Tworzenie obiektów prostych i złożonych

1. Przeciążanie funkcji składowych, przekazywanie parametrów obiektowych przez wartość, referencję oraz wskaźnik
2. Przeciążanie konstruktora - konstruktor bezparametrowy i z parametrami
3. Konstruktor z parametrami domniemanymi
4. Lista inicjalizująca konstruktora
5. Obiekty z atrybutami obiektowym – obiekty klas złożonych, obiekty klas agregujących
6. Zwracanie obiektów przez metodę, lista inicjalizująca konstruktora klasy z atrybutami obiektowymi

Tworzenie obiektów prostych i złożonych

1. **Przeciążanie funkcji składowych, przekazywanie parametrów obiektowych przez wartość, referencję oraz wskaźnik**

Przeciążanie funkcji

Przeciążanie funkcji - nadawanie tej samej nazwy w tym samym zasięgu dla różnych operacji (różne ciała funkcji) na różnych typach danych (*różne listy parametrów funkcji - liczba i typ parametrów*).

Przy przeciążaniu funkcji wybór egzemplarza funkcji przez kompilator następuje według następujących zasad kolejności konwersji (najlepszego dopasowania):

1. *Ścisła zgodność: nie trzeba stosować żadnych konwersji (np. nazwa tablicy na wskaźnik elementu tablicy, nazwa funkcji na wskaźnik do funkcji),*
2. *Zgodność przy zastosowaniu promowania w zakresie typów całkowitych (np. **char** na **int**, **short** na **int** i ich odpowiedniki bez znaku) oraz typów zmiennoprzecinkowych np. **float** na **double***
3. *Zgodność przy zastosowaniu standardowych konwersji (np. **int** na **double**, klasa pochodna* na podstawowa*, **unsigned int** na **int**)*
4. *Zgodność przy zastosowaniu konwersji zdefiniowanych prze użytkownika*
5. *Zgodność przy zastosowaniu wielokropka .... w deklaracji funkcji*

Uwagi:

1. *Przy wywołaniu funkcje z parametrami przekazywanymi przez referencję nie są rozróżniane z parametrami przekazywanymi przez wartość, stąd czasem stosuje się sztuczne zabiegi (np. dodatkowy argument)*

   np.

***void** f(**int**&) i **void** f(**int**) → **void** f(**int**&, int) i **void** f(**int**)*

2. *Funkcje, które różnią się jedynie typem zwracanej wartości, nie mogą mieć tej samej nazwy np.* ***int*** *f() i* ***double*** *f()*

3. *Obiekty przekazane jako parametry funkcji niezależnych udostępniają jedynie swoje składowe publiczne (**public**)*

Przekazywanie parametrów przez funkcje i metody

Sposób	Uwagi	Przykład
Referencja	na stos przekazywany jest niejawnie adres obiektu aktualnego i nie są tworzone obiekty automatyczne	**void** wyswietl(punkt &p)
Wskaźnik	na stos przekazywany jest jawnie adres obiektu aktualnego i nie są tworzone obiekty automatyczne	**void** wyswietl(punkt *p)
Wartość	na stosie tworzony jest obiekt automatyczny za pomocą konstruktora kopiującego (jawnego lub domyślnego-wykład 2_3). Po zakończeniu bloku obiektu automatycznego ({}) w funkcji usuwa się obiekt za pomocą destruktora (jawnego lub domyślnego-wykład 2_3)	**void** wyswietl(punkt p)

Obiekty przekazane jako parametry funkcji składowych swojej klasy:

udostępniają swoje składowe prywatne i zabezpieczone (private i protected).

Obiekty przekazane jako parametry funkcji składowych innej klasy:

udostępniają jedynie składowe publiczne (public).

C:\Settings\dydaktyka\Programowanie_obiektowe\w2_1\TProdukt1.h

produkty.cpp | TProdukt1.cpp | TProdukt1.h

```
//plik nagłówkowy TProdukt1.h - nie należy go wstawiać do projektu
//---------------------------------------------------------------------
 #include <iostream.h>
//---------------------------------------------------------------------
#ifndef TPRODUKT1     //warunkowe dołączanie zawartości pliku
#define TPRODUKT1                          // nagłówkowego
class TProdukt1
 { //protected:
         string nazwa;
         float cena;
  public:
         TProdukt1(string, float );
         TProdukt1(TProdukt1&);
         ~TProdukt1();
         float Podaj_cene();
         void Nadaj_cene(float );
         string Podaj_nazwe ();
         void Nadaj_nazwe(string );
         int Porownaj_produkty(TProdukt1 );
         int Porownaj_produkty(TProdukt1&,int);
         int Porownaj_produkty(TProdukt1* );
         void Wyswietl();
 };
#endif
```

Przeciążone nazwy metod

6: 1 | Modified | Insert

TProdukt1.cpp

produkty.cpp | TProdukt1.cpp

```
#include "TProdukt1.h"     // zawartość pliku modułowego TProdukt1.cpp

TProdukt1::TProdukt1(string nazwa_, float cena_)
 { cout<<"Wywolany zwykly konstruktor z parametrami"<<endl;
   nazwa = nazwa_;
   cena = cena_; }
TProdukt1::TProdukt1(TProdukt1& p)
 { cout<<"Wywolany konstruktor kopiujacy"<<endl;
   nazwa = p.nazwa;
   cena = p.cena; }
TProdukt1::~TProdukt1()
 { cout << "Wywolany destrutor"<<endl; }
float TProdukt1::Podaj_cene()
 { return cena; }
void TProdukt1::Nadaj_cene(float cena_)
 { cena = cena_; }
string TProdukt1::Podaj_nazwe ()
 { return nazwa;}
void TProdukt1::Nadaj_nazwe(string nazwa_)
 { nazwa = nazwa_;}
int TProdukt1::Porownaj_produkty(TProdukt1 p)
 { return  Podaj_nazwe()==p.nazwa && Podaj_cene()==p.Podaj_cene(); }
int TProdukt1::Porownaj_produkty(TProdukt1& p, int)
 { return  Podaj_nazwe()==p.nazwa; }
int TProdukt1::Porownaj_produkty(TProdukt1* p)
 { return  Podaj_cene()==(*p).Podaj_cene(); }
void TProdukt1::Wyswietl()
 { cout<<"Cena produktu: "<<cena<<", Nazwa produktu: "<<nazwa<<endl; }
```

Przeciążone nazwy metod – różne ciała metod czyli można na trzy sposoby porównywać produkty

1: 30 | Modified | Insert

```
#include "TProdukt1.h"    // zawartość pliku modułowego TProdukt1.cpp

TProdukt1::TProdukt1(string nazwa_, float cena_)
 { cout<<"Wywolany zwykly konstruktor z parametrami"<<endl;
   nazwa = nazwa_;
   cena = cena_; }
TProdukt1::TProdukt1(TProdukt1& p)
 { cout<<"Wywolany konstruktor kopiujacy"<<endl;
   nazwa = p.nazwa;
   cena = p.cena; }
TProdukt1::~TProdukt1()
 { cout << "Wywolany destrutor"<<endl; }
float TProdukt1::Podaj_cene()
 { return cena; }
void TProdukt1::Nadaj_cene(float cena_)
 { cena = cena_; }
string TProdukt1::Podaj_nazwe ()
 { return nazwa;}
void TProdukt1::Nadaj_nazwe(string nazwa_)
 { nazwa = nazwa_;}
```

//1 – przekazywanie obiektu przez wartość

//2 – przekazywanie przez referencję. Aby kompilator rozróżnił nagłówki metod z p.1 i z p.2, należy jeden z nich zróżnicować np. w p.2 przez dodanie dodatkowego parametru

//3 – przekazywanie obiektu przez wskaźnik

W metodach własnej klasy nie obowiązuje ochrona protected oraz private dla składowych obiektów tej klasy

```
int TProdukt1::Porownaj_produkty(TProdukt1 p) //1
 { return  Podaj_nazwe() == p.nazwa &&
           Podaj_cene() == p.Podaj_cene(); }
int TProdukt1::Porownaj_produkty(TProdukt1& p, int) //2
 { return  Podaj_nazwe() == p.nazwa ;}
int TProdukt1::Porownaj_produkty(TProdukt1* p) //3
 { return Podaj_cene() == (*p).Podaj_cene(); }
          //p->Podaj_cene();
```

p-> lub (*p). - dwa równoważne sposoby wyłuskania wskaźnika do obiektu i wybór składowej

```
void TProdukt1::Wyswietl()
 { cout<<"Cena produktu: "<<cena<<", Nazwa produktu: "<<nazwa<<endl; }
```

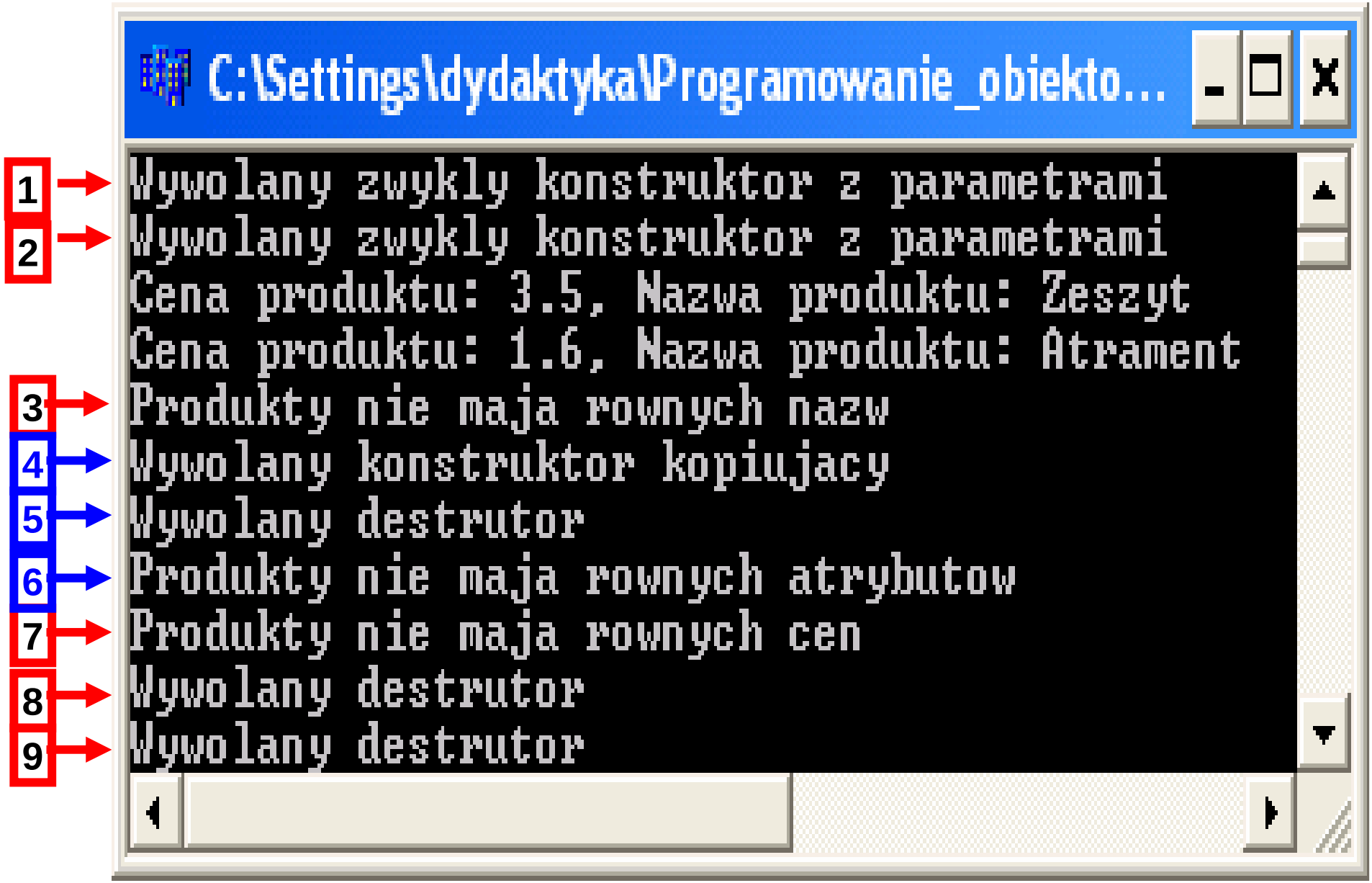
produkty.cpp

produkty.cpp | TProdukt1.cpp

```
//---------------------------------------------------------------------
#pragma hdrstop
 #include "TProdukt1.h"    //zawartość pliku z funkcją main
//---------------------------------------------------------------------
//#pragma argsused
//---------------------------------------------------------------------
int main(int argc, char* argv[])
{
   { TProdukt1 produkt1("Zeszyt", 3.5), produkt2("Atrament", 1.6);
     produkt1.Wyswietl();
     produkt2.Wyswietl();
     if (produkt1.Porownaj_produkty(produkt2,3))
                cout<<"Produkty maja rowne nazwy"<<endl;
     else    cout<<"Produkty nie maja rownych nazw"<<endl;
     if (produkt1.Porownaj_produkty(produkt2))
                cout<<"Produkty maja rowne atrybuty"<<endl;
     else    cout<<"Produkty nie maja rownych atrybutow"<<endl;
     if (produkt1.Porownaj_produkty(&produkt2))
                cout<<"Produkty maja rowne ceny"<<endl;
     else    cout<<"Produkty nie maja rownych cen"<<endl;
    }
    cin.get();
    return 0;
}
//---------------------------------------------------------------------
```

20: 48 | Modified | Insert

```
#include "TProdukt1.h"   //zawartość pliku z funkcją main
int main(int argc, char* argv[])
{
  { TProdukt1 produkt1("Zeszyt", 3.5), produkt2("Atrament", 1.6); //1,2
    produkt1.Wyswietl();
    produkt2.Wyswietl();
    if (produkt1.Porownaj_produkty(produkt2,3))
            cout<<"Produkty maja rowne nazwy"<<endl;
    else   cout<<"Produkty nie maja rownych nazw"<<endl; //3
    if (produkt1.Porownaj_produkty(produkt2)) //4,5
            cout<<"Produkty maja rowne atrybuty"<<endl;
    else   cout<<"Produkty nie maja rownych atrybutow"<<endl; //6
    if (produkt1.Porownaj_produkty(&produkt2)) //przekazanie przez wskaźnik
            cout<<"Produkty maja rowne ceny"<<endl;
    else   cout<<"Produkty nie maja rownych cen"<<endl; //7
   } //8,9
   cin.get();
   return 0;
}
```


C:\Settings\dydaktyka\Programowanie_obiekto...
Wywolany zwykly konstruktor z parametrami
Wywolany zwykly konstruktor z parametrami
Cena produktu: 3.5, Nazwa produktu: Zeszyt
Cena produktu: 1.6, Nazwa produktu: Atrament
Produkty nie maja rownych nazw
Wywolany konstruktor kopiujacy
Wywolany destrutor
Produkty nie maja rownych atrybutow
Produkty nie maja rownych cen
Wywolany destrutor
Wywolany destrutor
1
2
3
4
5
6
7
8
9

Tworzenie obiektów prostych i złożonych

1. **Przeciążanie funkcji składowych, przekazywanie parametrów obiektowych przez wartość, referencję oraz wskaźnik**
2. **Przeciążanie konstruktora - konstruktor bezparametrowy i z parametrami**

C:\Settings\dydaktyka\Programowanie_obiektowe\w2_2\TProdukt1.h

produkty.cpp | TProdukt1.cpp | TProdukt1.h

```
//plik nagłówkowy TProdukt1.h - nie należy go wstawiać do projektu
//---------------------------------------------------------------------
 #include <iostream.h>
//---------------------------------------------------------------------
#ifndef TPRODUKT1     //warunkowe dołączanie zawartości pliku
#define TPRODUKT1                          // nagłówkowego
class TProdukt1
 { //protected:
        string nazwa;
        float cena;
  public:
        TProdukt1(string, float );
        TProdukt1( );
        TProdukt1(TProdukt1&);
        ~TProdukt1();
        float Podaj_cene();
        void Nadaj_cene(float );
        string Podaj_nazwe ();
        void Nadaj_nazwe(string );
        int Porownaj_produkty(TProdukt1& );
        void Wyswietl();
 };
#endif
```

Konstruktory przeciążone:

1. z parametrami

2. bez parametrów

18: 43 | Insert

TProdukt1.cpp

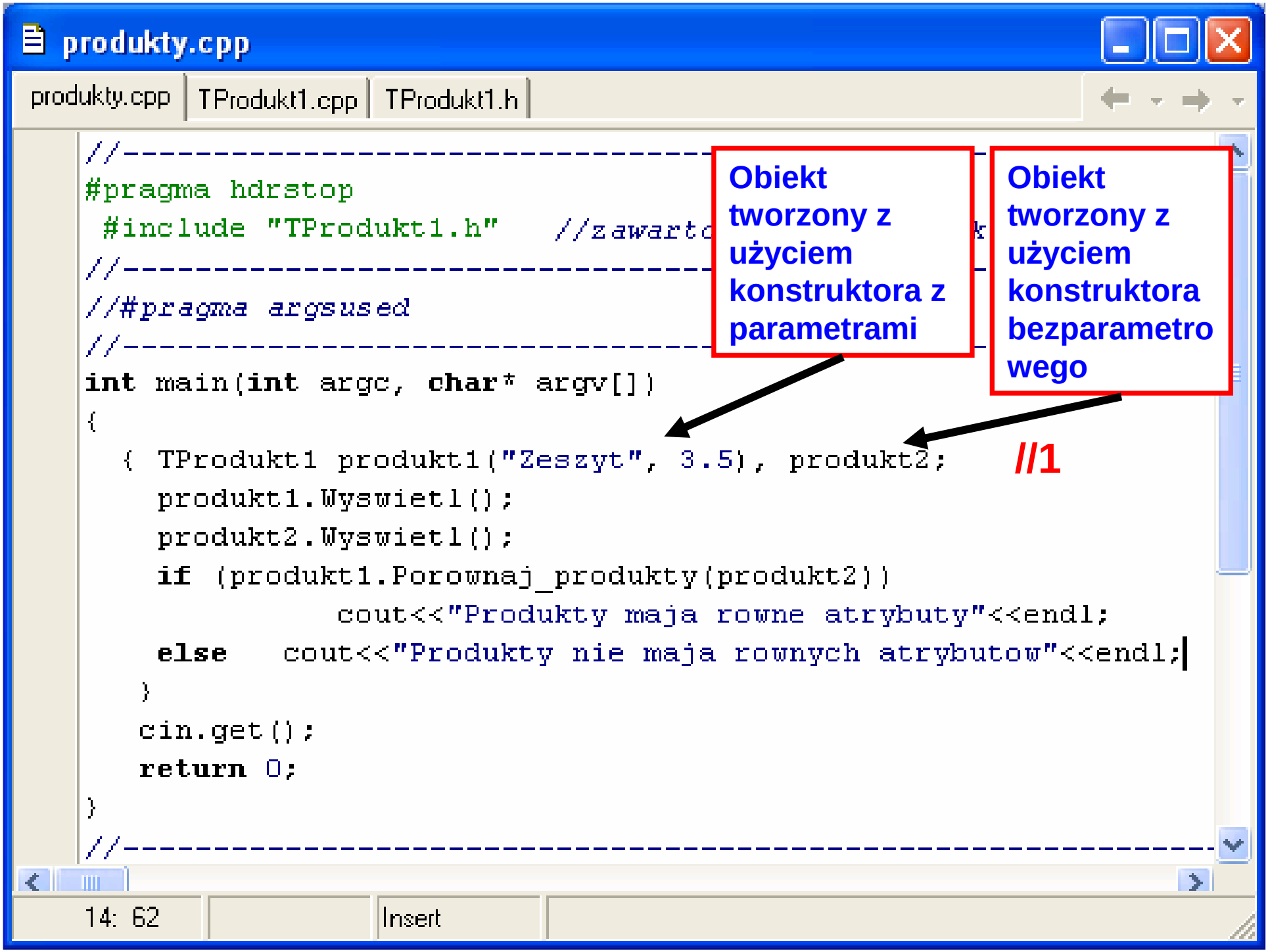
produkty.cpp | TProdukt1.cpp | TProdukt1.h

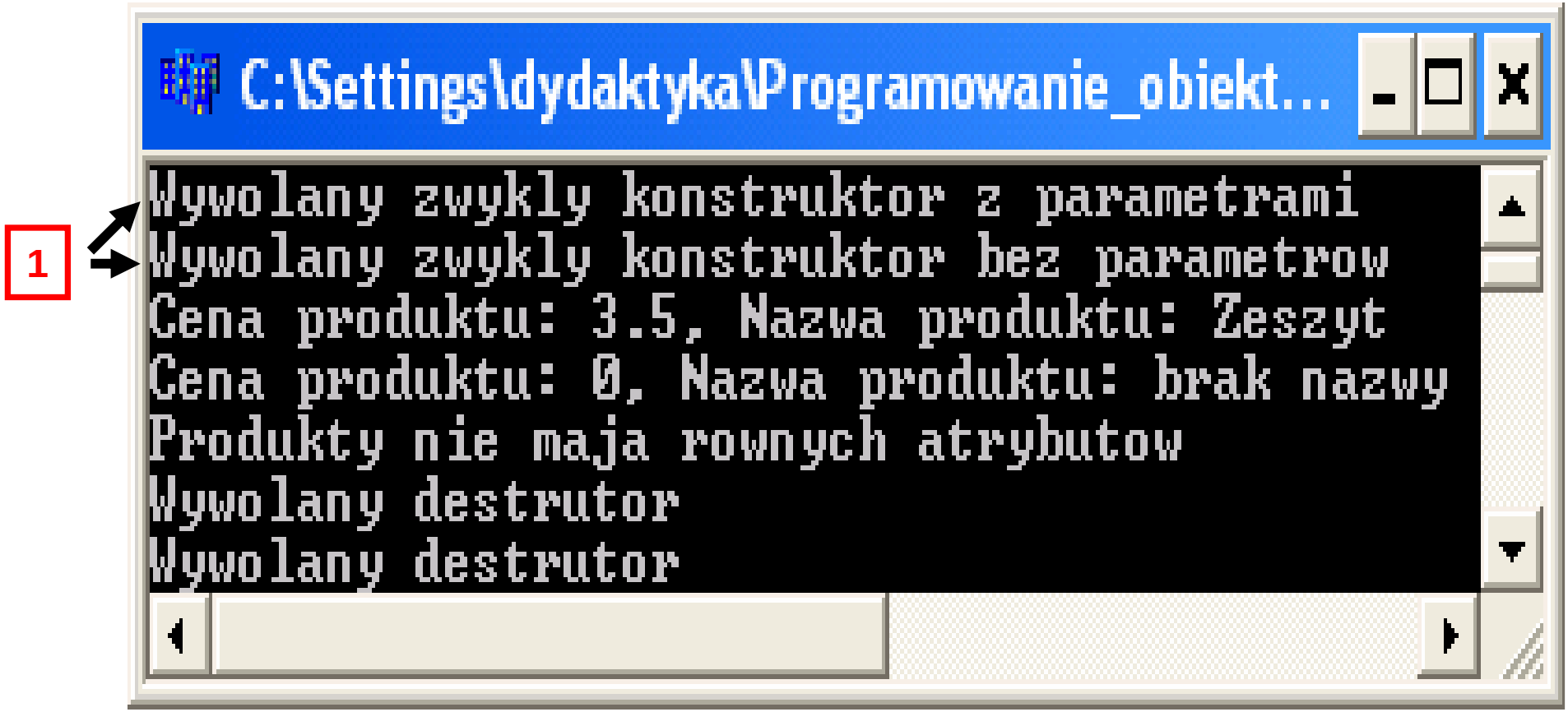
```
#include "TProdukt1.h"          // zawartość pliku modułowego TProdukt1

TProdukt1::TProdukt1(string nazwa_, float cena_)
 { cout<<"Wywolany zwykly konstruktor z parametrami"<<endl;
   nazwa = nazwa_;
   cena = cena_; }
TProdukt1::TProdukt1()
 { cout<<"Wywolany zwykly konstruktor bez parametrow"<<endl;
   nazwa ="brak nazwy";
   cena = 0; }
TProdukt1::TProdukt1(TProdukt1& p)
 { cout<<"Wywolany konstruktor kopiujacy"<<endl;
   nazwa = p.nazwa;
   cena = p.cena; }
TProdukt1::~TProdukt1()
 { cout << "Wywolany destrutor"<<endl; }
float TProdukt1::Podaj_cene()
 { return cena; }
void TProdukt1::Nadaj_cene(float cena_)
 { cena = cena_; }
string TProdukt1::Podaj_nazwe ()
 { return nazwa;}
void TProdukt1::Nadaj_nazwe(string nazwa_)
 { nazwa = nazwa_;}
int TProdukt1::Porownaj_produkty(TProdukt1& p)
 { return  Podaj_nazwe()==p.nazwa && Podaj_cene()==p.Podaj_cene(); }
void TProdukt1::Wyswietl()
 { cout<<"Cena produktu: "<<cena<<", Nazwa produktu: "<<nazwa<<endl; }
```

Definicje konstruktorów:

- **bez parametrów umożliwia tworzyć obiekty bez rzeczywistych danych**
- **z parametrami umożliwia tworzyć obiekty z danymi rzeczywistymi**

10: 13 Insert


produkty.cpp
produkty.cpp
TProdukt1.cpp
TProdukt1.h
//---------------------------------
#pragma hdrstop
#include "TProdukt1.h"
//---------------------------------
//#pragma argsused
//---------------------------------
int main(int argc, char* argv[])
{
{ TProdukt1 produkt1("Zeszyt", 3.5), produkt2;
produkt1.Wyswietl();
produkt2.Wyswietl();
if (produkt1.Porownaj_produkty(produkt2))
cout<<"Produkty maja rowne atrybuty"<<endl;
else cout<<"Produkty nie maja rownych atrybutow"<<endl;
}
cin.get();
return 0;
}
//---------------------------------
Obiekt tworzony z użyciem konstruktora z parametrami
Obiekt tworzony z użyciem konstruktora bezparametrowego
//1
14: 62
Insert


C:\Settings\dydaktyka\Programowanie_obiekt...
Wywolany zwykly konstruktor z parametrami
Wywolany zwykly konstruktor bez parametrow
Cena produktu: 3.5, Nazwa produktu: Zeszyt
Cena produktu: 0, Nazwa produktu: brak nazwy
Produkty nie maja rownych atrybutow
Wywolany destrutor
Wywolany destrutor
1

Tworzenie obiektów prostych i złożonych

1. **Przeciążanie funkcji składowych, przekazywanie parametrów obiektowych przez wartość, referencję oraz wskaźnik**
2. **Przeciążanie konstruktora - konstruktor bezparametrowy i z parametrami**
3. **Konstruktor z parametrami domniemanymi**

C:\Settings\dydaktyka\Programowanie_obiektowe\w2_3\TProdukt1.h

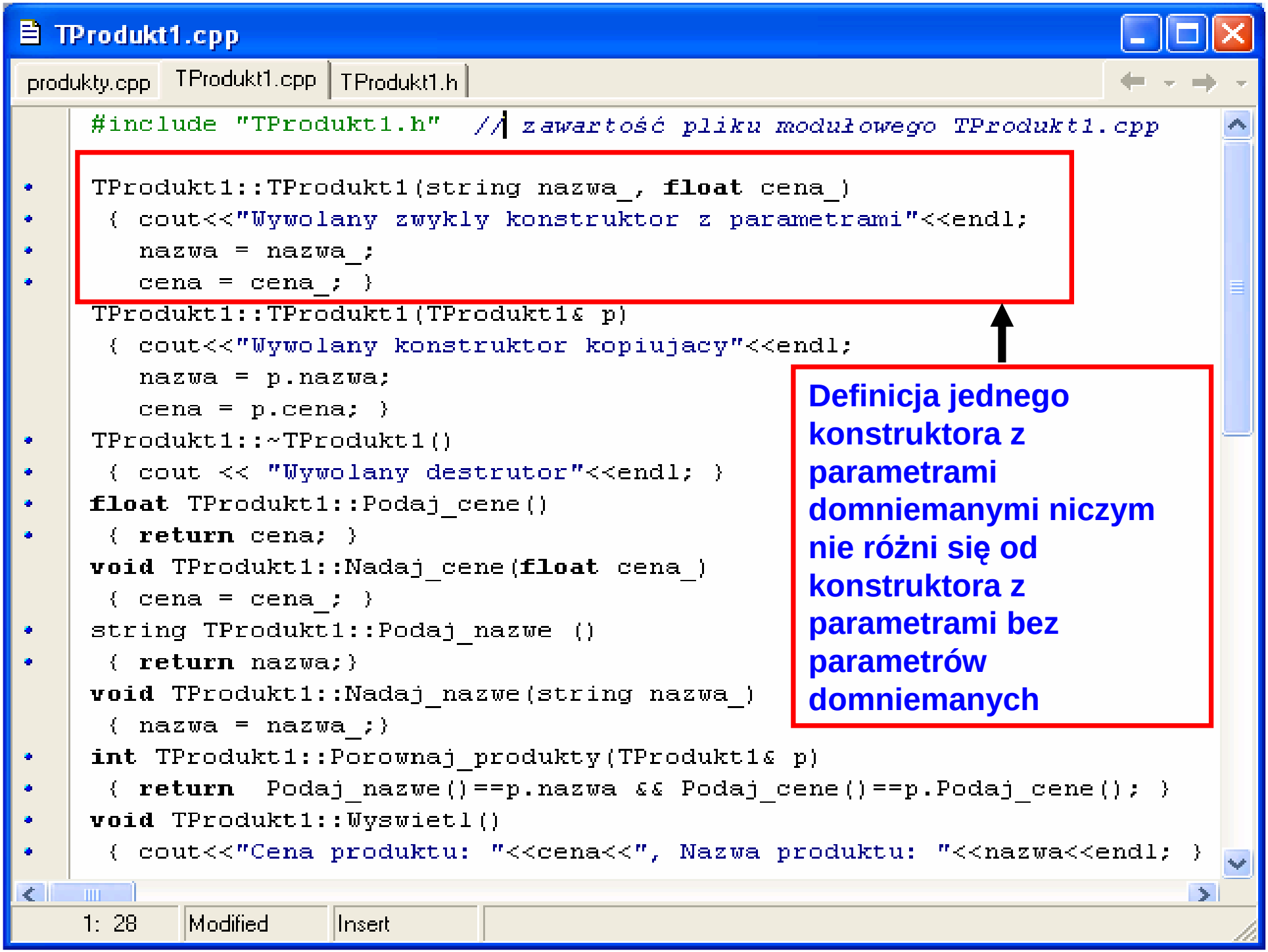
produkty.cpp | TProdukt1.cpp | TProdukt1.h

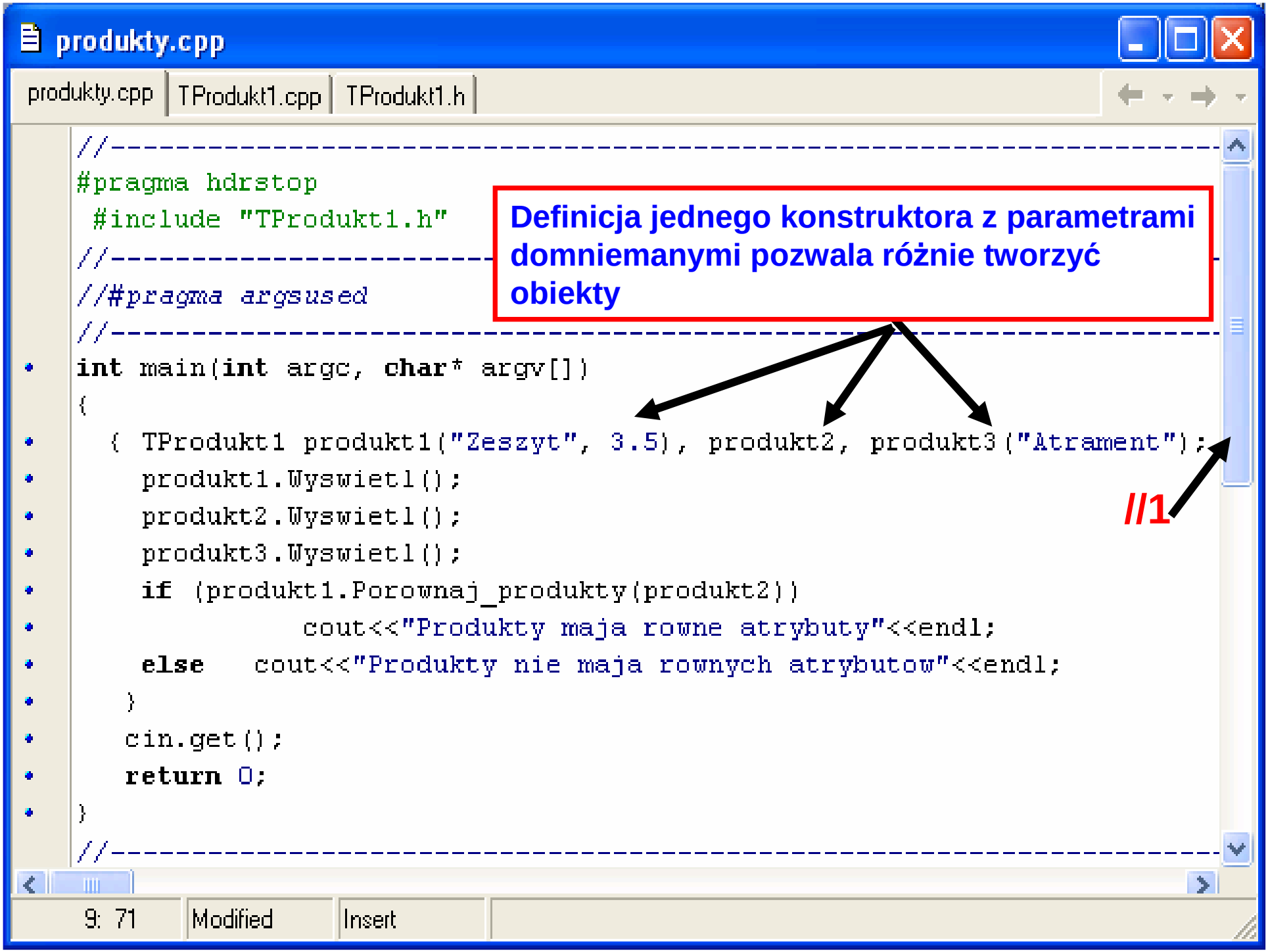
```
//plik nagłówkowy TProdukt1.h - nie należy go wstawiać do projektu
//-------------------------------------------------------------------
 #include <iostream.h>
//-------------------------------------------------------------------
#ifndef TPRODUKT1     //warunkowe dołączanie zawartości pliku
#define TPRODUKT1                              // nagłówkowego
class TProdukt1
 { //protected:
        string nazwa;
        float cena;
  public:
        TProdukt1(string ="brak nazwy", float=0);
        //TProdukt1( );
        TProdukt1(TProdukt1&);
        ~TProdukt1();
        float Podaj_cene();
        void Nadaj_cene(float );
        string Podaj_nazwe ();
        void Nadaj_nazwe(string );
        int Porownaj_produkty(TProdukt1& );
        void Wyswietl();
 };
#endif
```

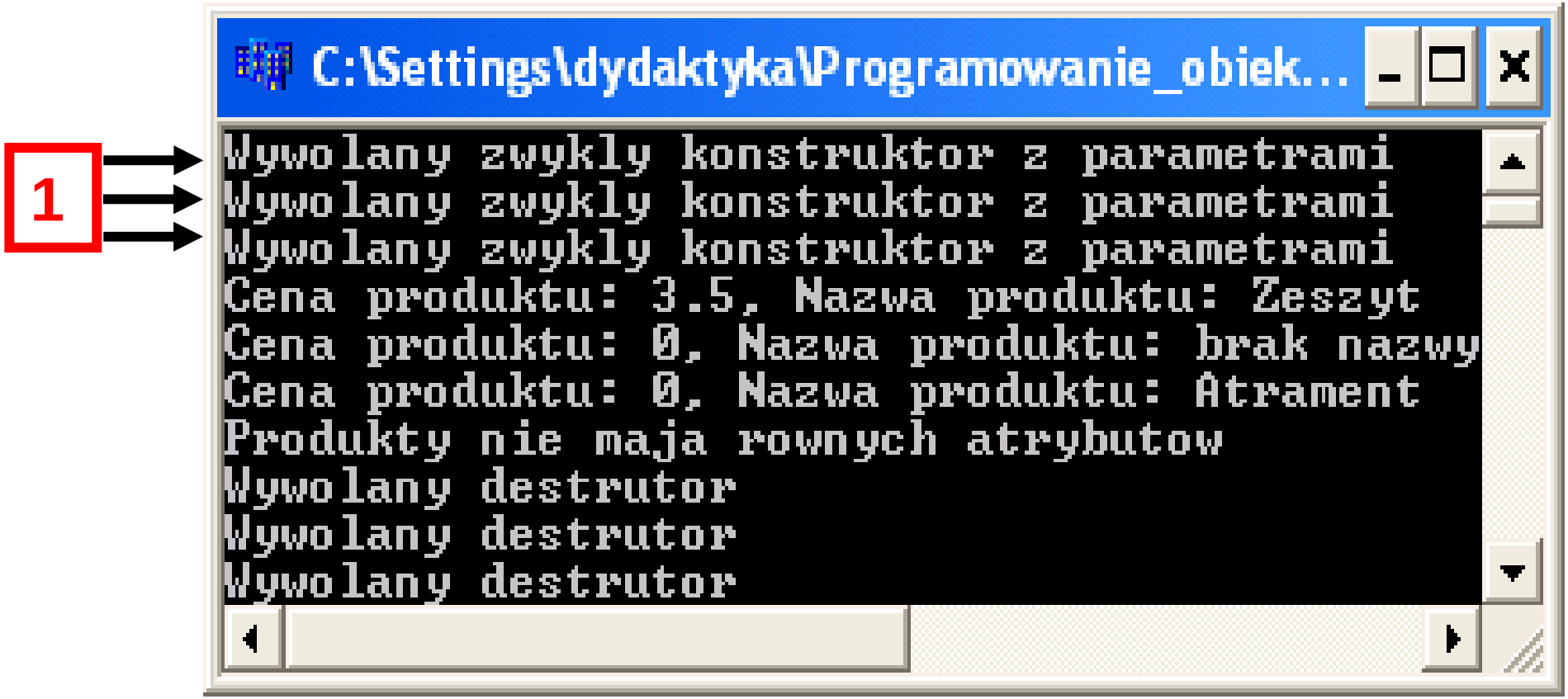
Konstruktor z parametrami domniemanymi.

Pełni on rolę konstruktora z parametrami i bez parametrów oraz z niepełną liczbą parametrów

18: 50 | Modified | Insert


TProdukt1.cpp
produkty.cpp
TProdukt1.cpp
TProdukt1.h
#include "TProdukt1.h" // zawartość pliku modułowego TProdukt1.cpp
TProdukt1::TProdukt1(string nazwa_, float cena_)
{ cout<<"Wywolany zwykly konstruktor z parametrami"<<endl;
nazwa = nazwa_;
cena = cena_; }
TProdukt1::TProdukt1(TProdukt1& p)
{ cout<<"Wywolany konstruktor kopiujacy"<<endl;
nazwa = p.nazwa;
cena = p.cena; }
TProdukt1::~TProdukt1()
{ cout << "Wywolany destrutor"<<endl; }
float TProdukt1::Podaj_cene()
{ return cena; }
void TProdukt1::Nadaj_cene(float cena_)
{ cena = cena_; }
string TProdukt1::Podaj_nazwe ()
{ return nazwa;}
void TProdukt1::Nadaj_nazwe(string nazwa_)
{ nazwa = nazwa_;}
int TProdukt1::Porownaj_produkty(TProdukt1& p)
{ return Podaj_nazwe()==p.nazwa && Podaj_cene()==p.Podaj_cene(); }
void TProdukt1::Wyswietl()
{ cout<<"Cena produktu: "<<cena<<", Nazwa produktu: "<<nazwa<<endl; }
Definicja jednego konstruktora z parametrami domniemanymi niczym nie różni się od konstruktora z parametrami bez parametrów domniemanych
1: 28
Modified
Insert


produkty.cpp
produkty.cpp
TProdukt1.cpp
TProdukt1.h
//---
#pragma hdrstop
#include "TProdukt1.h"
//---
//#pragma argsused
//---
int main(int argc, char* argv[])
{
{ TProdukt1 produkt1("Zeszyt", 3.5), produkt2, produkt3("Atrament");
produkt1.Wyswietl();
produkt2.Wyswietl();
produkt3.Wyswietl();
if (produkt1.Porownaj_produkty(produkt2))
cout<<"Produkty maja rowne atrybuty"<<endl;
else cout<<"Produkty nie maja rownych atrybutow"<<endl;
}
cin.get();
return 0;
}
//---
Definicja jednego konstruktora z parametrami domniemanymi pozwala różnie tworzyć obiekty
//1
9: 71
Modified
Insert


1
C:\Settings\dydaktyka\Programowanie_obiek...
Wywolany zwykly konstruktor z parametrami
Wywolany zwykly konstruktor z parametrami
Wywolany zwykly konstruktor z parametrami
Cena produktu: 3.5, Nazwa produktu: Zeszyt
Cena produktu: 0, Nazwa produktu: brak nazwy
Cena produktu: 0, Nazwa produktu: Atrament
Produkty nie maja rownych atrybutow
Wywolany destrutor
Wywolany destrutor
Wywolany destrutor

Tworzenie obiektów prostych i złożonych

1. **Przeciążanie funkcji składowych, przekazywanie parametrów obiektowych przez wartość, referencję oraz wskaźnik**
2. **Przeciążanie konstruktora - konstruktor bezparametrowy i z parametrami**
3. **Konstruktor z parametrami domniemanymi**
4. **Lista inicjalizująca konstruktora**

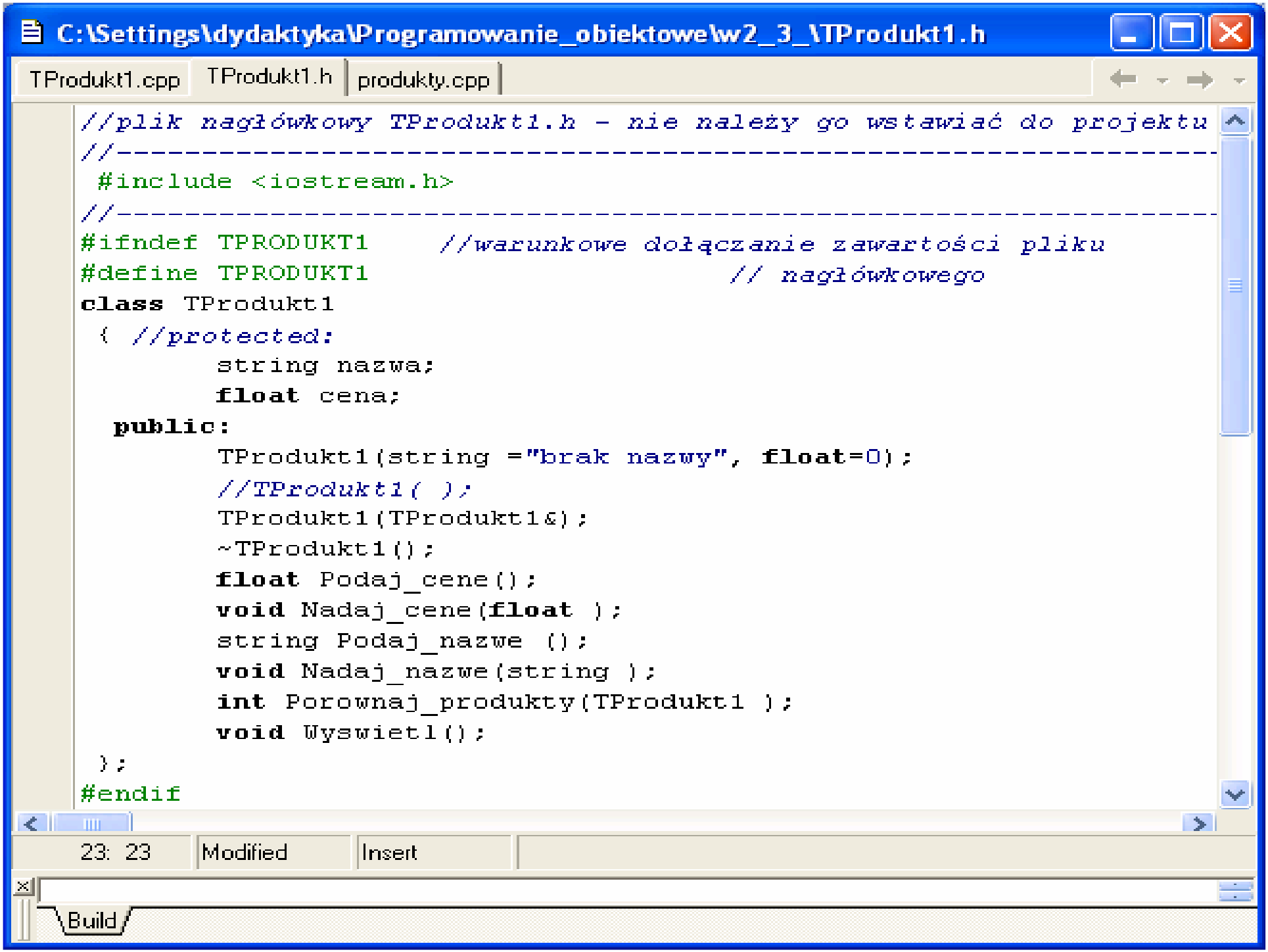
Postać konstruktora z listą inicjalizująca

Klasa::Nagłówek_konstruktora
: lista_inicjalizująca
{ ciało konstruktora }

gdzie:
lista_inicjalizująca zawiera przypisania argumentów konstruktora do składowych klasy typu dane w postaci nawiasu okrągłego, przedzielone przecinkami

np.

```
TProdukt1::TProdukt1(string nazwa_, float cena_)
                    : nazwa(nazwa_), cena(cena_)
 { cout<<"Wywolany zwykly konstruktor z parametrami"<<endl; }

 TProdukt1::TProdukt1(TProdukt1& p)
                    : nazwa(p.nazwa), cena(p.cena)
 { cout<<"Wywolany konstruktor kopiujacy"<<endl; }
```


C:\Settings\dydaktyka\Programowanie_obiektowe\w2_3_\TProdukt1.h
TProdukt1.cpp
TProdukt1.h
produkty.cpp
//plik nagłówkowy TProdukt1.h - nie należy go wstawiać do projektu
//---
#include <iostream.h>
//---
#ifndef TPRODUKT1 //warunkowe dołączanie zawartości pliku
#define TPRODUKT1 // nagłówkowego
class TProdukt1
{ //protected:
string nazwa;
float cena;
public:
TProdukt1(string ="brak nazwy", float=0);
//TProdukt1( );
TProdukt1(TProdukt1&);
~TProdukt1();
float Podaj_cene();
void Nadaj_cene(float);
string Podaj_nazwe ();
void Nadaj_nazwe(string);
int Porownaj_produkty(TProdukt1);
void Wyswietl();
};
#endif
23: 23
Modified
Insert
Build

TProdukt1.cpp

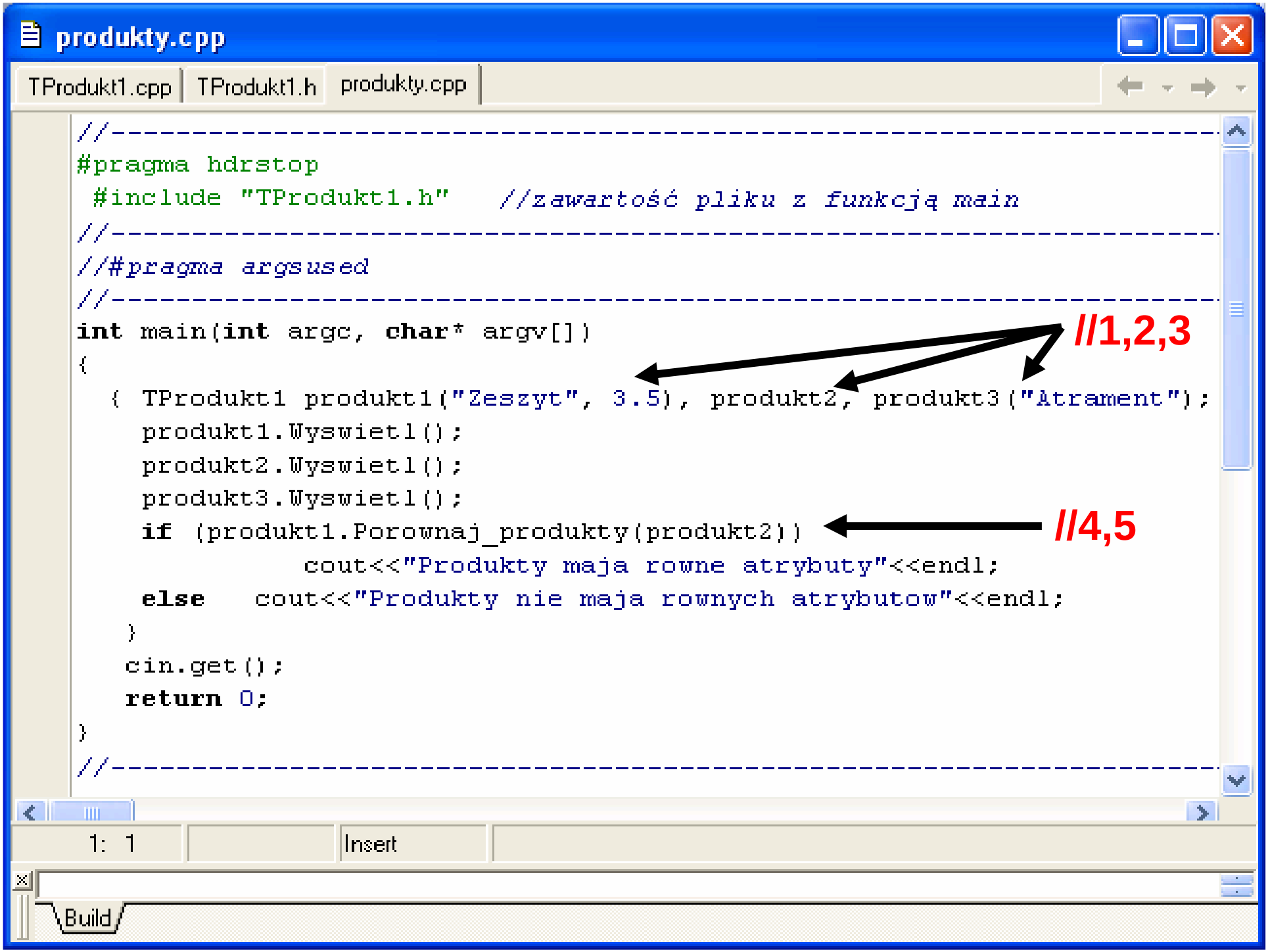
TProdukt1.cpp | TProdukt1.h | produkty.cpp

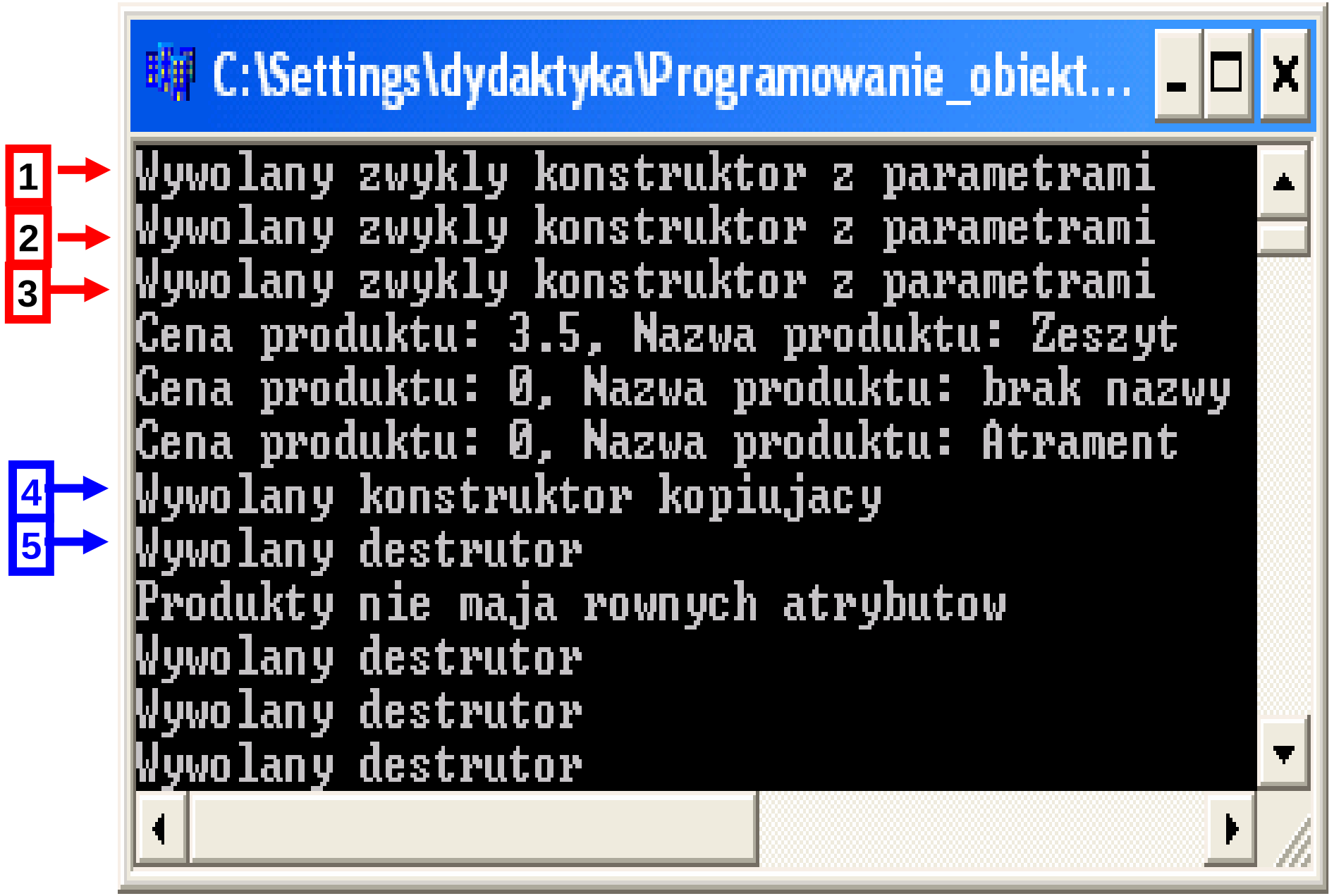
```
#include "TProdukt1.h"  // zawartość pliku modułowego TProdukt1.cpp

TProdukt1::TProdukt1(string nazwa_, float cena_):nazwa(nazwa_),cena(cena_)
 { cout<<"Wywolany zwykly konstruktor z parametrami"<<endl; }
TProdukt1::TProdukt1(TProdukt1& p): nazwa(p.nazwa), cena(p.cena)
 { cout<<"Wywolany konstruktor kopiujacy"<<endl; }
TProdukt1::~TProdukt1()
 { cout << "Wywolany destrutor"<<endl; }
float TProdukt1::Podaj_cene()
 { return cena; }
void TProdukt1::Nadaj_cene(float cena_)
 { cena = cena_; }
string TProdukt1::Podaj_nazwe ()
 { return nazwa;}
void TProdukt1::Nadaj_nazwe(string nazwa_)
 { nazwa = nazwa_;}
int TProdukt1::Porownaj_produkty(TProdukt1 p)
 { return  Podaj_nazwe()==p.nazwa && Podaj_cene()==p.Podaj_cene(); }
void TProdukt1::Wyswietl()
 { cout<<"Cena produktu: "<<cena<<", Nazwa produktu: "<<nazwa<<endl; }
```

Definicje konstruktorów z listą inicjalizującą

20: 74 | Modified | Insert

Build


produkty.cpp
TProdukt1.cpp
TProdukt1.h
produkty.cpp
//--
#pragma hdrstop
#include "TProdukt1.h" //zawartość pliku z funkcją main
//--
//#pragma argsused
//--
int main(int argc, char* argv[])
{
{ TProdukt1 produkt1("Zeszyt", 3.5), produkt2, produkt3("Atrament");
produkt1.Wyswietl();
produkt2.Wyswietl();
produkt3.Wyswietl();
if (produkt1.Porownaj_produkty(produkt2))
cout<<"Produkty maja rowne atrybuty"<<endl;
else cout<<"Produkty nie maja rownych atrybutow"<<endl;
}
cin.get();
return 0;
}
//--
//1,2,3
//4,5
1: 1
Insert
Build


C:\Settings\dydaktyka\Programowanie_obiekt...
1
2
3
4
5
Wywolany zwykly konstruktor z parametrami
Wywolany zwykly konstruktor z parametrami
Wywolany zwykly konstruktor z parametrami
Cena produktu: 3.5, Nazwa produktu: Zeszyt
Cena produktu: 0, Nazwa produktu: brak nazwy
Cena produktu: 0, Nazwa produktu: Atrament
Wywolany konstruktor kopiujacy
Wywolany destrutor
Produkty nie maja rownych atrybutow
Wywolany destrutor
Wywolany destrutor
Wywolany destrutor

Tworzenie obiektów prostych i złożonych

1. **Przeciążanie funkcji składowych, przekazywanie parametrów obiektowych przez wartość, referencję oraz wskaźnik**
2. **Przeciążanie konstruktora - konstruktor bezparametrowy i z parametrami**
3. **Konstruktor z parametrami domniemanymi**
4. **Lista inicjalizująca konstruktora**
5. **Obiekty z atrybutami obiektowym – obiekty klas złożonych, obiekty klas agregujących**

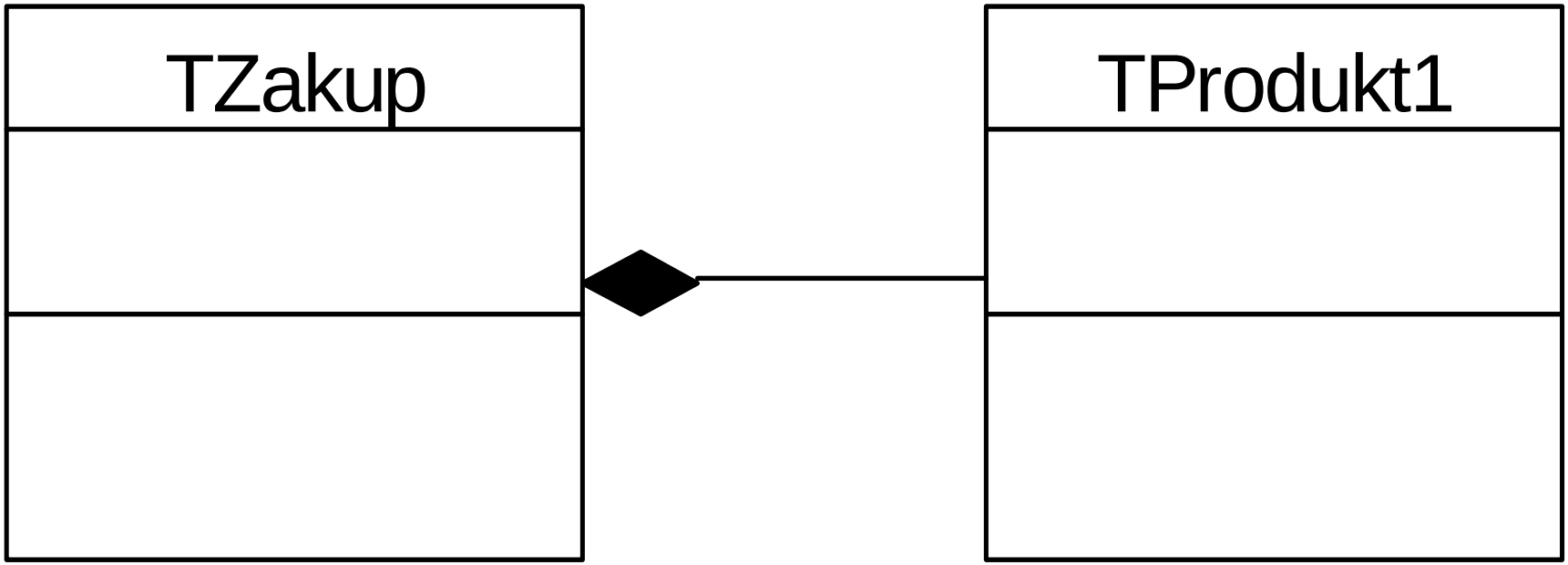
Obiekt posiadający składowe typu obiektowego (agregacja mocna) jest tworzony następująco:

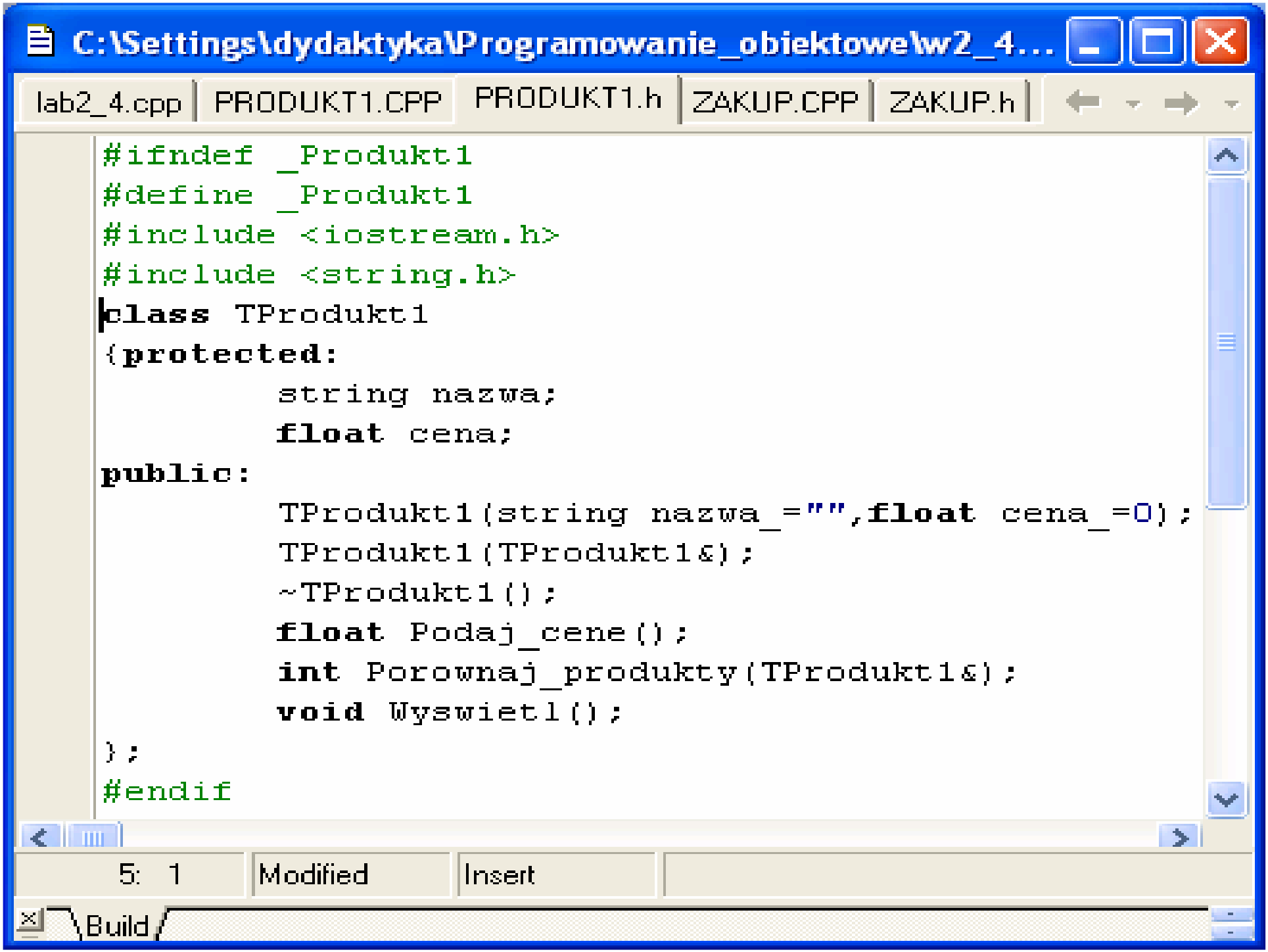
1. Najpierw są tworzone obiekty składowe w kolejności deklaracji własnymi konstruktorami zwykłymi bezparametrowymi
2. Na końcu jest wywoływany konstruktor klasy obiektu, który zawiera składowe typu obiektowego

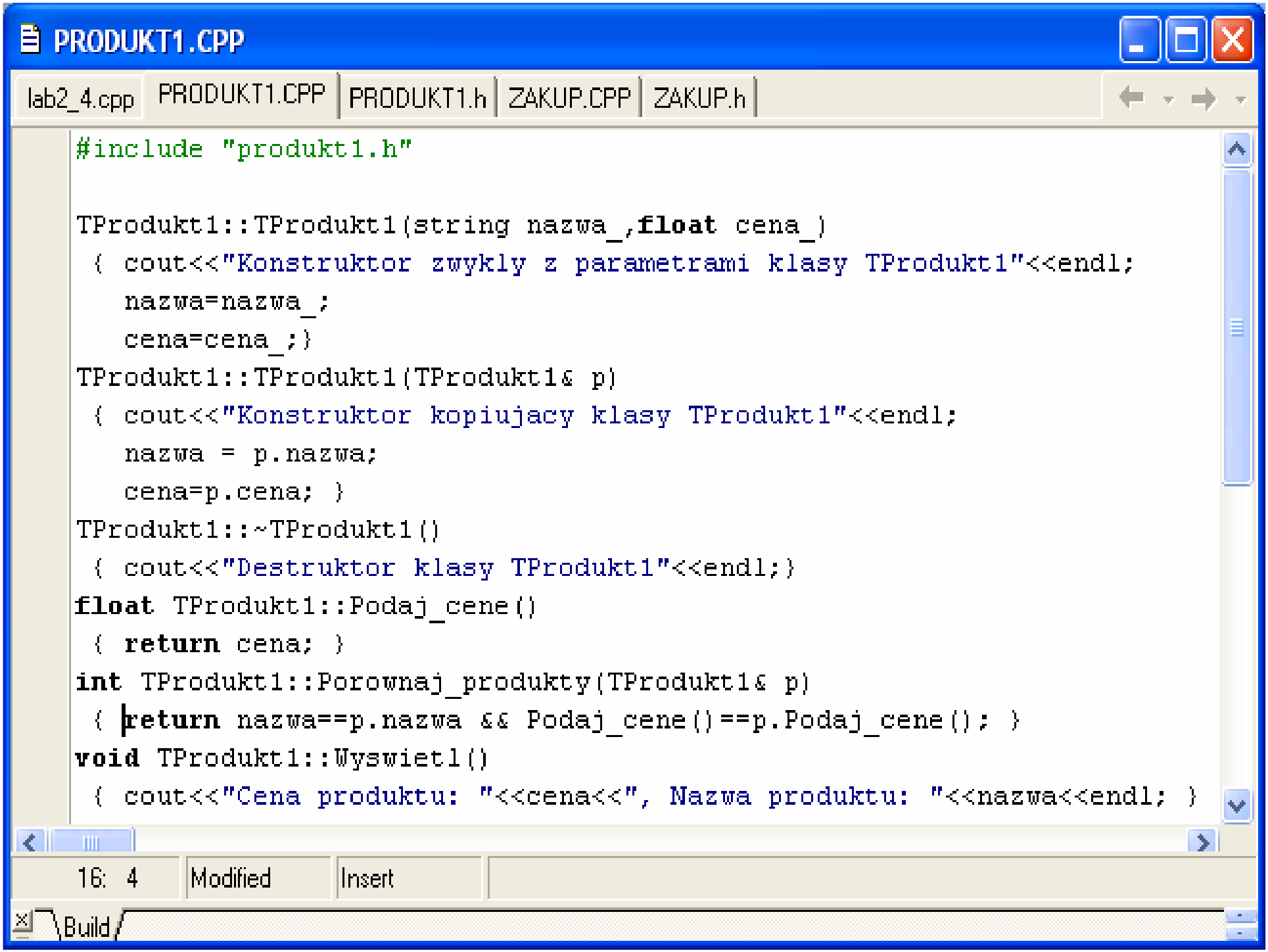
Obiekt posiadający składowe typu obiektowego (agregacja mocna) jest usuwany następująco:

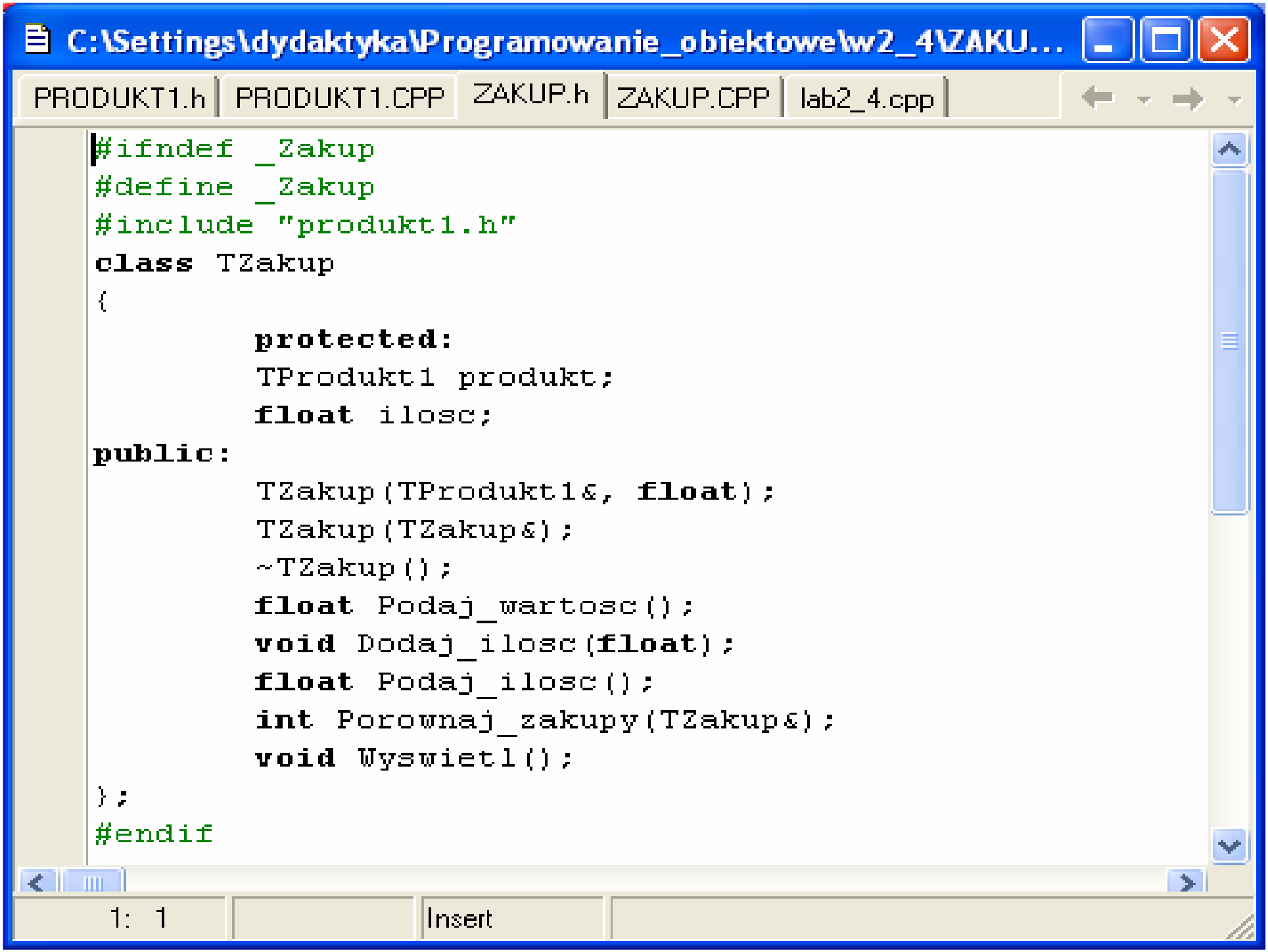
1. Najpierw jest usuwany obiekt zawierający składowe – wywołany jest destruktor klasy tego obiektu
2. Na końcu są usuwane obiekty składowe w odwrotnej kolejności do ich tworzenia– wywołane są destruktory klas tych obiektów

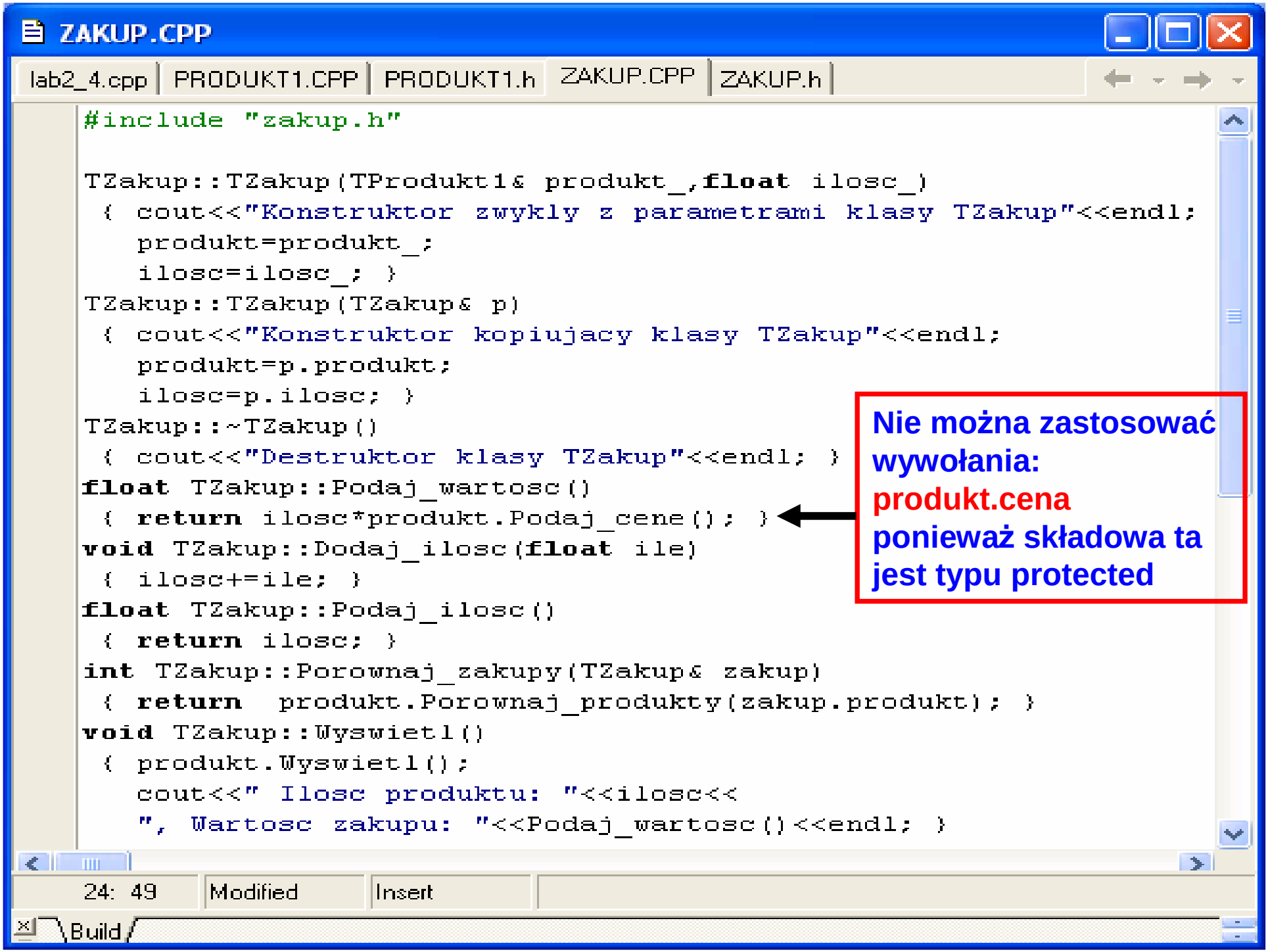
Klasa złożona TZakup agreguje mocno obiekt klasy TProdukt1




C:\Settings\dydaktyka\Programowanie_obiektowe\w2_4...
lab2_4.cpp
PRODUKT1.CPP
PRODUKT1.h
ZAKUP.CPP
ZAKUP.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 Porownaj_produkty(TProdukt1&);
void Wyswietl();
};
#endif
5: 1
Modified
Insert
Build


PRODUKT1.CPP
lab2_4.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; }
16: 4
Modified
Insert
Build


C:\Settings\dydaktyka\Programowanie_obiektowe\w2_4\ZAKU...
PRODUKT1.h
PRODUKT1.CPP
ZAKUP.h
ZAKUP.CPP
lab2_4.cpp
#ifndef _Zakup
#define _Zakup
#include "produkt1.h"
class TZakup
{
protected:
TProdukt1 produkt;
float ilosc;
public:
TZakup(TProdukt1&, float);
TZakup(TZakup&);
~TZakup();
float Podaj_wartosc();
void Dodaj_ilosc(float);
float Podaj_ilosc();
int Porownaj_zakupy(TZakup&);
void Wyswietl();
};
#endif
1: 1
Insert


ZAKUP.CPP
lab2_4.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.Podaj_cene(); }
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; }
Nie można zastosować wywołania:
produkt.cena
ponieważ składowa ta jest typu protected
24: 49
Modified
Insert
Build

```
#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.Podaj_cene(); }

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; }
```

Obiekt jako atrybut klasy agregującej udostępnia jej jedynie składowe publiczne

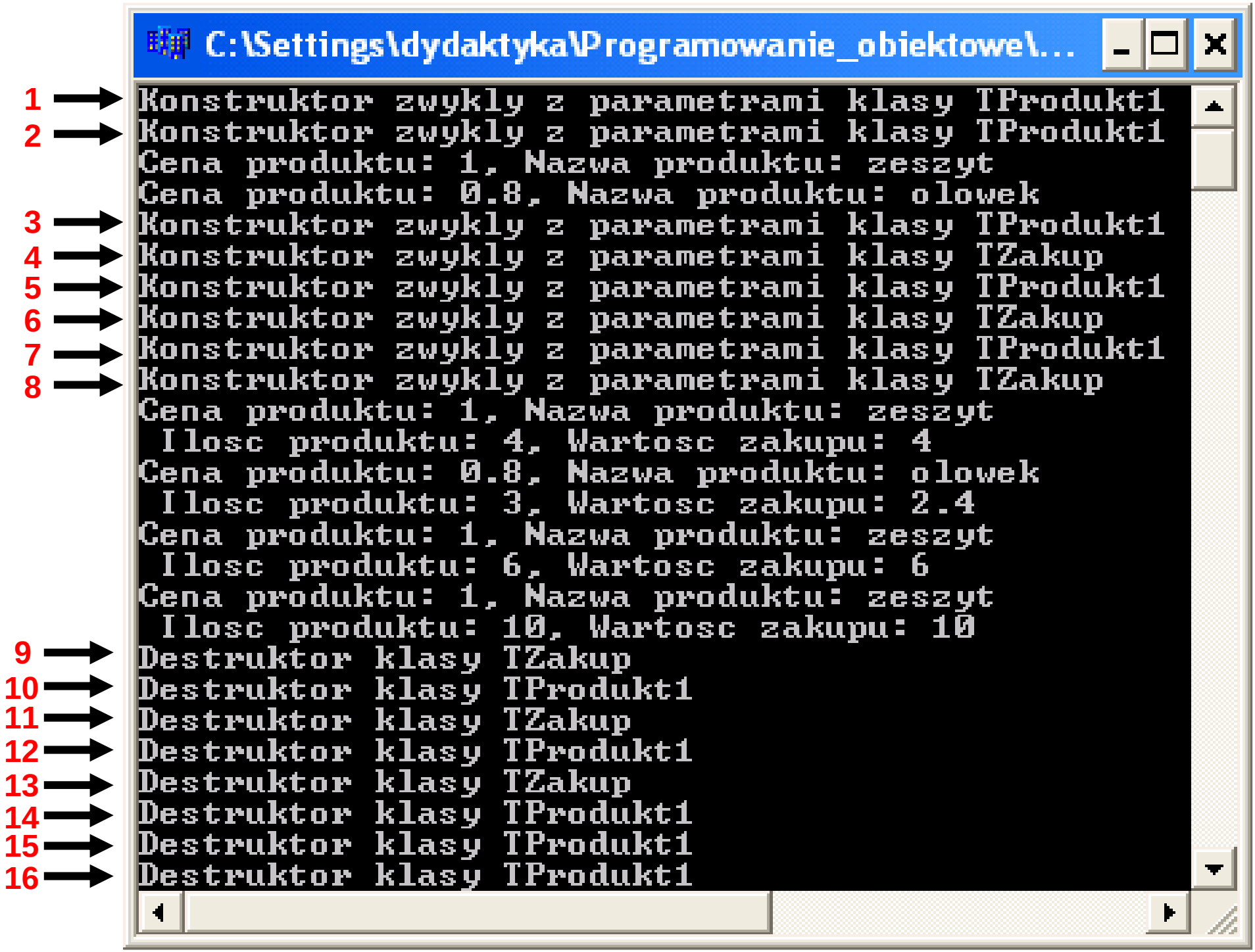
lab2_4.cpp

lab2_4.cpp | PRODUKT1.CPP | PRODUKT1.h | ZAKUP.CPP | ZAKUP.h

```
#include "zakup.h"
//Przyklad1
//program nie zawiera operacji na plikach dyskowych oraz nie posiada menu

void main()
{
 {TProdukt1 p1("zeszyt", 1.0);     //1
  TProdukt1 p2 ("olowek",0.80);    //2
  p1.Wyswietl();
  p2.Wyswietl();
  //kazdy obiekt TZakup musi posiadać wlasny obiekt TProdukt..
  TZakup z1(p1,4);  //3, 4
  TZakup z2(p2,3);  //5 ,6
  TZakup z3(p1,6);  //7, 8
  z1.Wyswietl();
  z2.Wyswietl();
  z3.Wyswietl();
  if(z1.Porownaj_zakupy(z2))
   z1.Dodaj_ilosc(z2.Podaj_ilosc());
  if(z1.Porownaj_zakupy(z3))
   z1.Dodaj_ilosc(z3.Podaj_ilosc());
  z1.Wyswietl();
  }// 9, 10, 11, 12, 13, 14, 15, 16
  cin.get();
}
```

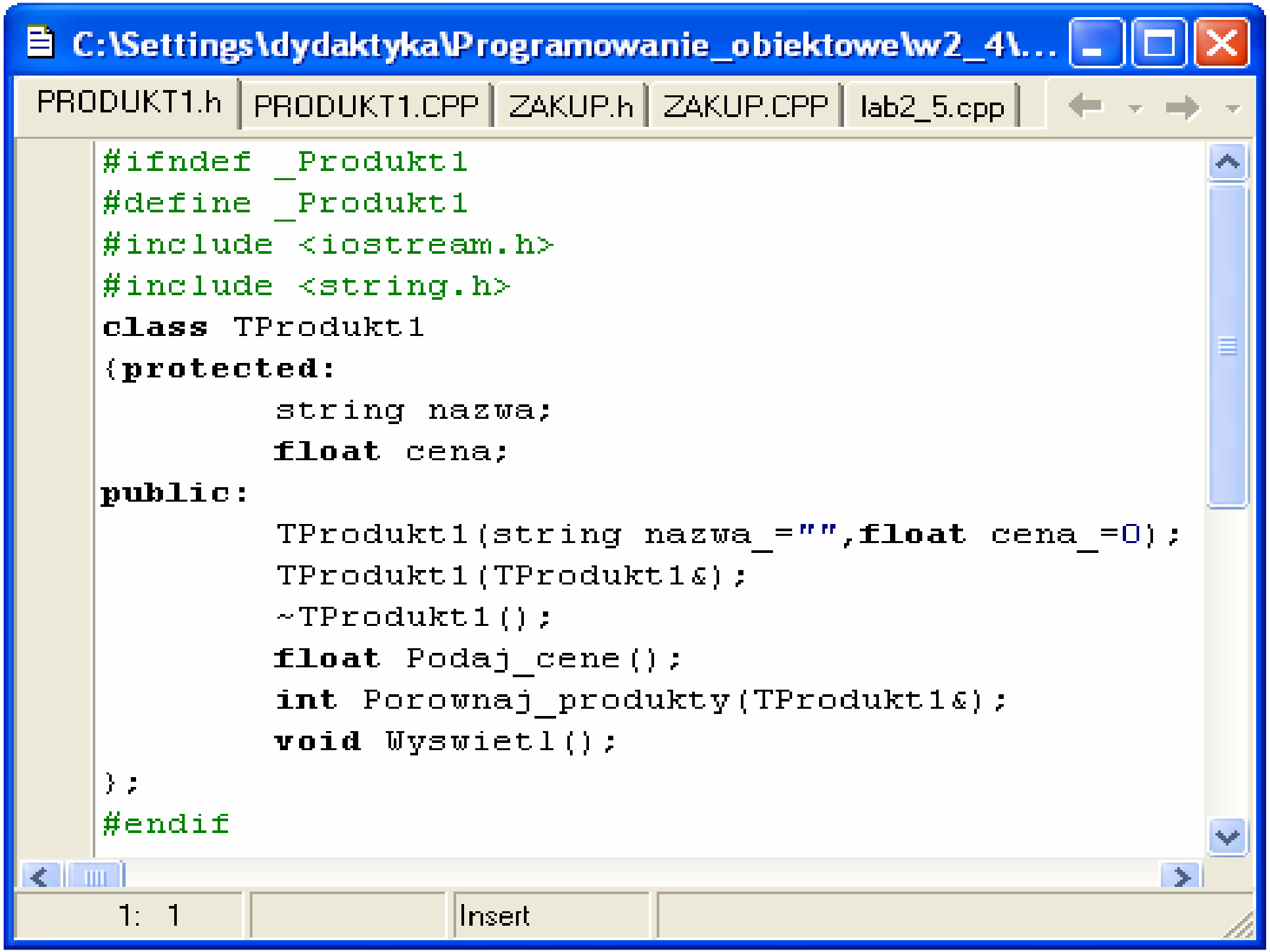
24: 3 | Modified | Insert

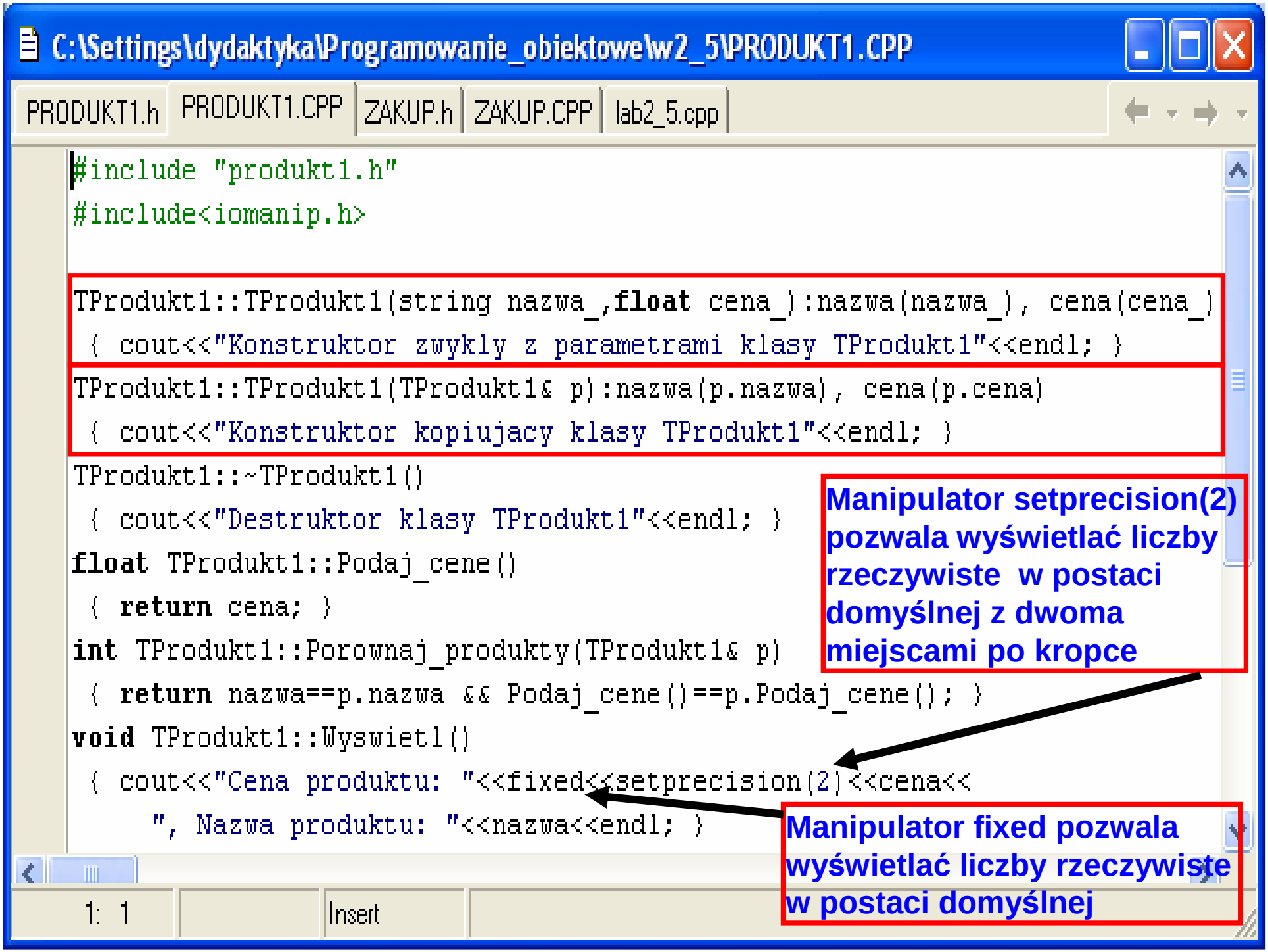
Build

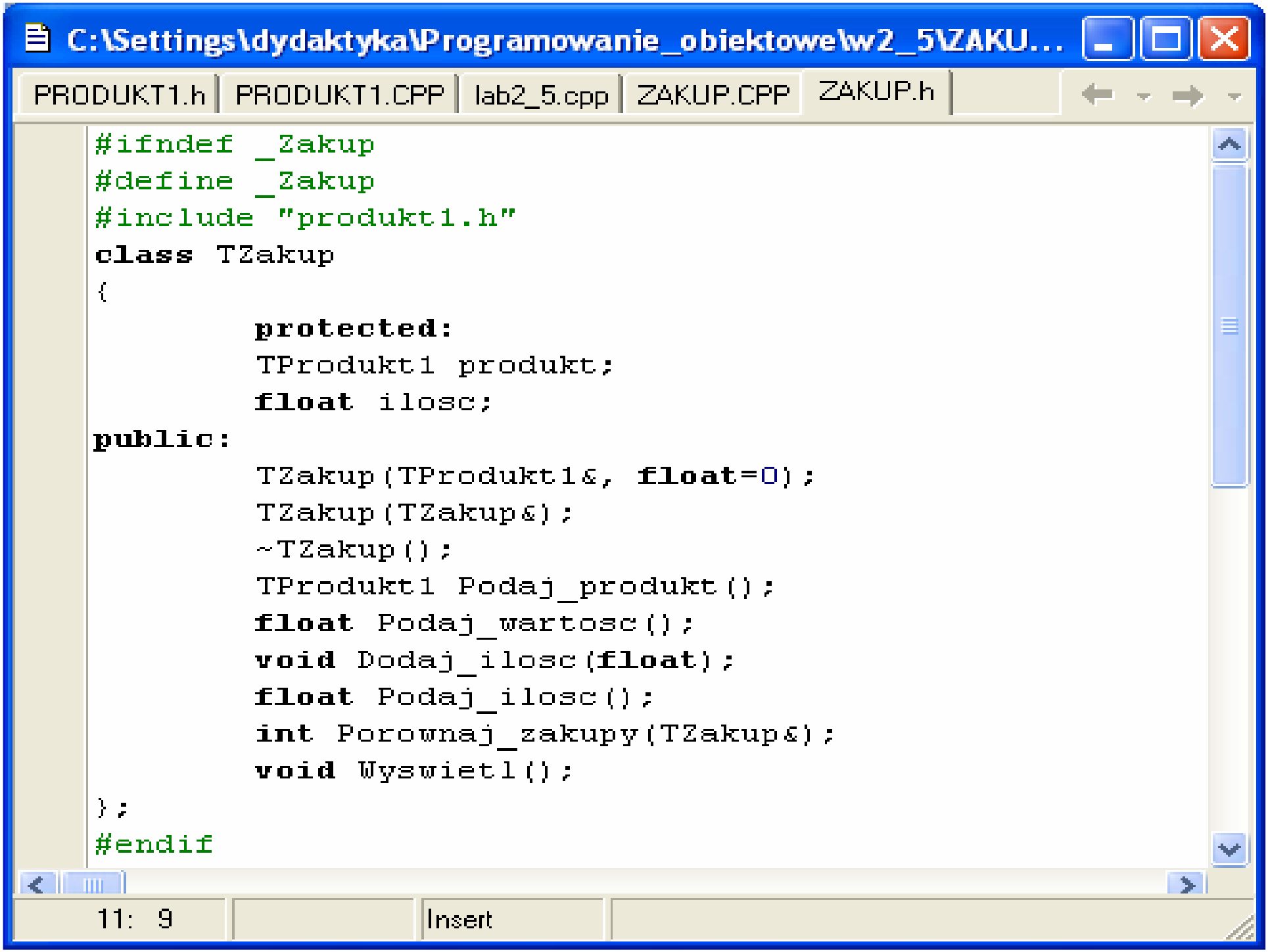

C:\Settings\dydaktyka\Programowanie_obiektowe\...
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TProdukt1
Cena produktu: 1, Nazwa produktu: zeszyt
Cena produktu: 0.8, Nazwa produktu: olowek
Konstruktor zwykly z parametrami 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

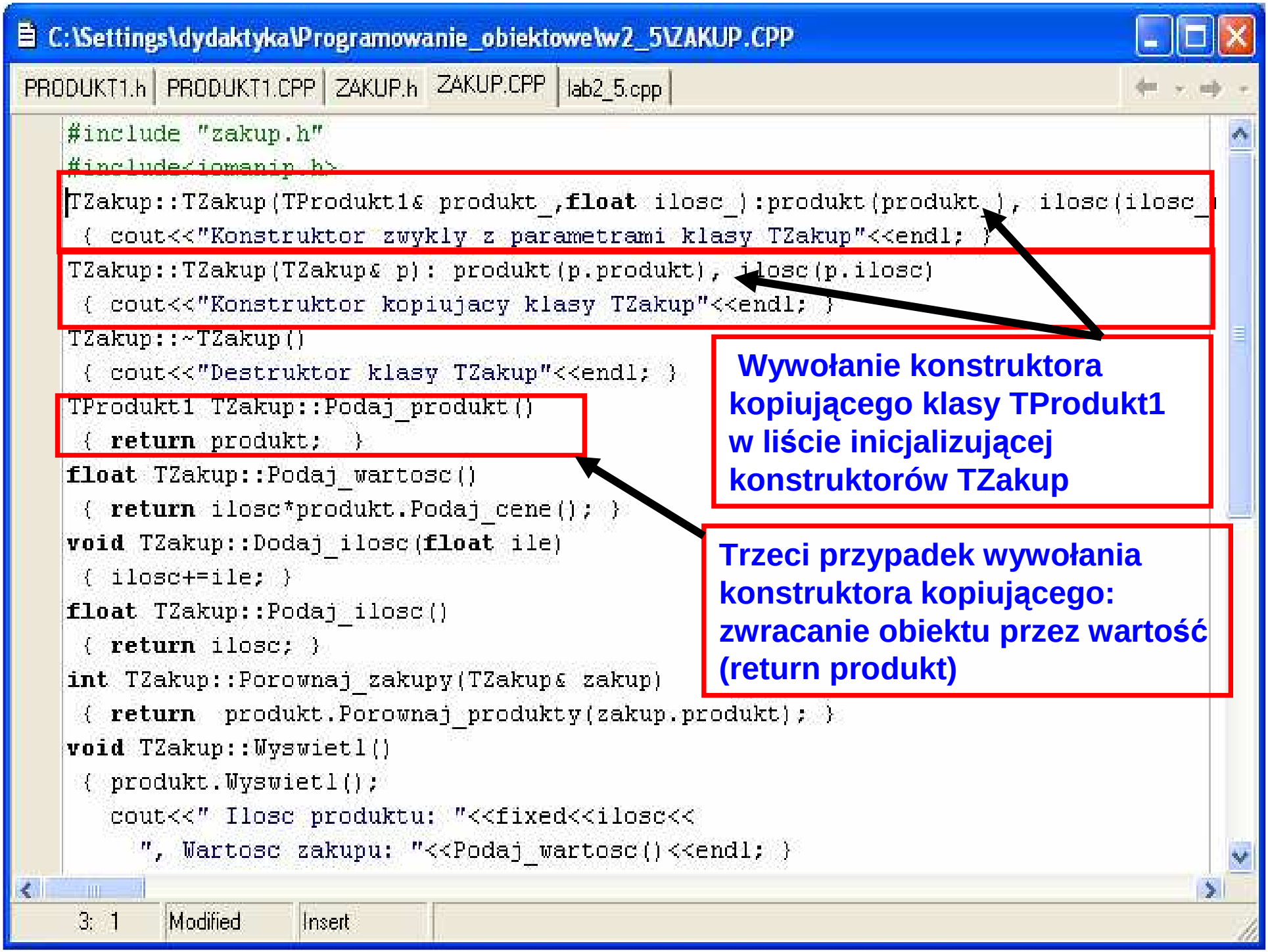
Tworzenie obiektów prostych i złożonych

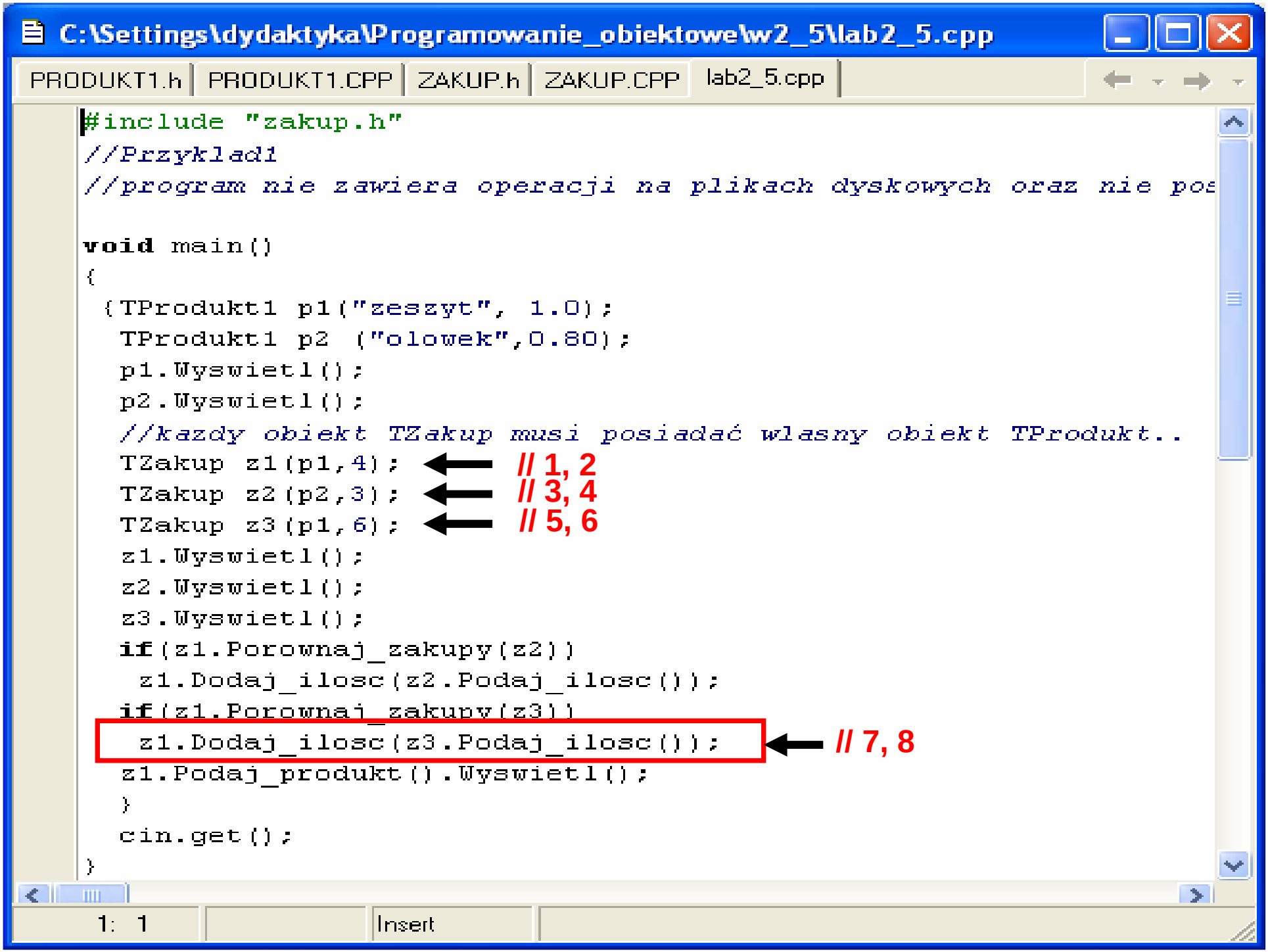
1. Przeciążanie funkcji składowych, przekazywanie parametrów obiektowych przez wartość, referencję oraz wskaźnik
2. Przeciążanie konstruktora - konstruktor bezparametrowy i z parametrami
3. Konstruktor z parametrami domniemanymi
4. Lista inicjalizująca konstruktora
5. Obiekty z atrybutami obiektowym – obiekty klas złożonych, obiekty klas agregujących
6. Zwracanie obiektów przez metodę, lista inicjalizująca konstruktora klasy z atrybutami obiektowymi

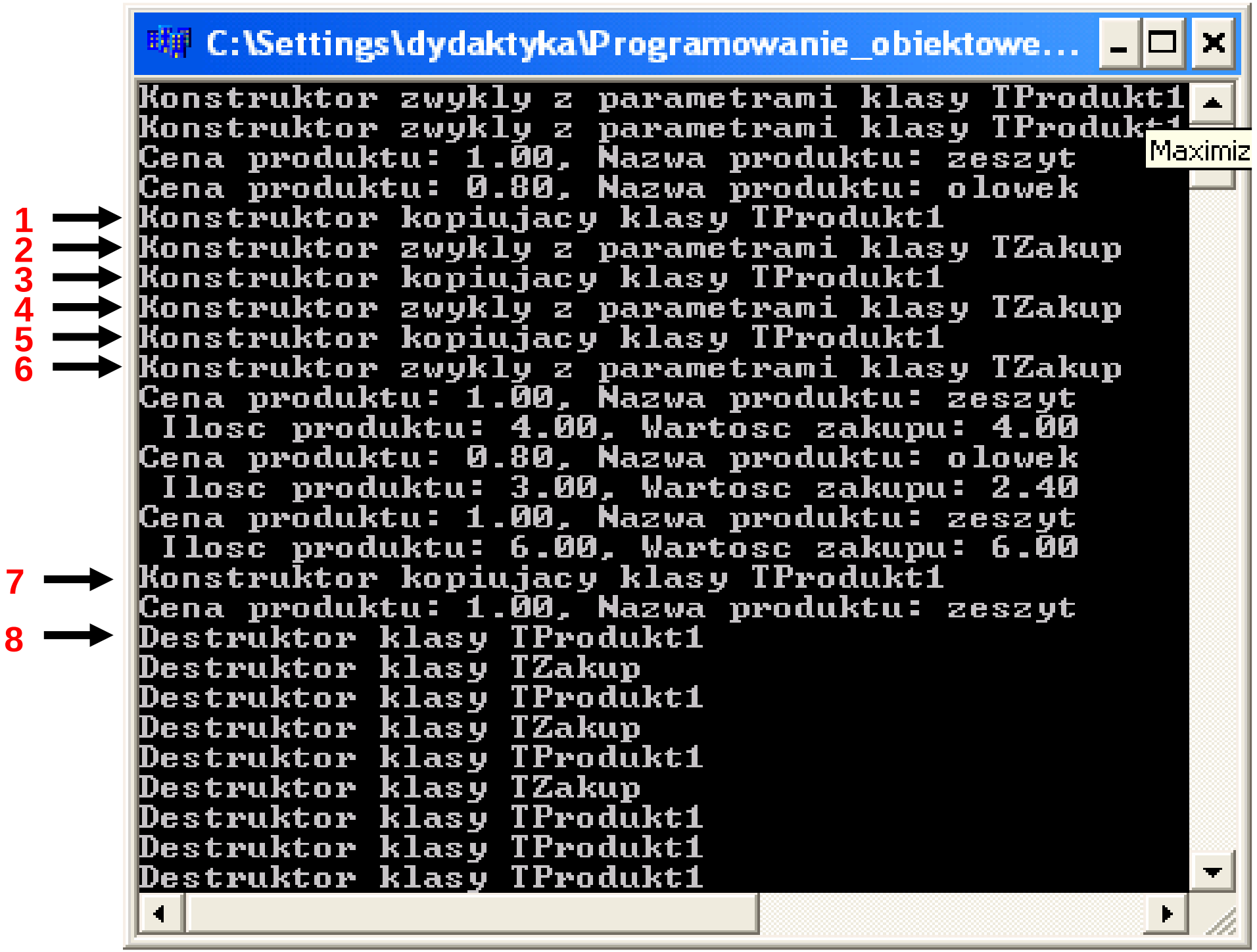

C:\Settings\dydaktyka\Programowanie_obiektowe\w2_4\...
PRODUKT1.h
PRODUKT1.CPP
ZAKUP.h
ZAKUP.CPP
lab2_5.cpp
#ifndef _Produkt1
#define _Produkt1
#include <iostream.h>
#include <string.h>
class TProdukt1
{protected:
string nazwa;
float cena;
public:
TProdukt1(string nazwa_="",float cena_=0);
TProdukt1(TProdukt1&);
~TProdukt1();
float Podaj_cene();
int Porownaj_produkty(TProdukt1&);
void Wyswietl();
};
#endif
1: 1
Insert


C:\Settings\dydaktyka\Programowanie_obiektowe\w2_5\PRODUKT1.CPP
PRODUKT1.h
PRODUKT1.CPP
ZAKUP.h
ZAKUP.CPP
lab2_5.cpp
#include "produkt1.h"
#include<iomanip.h>
TProdukt1::TProdukt1(string nazwa_,float cena_):nazwa(nazwa_), cena(cena_)
{ cout<<"Konstruktor zwykly z parametrami klasy TProdukt1"<<endl; }
TProdukt1::TProdukt1(TProdukt1& p):nazwa(p.nazwa), cena(p.cena)
{ cout<<"Konstruktor kopiujacy klasy TProdukt1"<<endl; }
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: "<<fixed<<setprecision(2)<<cena<<
", Nazwa produktu: "<<nazwa<<endl; }
Manipulator setprecision(2) pozwala wyświetlać liczby rzeczywiste w postaci domyślnej z dwoma miejscami po kropce
Manipulator fixed pozwala wyświetlać liczby rzeczywiste w postaci domyślnej
1: 1
Insert


C:\Settings\dydaktyka\Programowanie_obiektowe\w2_5\ZAKU...
PRODUKT1.h
PRODUKT1.CPP
lab2_5.cpp
ZAKUP.CPP
ZAKUP.h
#ifndef _Zakup
#define _Zakup
#include "produkt1.h"
class TZakup
{
protected:
TProdukt1 produkt;
float ilosc;
public:
TZakup(TProdukt1&, float=0);
TZakup(TZakup&);
~TZakup();
TProdukt1 Podaj_produkt();
float Podaj_wartosc();
void Dodaj_ilosc(float);
float Podaj_ilosc();
int Porownaj_zakupy(TZakup&);
void Wyswietl();
};
#endif
11: 9
Insert


C:\Settings\dydaktyka\Programowanie_obiektowe\w2_5\ZAKUP.CPP
PRODUKT1.h
PRODUKT1.CPP
ZAKUP.h
ZAKUP.CPP
lab2_5.cpp
#include "zakup.h"
#include<iomanip.h>
TZakup::TZakup(TProdukt1& produkt_,float ilosc_):produkt(produkt_), ilosc(ilosc_)
{ cout<<"Konstruktor zwykly z parametrami klasy TZakup"<<endl; }
TZakup::TZakup(TZakup& p): produkt(p.produkt), ilosc(p.ilosc)
{ cout<<"Konstruktor kopiujacy klasy TZakup"<<endl; }
TZakup::~TZakup()
{ cout<<"Destruktor klasy TZakup"<<endl; }
TProdukt1 TZakup::Podaj_produkt()
{ return produkt; }
float TZakup::Podaj_wartosc()
{ return ilosc*produkt.Podaj_cene(); }
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: "<<fixed<<ilosc<<
", Wartosc zakupu: "<<Podaj_wartosc()<<endl; }
Wywołanie konstruktora kopiującego klasy TProdukt1 w liście inicjalizującej konstruktorów TZakup
Trzeci przypadek wywołania konstruktora kopiującego: zwracanie obiektu przez wartość (return produkt)
3: 1
Modified
Insert


C:\Settings\dydaktyka\Programowanie_obiektowe\w2_5\lab2_5.cpp
PRODUKT1.h
PRODUKT1.CPP
ZAKUP.h
ZAKUP.CPP
lab2_5.cpp
#include "zakup.h"
//Przyklad1
//program nie zawiera operacji na plikach dyskowych oraz nie pos
void main()
{
{TProdukt1 p1("zeszyt", 1.0);
TProdukt1 p2 ("olowek",0.80);
p1.Wyswietl();
p2.Wyswietl();
//kazdy obiekt TZakup musi posiadać wlasny obiekt TProdukt..
TZakup z1(p1,4);
// 1, 2
TZakup z2(p2,3);
// 3, 4
TZakup z3(p1,6);
// 5, 6
z1.Wyswietl();
z2.Wyswietl();
z3.Wyswietl();
if(z1.Porownaj_zakupy(z2))
z1.Dodaj_ilosc(z2.Podaj_ilosc());
if(z1.Porownaj_zakupy(z3))
z1.Dodaj_ilosc(z3.Podaj_ilosc());
// 7, 8
z1.Podaj_produkt().Wyswietl();
}
cin.get();
}
1: 1
Insert


C:\Settings\dydaktyka\Programowanie_obiektowe...
Konstruktor zwykly z parametrami klasy TProdukt1
Konstruktor zwykly z parametrami klasy TProdukt1
Cena produktu: 1.00, Nazwa produktu: zeszyt
Cena produktu: 0.80, Nazwa produktu: olowek
1 Konstruktor kopiujacy klasy TProdukt1
2 Konstruktor zwykly z parametrami klasy TZakup
3 Konstruktor kopiujacy klasy TProdukt1
4 Konstruktor zwykly z parametrami klasy TZakup
5 Konstruktor kopiujacy klasy TProdukt1
6 Konstruktor zwykly z parametrami klasy TZakup
Cena produktu: 1.00, Nazwa produktu: zeszyt
Ilosc produktu: 4.00, Wartosc zakupu: 4.00
Cena produktu: 0.80, Nazwa produktu: olowek
Ilosc produktu: 3.00, Wartosc zakupu: 2.40
Cena produktu: 1.00, Nazwa produktu: zeszyt
Ilosc produktu: 6.00, Wartosc zakupu: 6.00
7 Konstruktor kopiujacy klasy TProdukt1
Cena produktu: 1.00, Nazwa produktu: zeszyt
8 Destruktor klasy TProdukt1
Destruktor klasy TZakup
Destruktor klasy TProdukt1
Destruktor klasy TZakup
Destruktor klasy TProdukt1
Destruktor klasy TZakup
Destruktor klasy TProdukt1
Destruktor klasy TProdukt1
Destruktor klasy TProdukt1
Maximiz