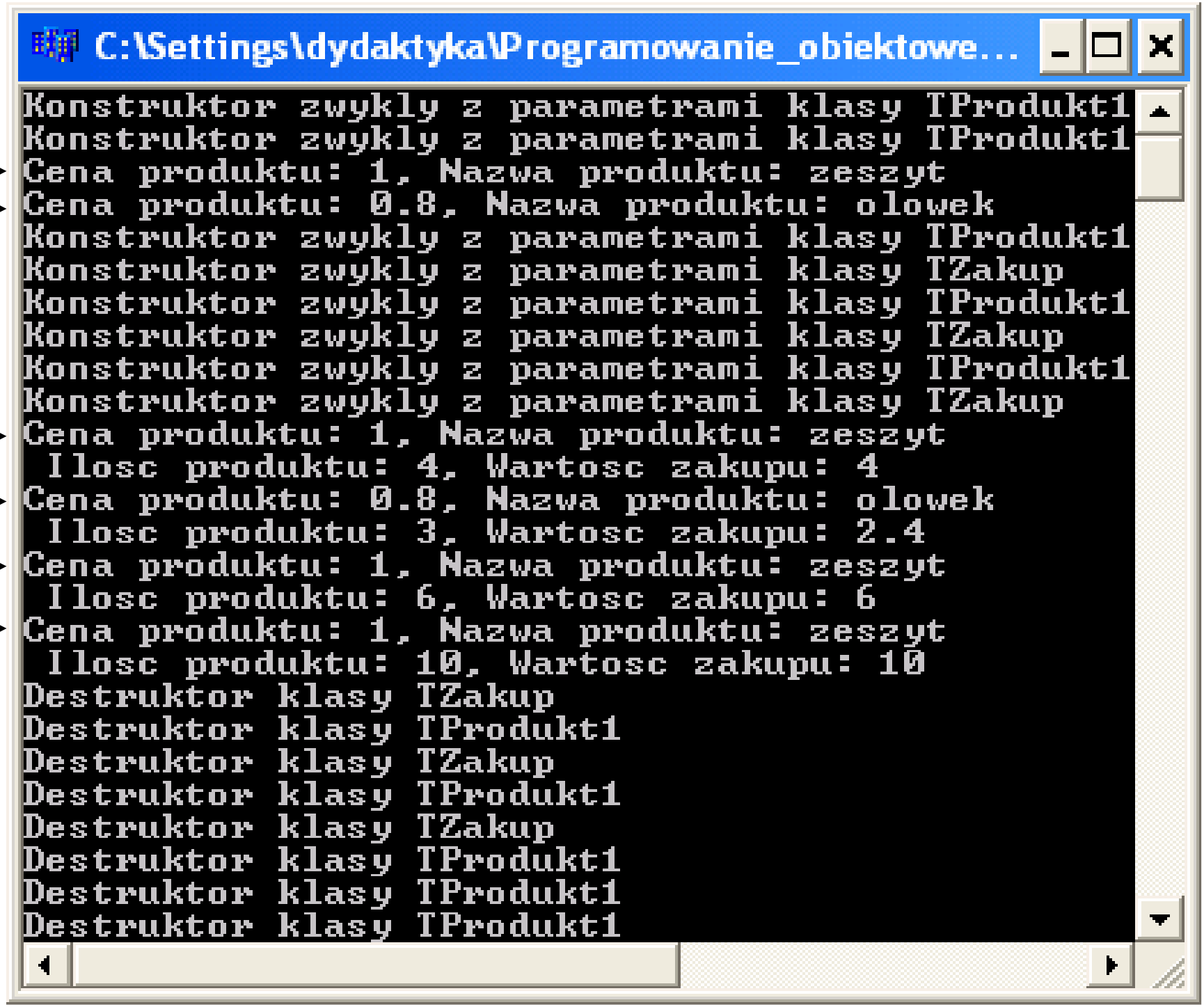
```
#include "produkt1.h"
TProdukt1::TProdukt1(string nazwa_,float cena_)
 { cout<<"Konstruktor zwykly z parametrami klasy TProdukt1"<<endl;
   nazwa=nazwa_;
   cena=cena_;}
TProdukt1::TProdukt1(TProdukt1& p)
 { cout<<"Konstruktor kopiujacy klasy TProdukt1"<<endl;
   nazwa = p.nazwa;
   cena=p.cena; }
TProdukt1::~TProdukt1()
 { cout<<"Destruktor klasy TProdukt1"<<endl; }
float TProdukt1::Podaj_cene()
 { return cena; }

int TProdukt1::operator==(TProdukt1& p)
 { return nazwa==p.nazwa && Podaj_cene()==p.Podaj_cene(); }

ostream& operator<< (ostream & wy, TProdukt1 & p)
 { return wy<<"Cena produktu: "<<p.cena<<
              ", Nazwa produktu: "<<p.nazwa<<endl; }
```

14: 2 | Modified | Insert


C:\Settings\dydaktyka\Programowanie_obiektowe\w3_2\ZAKUP.h
ZAKUP.CPP
ZAKUP.h
lab3_2.cpp
PRODUKT1.CPP
PRODUKT1.h
#ifndef _Zakup
#define _Zakup
#include "produkt1.h"
class TZakup
{
protected:
TProdukt1 produkt;
float ilosc;
public:
TZakup(TProdukt1&, float);
TZakup::TZakup(TZakup&);
~TZakup();
float Podaj_wartosc();
float Podaj_ilosc();
void operator+=(TZakup&);
int operator==(TZakup&);
friend ostream& operator<< (ostream &, TZakup &);
};
#endif
14: 9
Modified
Insert

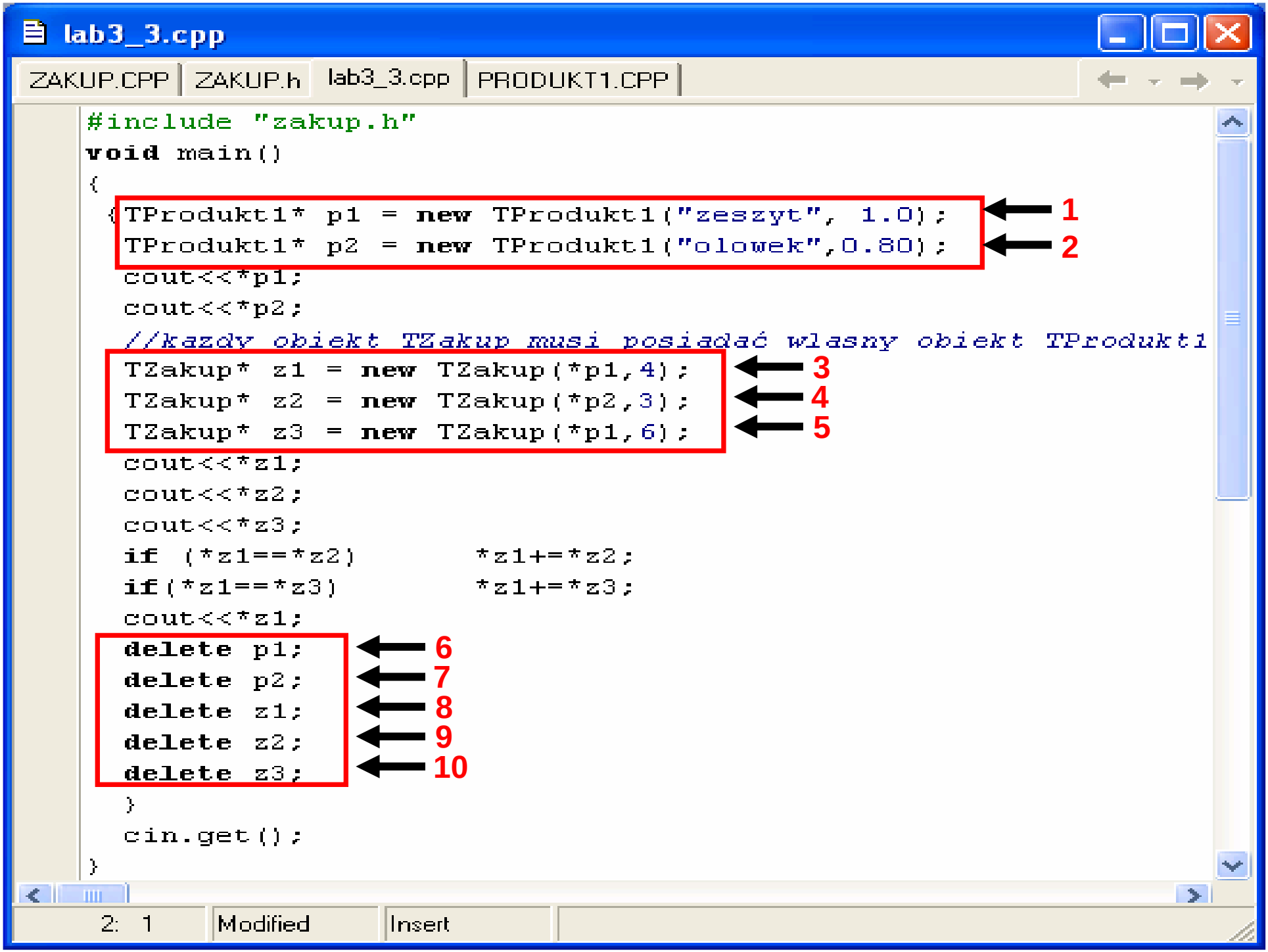

ZAKUP.CPP
ZAKUP.CPP
ZAKUP.h
lab3_2.cpp
PRODUKT1.CPP
PRODUKT1.h
#include "zakup.h"
TZakup::TZakup(TProdukt1& produkt_,float ilosc_)
{ cout<<"Konstruktor zwykly z parametrami klasy TZakup"<<endl;
produkt=produkt_;
ilosc=ilosc_; }
TZakup::TZakup(TZakup& p)
{ cout<<"Konstruktor kopiujacy klasy TZakup"<<endl;
produkt=p.produkt;
ilosc=p.ilosc; }
TZakup::~TZakup()
{ cout<<"Destruktor klasy TZakup"<<endl; }
float TZakup::Podaj_ilosc()
{ return ilosc; }
float TZakup::Podaj_wartosc()
{ return ilosc*produkt.Podaj_cene(); }
void TZakup::operator+=(TZakup& z)
{ ilosc+=z.ilosc;}
int TZakup::operator==(TZakup& zakup)
{ return produkt==zakup.produkt; }
ostream& operator<< (ostream & wy, TZakup& z)
{ return wy<<z.produkt<<
" Ilosc produktu: "<<z.ilosc<<
", Wartosc zakupu: "<<z.Podaj_wartosc()<<endl; }
Operatory przeciążone z klasy TProdukt1
19: 2
Modified
Insert

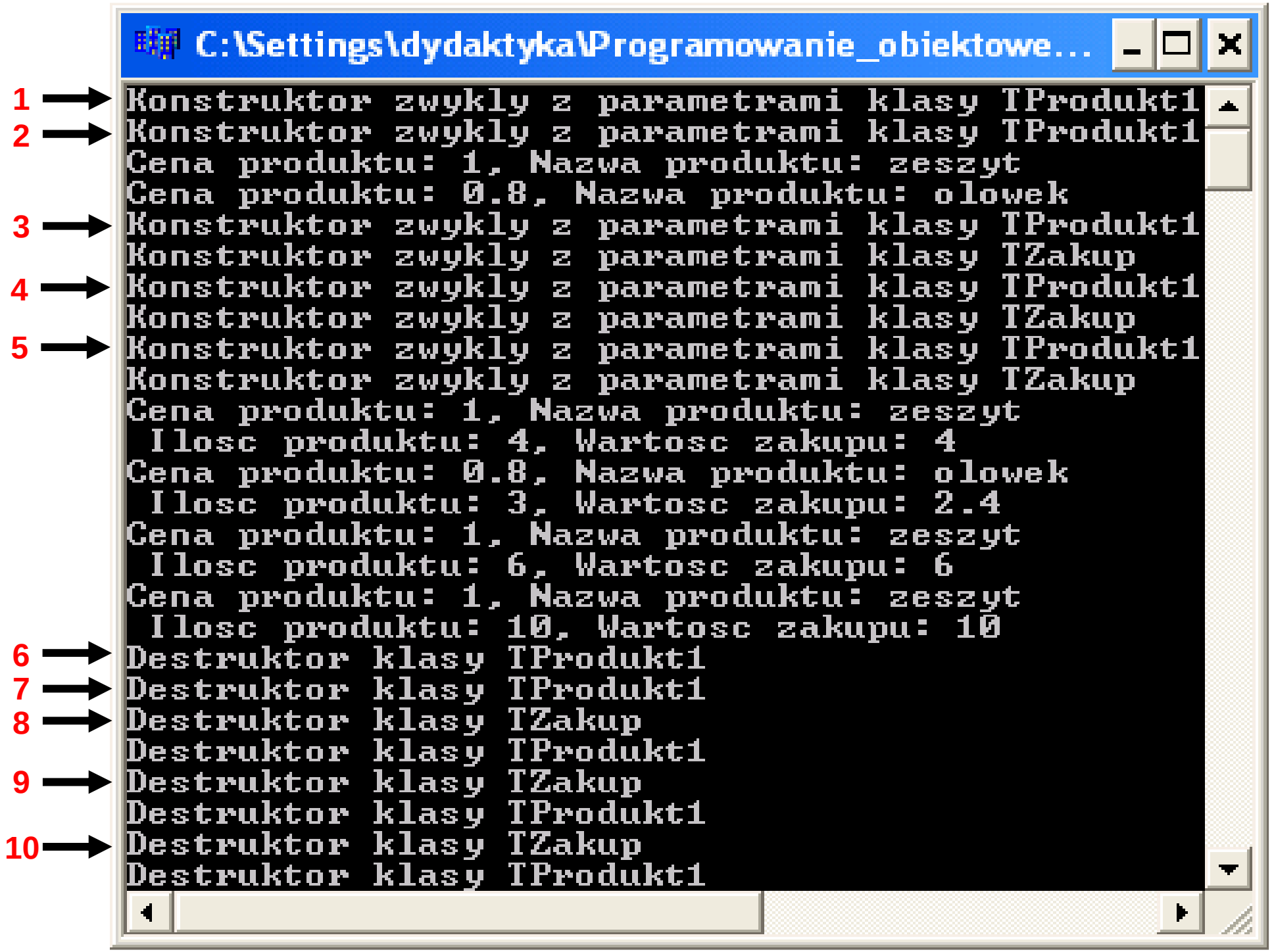

lab3_2.cpp
ZAKUP.CPP
ZAKUP.h
lab3_2.cpp
PRODUKT1.CPP
PRODUKT1.h
#include "zakup.h"
void main()
{
{TProdukt1 p1("zeszyt", 1.0);
TProdukt1 p2 ("olowek",0.80);
cout<<p1;
cout<<p2;
//kazdy obiekt TZakup musi posiadać wlasny obiekt TProdukt1
TZakup z1(p1,4);
TZakup z2(p2,3);
TZakup z3(p1,6);
cout<<z1;
cout<<z2;
cout<<z3;
if (z1==z2)
z1+=z2;
if(z1==z3)
z1+=z3;
cout<<z1;
}
cin.get();
}
//1
//2
//3
//4
//5
//6
19: 10
Modified
Insert


C:\Settings\dydaktyka\Programowanie_obiektowe...
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TProdukt1
1
Cena produktu: 1, Nazwa produktu: zeszyt
2
Cena produktu: 0.8, Nazwa produktu: olowek
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TZakup
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TZakup
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TZakup
3
Cena produktu: 1, Nazwa produktu: zeszyt
Ilosc produktu: 4, Wartosc zakupu: 4
4
Cena produktu: 0.8, Nazwa produktu: olowek
Ilosc produktu: 3, Wartosc zakupu: 2.4
5
Cena produktu: 1, Nazwa produktu: zeszyt
Ilosc produktu: 6, Wartosc zakupu: 6
6
Cena produktu: 1, Nazwa produktu: zeszyt
Ilosc produktu: 10, Wartosc zakupu: 10
Destruktor klasy TZakup
Destruktor klasy TProdukt1
Destruktor klasy TZakup
Destruktor klasy TProdukt1
Destruktor klasy TZakup
Destruktor klasy TProdukt1
Destruktor klasy TProdukt1
Destruktor klasy TProdukt1

Obiekty dynamiczne, klasy i funkcje zaprzyjaźnione, przeciążanie operatorów

1. Klasy zaprzyjaźnione i funkcje zaprzyjaźnione
2. Przeciążanie operatorów
3. Obiekty dynamiczne

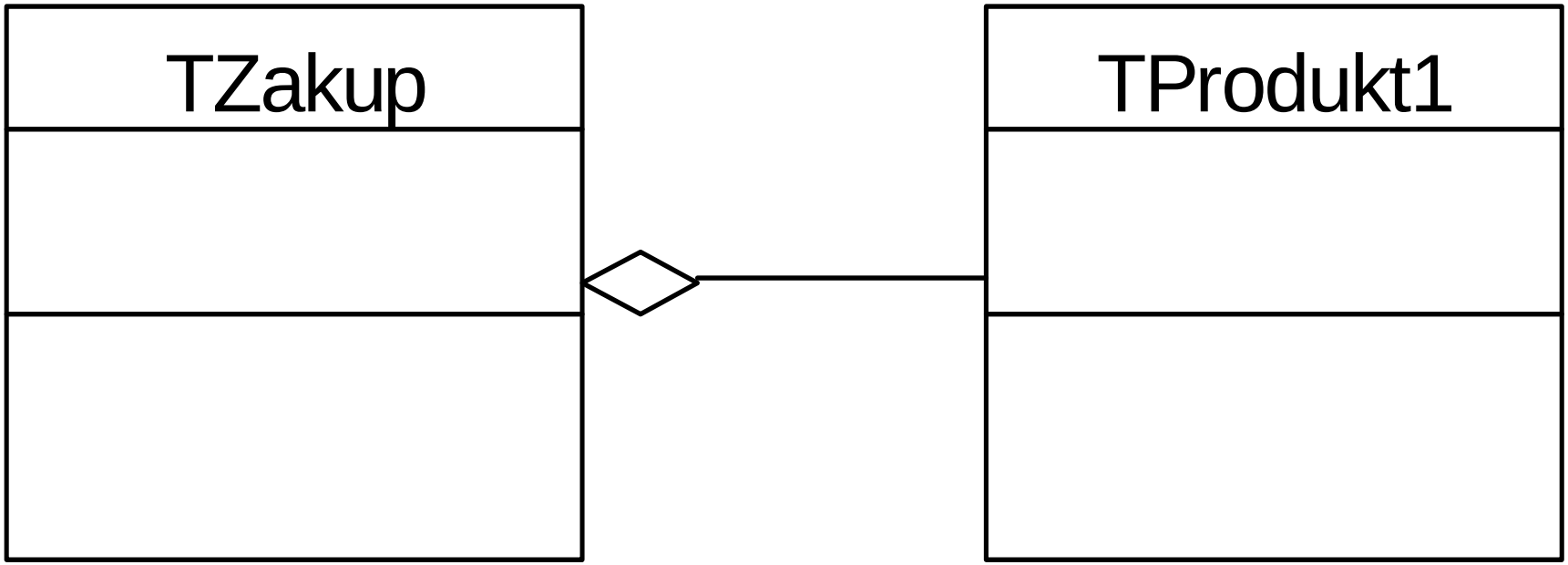

lab3_3.cpp
ZAKUP.CPP
ZAKUP.h
lab3_3.cpp
PRODUKT1.CPP
#include "zakup.h"
void main()
{
{TProdukt1* p1 = new TProdukt1("zeszyt", 1.0);
TProdukt1* p2 = new TProdukt1("olowek",0.80);
cout<<*p1;
cout<<*p2;
//kazdy obiekt TZakup musi posiadać wlasny obiekt TProdukt1
TZakup* z1 = new TZakup(*p1,4);
TZakup* z2 = new TZakup(*p2,3);
TZakup* z3 = new TZakup(*p1,6);
cout<<*z1;
cout<<*z2;
cout<<*z3;
if (*z1==*z2) *z1+=*z2;
if(*z1==*z3) *z1+=*z3;
cout<<*z1;
delete p1;
delete p2;
delete z1;
delete z2;
delete z3;
}
cin.get();
}
1
2
3
4
5
6
7
8
9
10
2: 1
Modified
Insert

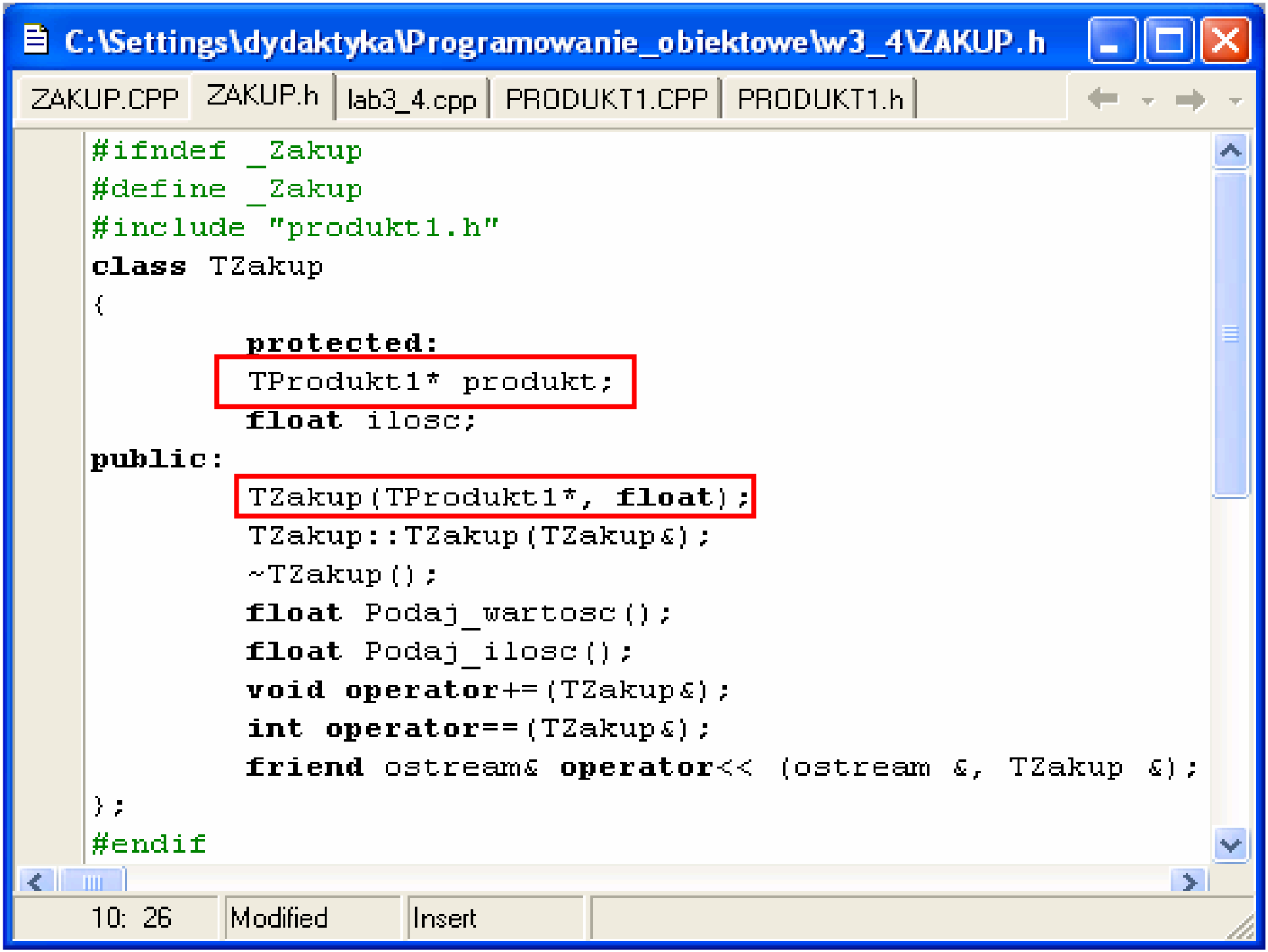

C:\Settings\dydaktyka\Programowanie_obiektowe...
1
2
3
4
5
6
7
8
9
10
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TProdukt1
Cena produktu: 1, Nazwa produktu: zeszyt
Cena produktu: 0.8, Nazwa produktu: olowek
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TZakup
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TZakup
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TZakup
Cena produktu: 1, Nazwa produktu: zeszyt
Ilosc produktu: 4, Wartosc zakupu: 4
Cena produktu: 0.8, Nazwa produktu: olowek
Ilosc produktu: 3, Wartosc zakupu: 2.4
Cena produktu: 1, Nazwa produktu: zeszyt
Ilosc produktu: 6, Wartosc zakupu: 6
Cena produktu: 1, Nazwa produktu: zeszyt
Ilosc produktu: 10, Wartosc zakupu: 10
Destruktor klasy TProdukt1
Destruktor klasy TProdukt1
Destruktor klasy TZakup
Destruktor klasy TProdukt1
Destruktor klasy TZakup
Destruktor klasy TProdukt1
Destruktor klasy TZakup
Destruktor klasy TProdukt1

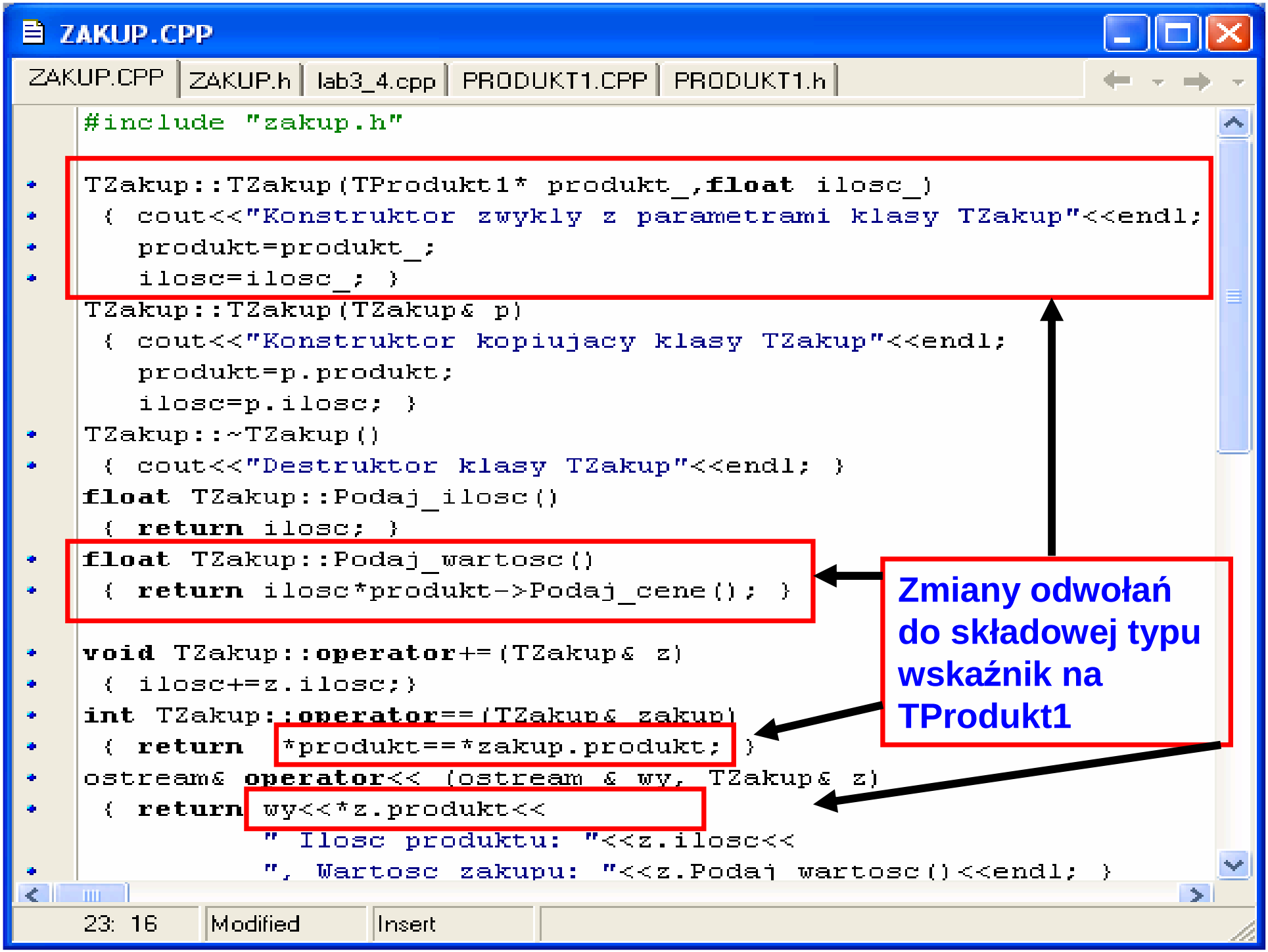
Obiekty dynamiczne, klasy i funkcje zaprzyjaźnione, przeciążanie operatorów

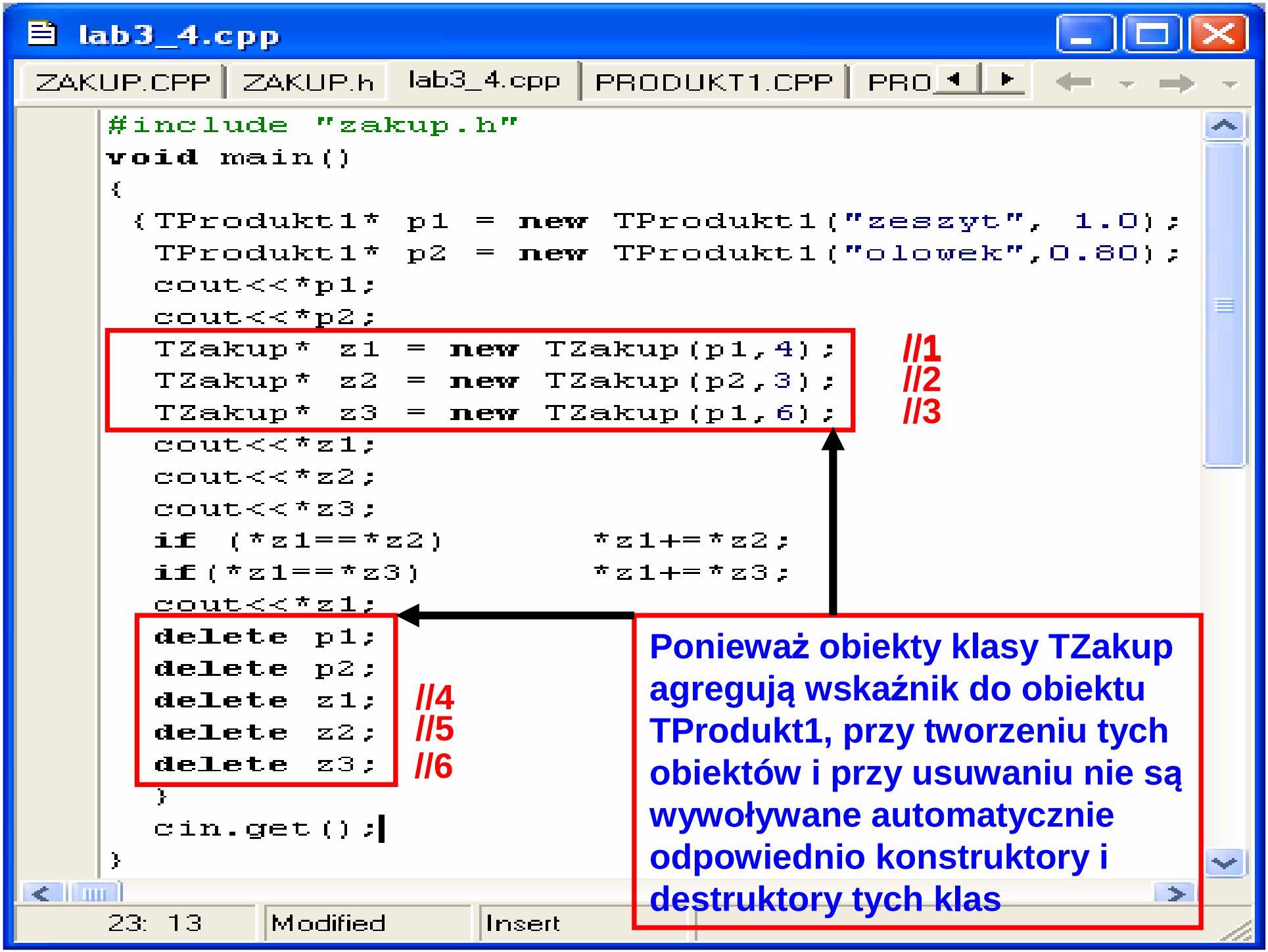
1. Klasy zaprzyjaźnione i funkcje zaprzyjaźnione
2. Przeciążanie operatorów
3. Obiekty dynamiczne
4. Obiekty dynamiczne jako składowe klas

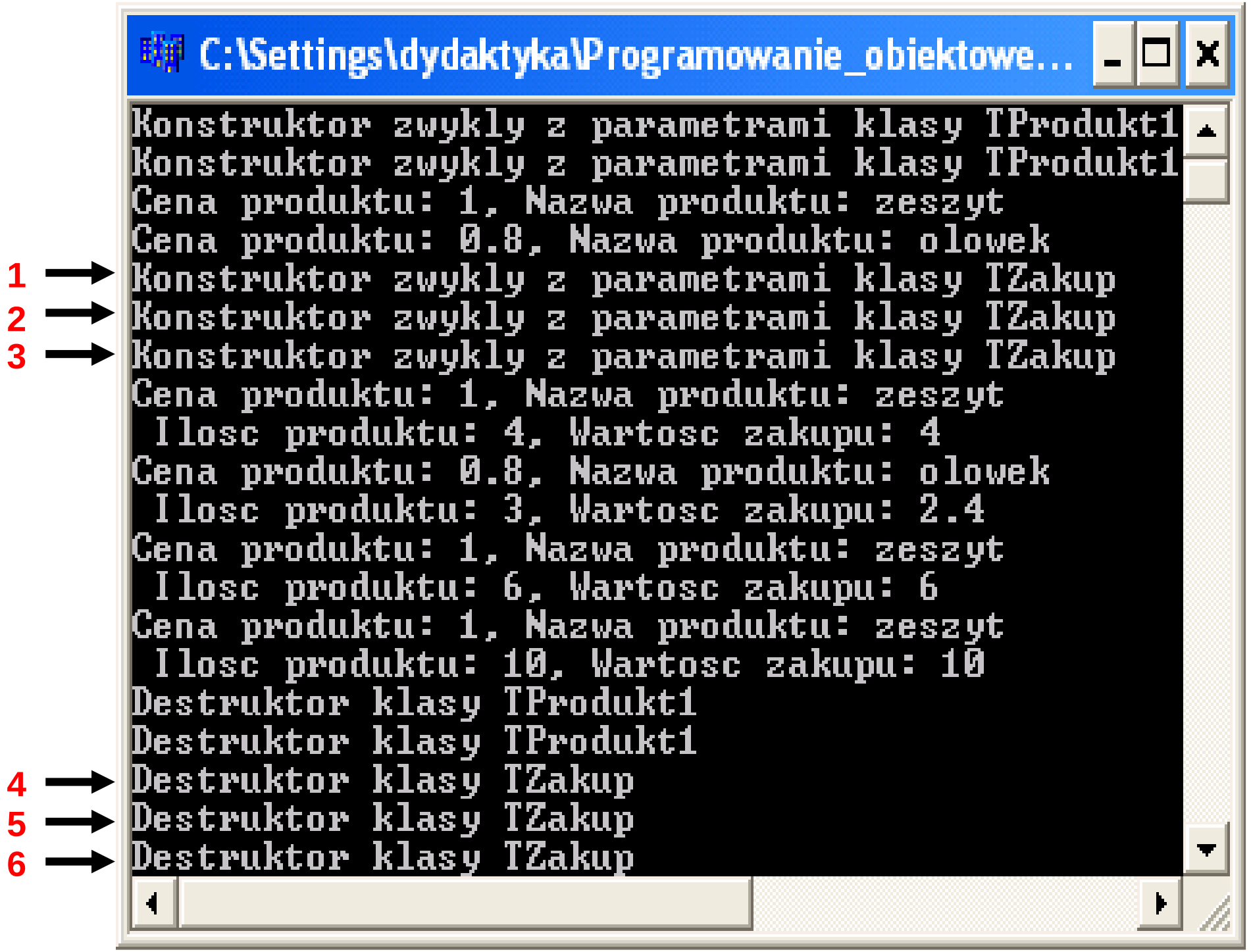
Klasa złożona TZakup agreguje słabo obiekt klasy TProdukt1




C:\Settings\dydaktyka\Programowanie_obiektowe\w3_4\ZAKUP.h
ZAKUP.CPP
ZAKUP.h
lab3_4.cpp
PRODUKT1.CPP
PRODUKT1.h
#ifndef _Zakup
#define _Zakup
#include "produkt1.h"
class TZakup
{
protected:
TProdukt1* produkt;
float ilosc;
public:
TZakup(TProdukt1*, float);
TZakup::TZakup(TZakup&);
~TZakup();
float Podaj_wartosc();
float Podaj_ilosc();
void operator+=(TZakup&);
int operator==(TZakup&);
friend ostream& operator<< (ostream &, TZakup &);
};
#endif
10: 26
Modified
Insert


ZAKUP.CPP
ZAKUP.CPP
ZAKUP.h
lab3_4.cpp
PRODUKT1.CPP
PRODUKT1.h
#include "zakup.h"
TZakup::TZakup(TProdukt1* produkt_,float ilosc_)
{ cout<<"Konstruktor zwykly z parametrami klasy TZakup"<<endl;
produkt=produkt_;
ilosc=ilosc_; }
TZakup::TZakup(TZakup& p)
{ cout<<"Konstruktor kopiujacy klasy TZakup"<<endl;
produkt=p.produkt;
ilosc=p.ilosc; }
TZakup::~TZakup()
{ cout<<"Destruktor klasy TZakup"<<endl; }
float TZakup::Podaj_ilosc()
{ return ilosc; }
float TZakup::Podaj_wartosc()
{ return ilosc*produkt->Podaj_cene(); }
void TZakup::operator+=(TZakup& z)
{ ilosc+=z.ilosc;}
int TZakup::operator==(TZakup& zakup)
{ return *produkt==*zakup.produkt; }
ostream& operator<< (ostream & wy, TZakup& z)
{ return wy<<*z.produkt<<
" Ilosc produktu: "<<z.ilosc<<
", Wartosc zakupu: "<<z.Podaj_wartosc()<<endl; }
Zmiany odwołań do składowej typu wskaźnik na TProdukt1
23: 16
Modified
Insert


lab3_4.cpp
ZAKUP.CPP
ZAKUP.h
lab3_4.cpp
PRODUKT1.CPP
PRO
#include "zakup.h"
void main()
{
(TProdukt1* p1 = new TProdukt1("zeszyt", 1.0);
TProdukt1* p2 = new TProdukt1("olowek",0.80);
cout<<*p1;
cout<<*p2;
TZakup* z1 = new TZakup(p1,4);
TZakup* z2 = new TZakup(p2,3);
TZakup* z3 = new TZakup(p1,6);
cout<<*z1;
cout<<*z2;
cout<<*z3;
if (*z1==*z2) *z1+=*z2;
if(*z1==*z3) *z1+=*z3;
cout<<*z1;
delete p1;
delete p2;
delete z1;
delete z2;
delete z3;
}
cin.get();
}
//1
//2
//3
//4
//5
//6
Ponieważ obiekty klasy TZakup agregują wskaźnik do obiektu TProdukt1, przy tworzeniu tych obiektów i przy usuwaniu nie są wywoływane automatycznie odpowiednio konstruktory i destruktory tych klas
23: 13
Modified
Insert


C:\Settings\dydaktyka\Programowanie_obiektowe...
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TProdukt1
Cena produktu: 1, Nazwa produktu: zeszyt
Cena produktu: 0.8, Nazwa produktu: olowek
1
Konstruktor zwykly z parametrami klasy TZakup
2
Konstruktor zwykly z parametrami klasy TZakup
3
Konstruktor zwykly z parametrami klasy TZakup
Cena produktu: 1, Nazwa produktu: zeszyt
Ilosc produktu: 4, Wartosc zakupu: 4
Cena produktu: 0.8, Nazwa produktu: olowek
Ilosc produktu: 3, Wartosc zakupu: 2.4
Cena produktu: 1, Nazwa produktu: zeszyt
Ilosc produktu: 6, Wartosc zakupu: 6
Cena produktu: 1, Nazwa produktu: zeszyt
Ilosc produktu: 10, Wartosc zakupu: 10
Destruktor klasy TProdukt1
Destruktor klasy TProdukt1
4
Destruktor klasy TZakup
5
Destruktor klasy TZakup
6
Destruktor klasy TZakup