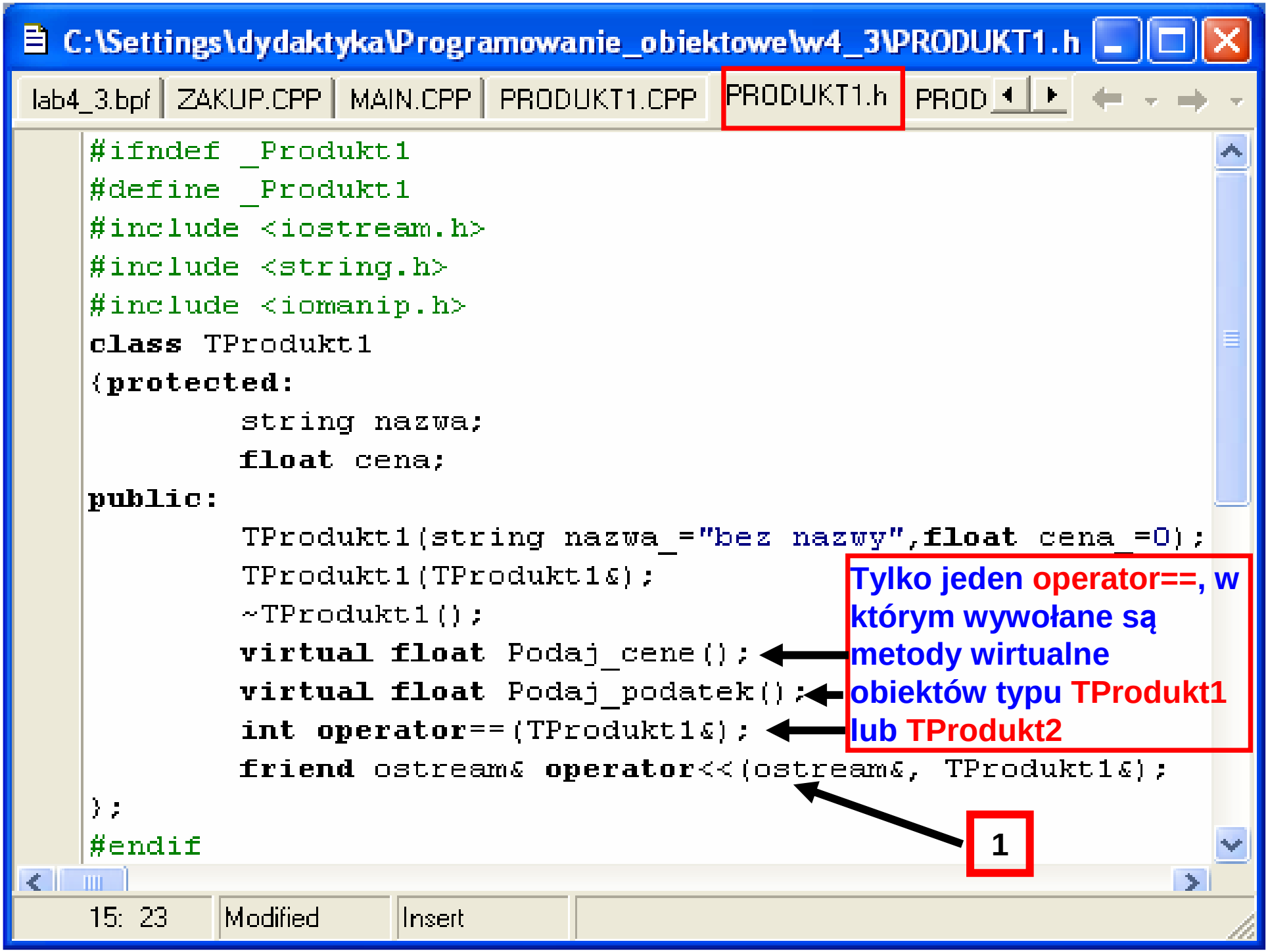
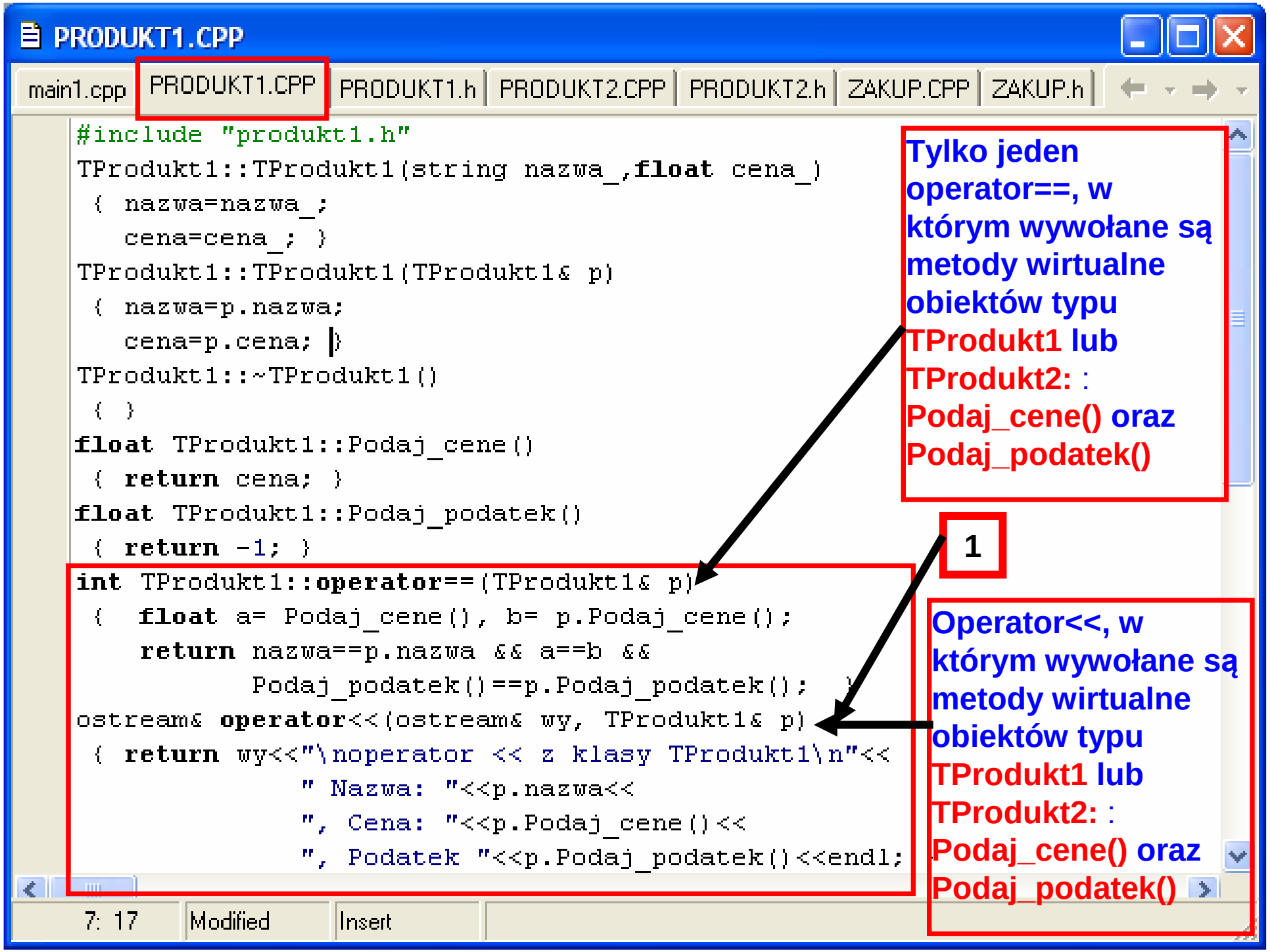
Dziedziczenie jednobazowe, poliformizm, tablice wskaźników na obiekty

1. **Polimorfizm (1) – tablice wskaźników na obiekty**
2. **Polimorfizm (2) – tablice wskaźników na obiekty**
3. **Polimorfizm (3) – tablice wskaźników na unikatowe obiekty, klasa abstrakcyjna**
4. **Klasa obsługująca uniwersalną tablicę zawierającą wskaźniki na unikatowe obiekty**

Dziedziczenie jednobazowe, poliformizm

1. Polimorfizm (1) – tablice wskaźników na obiekty


C:\Settings\dydaktyka\Programowanie_obiektowe\w4_3\PRODUKT1.h
lab4_3.bpf
ZAKUP.CPP
MAIN.CPP
PRODUKT1.CPP
PRODUKT1.h
PROD
#ifndef _Produkt1
#define _Produkt1
#include <iostream.h>
#include <string.h>
#include <iomanip.h>
class TProdukt1
{protected:
string nazwa;
float cena;
public:
TProdukt1(string nazwa_="bez nazwy",float cena_=0);
TProdukt1(TProdukt1&);
~TProdukt1();
virtual float Podaj_cene();
virtual float Podaj_podatek();
int operator==(TProdukt1&);
friend ostream& operator<<(ostream&, TProdukt1&);
};
#endif
Tylko jeden operator==, w którym wywołane są metody wirtualne obiektów typu TProdukt1 lub TProdukt2
1
15: 23
Modified
Insert


PRODUKT1.CPP
main1.cpp
PRODUKT1.CPP
PRODUKT1.h
PRODUKT2.CPP
PRODUKT2.h
ZAKUP.CPP
ZAKUP.h
#include "produkt1.h"
TProdukt1::TProdukt1(string nazwa_,float cena_)
{ nazwa=nazwa_;
cena=cena_; }
TProdukt1::TProdukt1(TProdukt1& p)
{ nazwa=p.nazwa;
cena=p.cena; }
TProdukt1::~TProdukt1()
{ }
float TProdukt1::Podaj_cene()
{ return cena; }
float TProdukt1::Podaj_podatek()
{ return -1; }
int TProdukt1::operator==(TProdukt1& p)
{ float a= Podaj_cene(), b= p.Podaj_cene();
return nazwa==p.nazwa && a==b &&
Podaj_podatek()==p.Podaj_podatek(); }
ostream& operator<<(ostream& wy, TProdukt1& p)
{ return wy<<"\noperator << z klasy TProdukt1\n"<<
" Nazwa: "<<p.nazwa<<
", Cena: "<<p.Podaj_cene()<<
", Podatek "<<p.Podaj_podatek()<<endl;
Tylko jeden operator==, w którym wywołane są metody wirtualne obiektów typu TProdukt1 lub TProdukt2: : Podaj_cene() oraz Podaj_podatek()
1
Operator<<, w którym wywołane są metody wirtualne obiektów typu TProdukt1 lub TProdukt2: : Podaj_cene() oraz Podaj_podatek()
7: 17
Modified
Insert

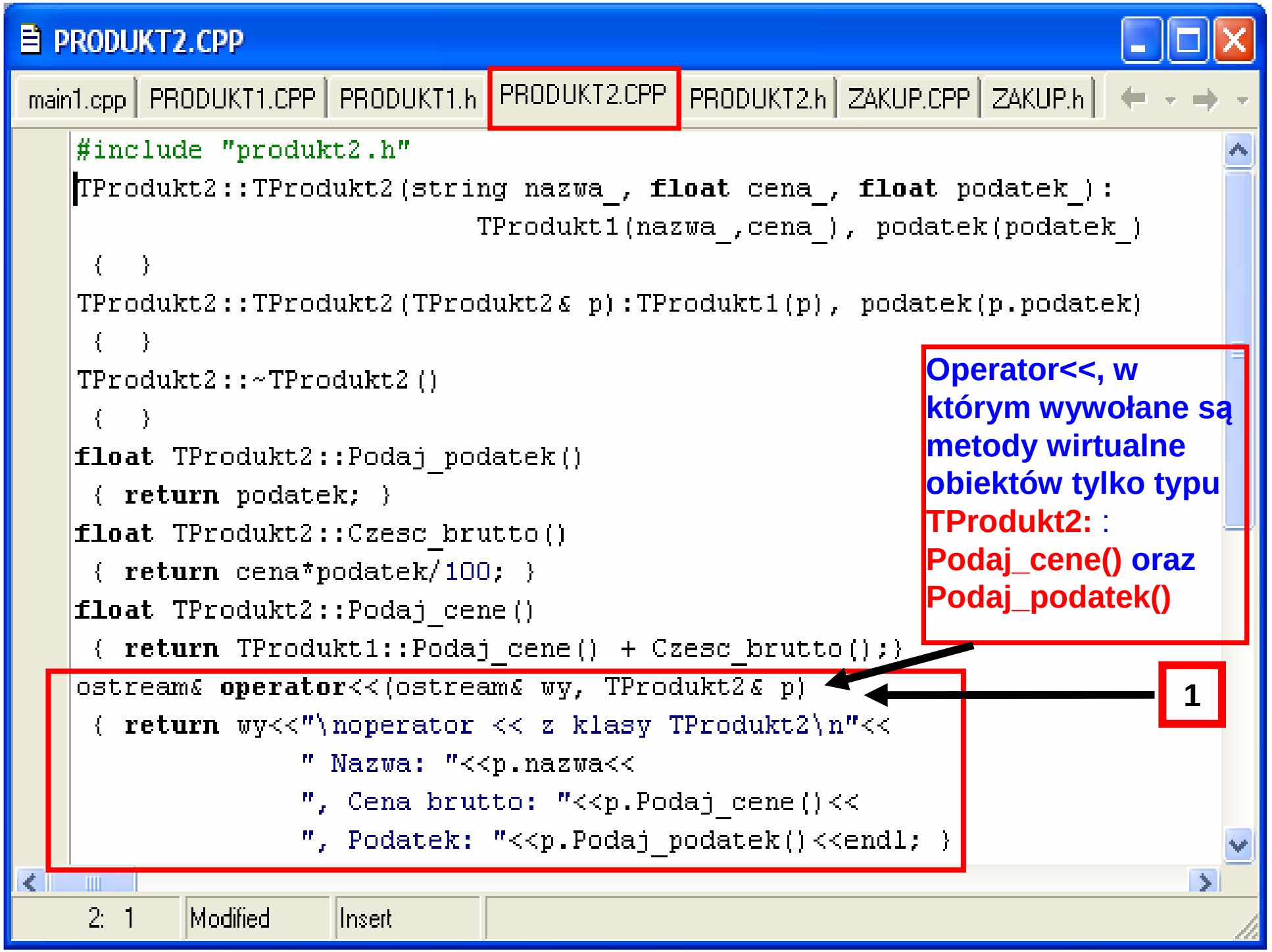
C:\Settings\dydaktyka\Programowanie_obiektowe\w4_3\PRODUKT2.h

lab4_3.bpf | ZAKUP.CPP | MAIN.CPP | PRODUKT1.CPP | PRODUKT1.h | PRODUKT2.CPP | PRODUKT2.h

```
#ifndef _Produkt2
#define _Produkt2
#include "produkt1.h"

class TProdukt2: public TProdukt1
{  protected:
        float podatek;
public:
        TProdukt2(string nazwa_="bez nazwy",float cena_=0, float podatek=0);
        TProdukt2(TProdukt2&);
        ~TProdukt2();
        float Czesc_brutto();
        float Podaj_cene();
        float Podaj_podatek();
        friend ostream& operator<<(ostream&, TProdukt2&);
};
#endif
```

2

14: 31 | Modified | Insert


PRODUKT2.CPP
main1.cpp
PRODUKT1.CPP
PRODUKT1.h
PRODUKT2.CPP
PRODUKT2.h
ZAKUP.CPP
ZAKUP.h
#include "produkt2.h"
TProdukt2::TProdukt2(string nazwa_, float cena_, float podatek_):
TProdukt1(nazwa_,cena_), podatek(podatek_)
{ }
TProdukt2::TProdukt2(TProdukt2& p):TProdukt1(p), podatek(p.podatek)
{ }
TProdukt2::~TProdukt2()
{ }
float TProdukt2::Podaj_podatek()
{ return podatek; }
float TProdukt2::Czesc_brutto()
{ return cena*podatek/100; }
float TProdukt2::Podaj_cene()
{ return TProdukt1::Podaj_cene() + Czesc_brutto();}
ostream& operator<<(ostream& wy, TProdukt2& p)
{ return wy<<"\noperator << z klasy TProdukt2\n"<<
" Nazwa: "<<p.nazwa<<
", Cena brutto: "<<p.Podaj_cene()<<
", Podatek: "<<p.Podaj_podatek()<<endl; }
Operator<<, w którym wywołane są metody wirtualne obiektów tylko typu TProdukt2: : Podaj_cene() oraz Podaj_podatek()
1
2: 1
Modified
Insert

main1.cpp

lab6_1.bpf | main1.cpp

```
#include "Zakup.h"
const int N=5;
void main()
{
 TProdukt1* produkty[N];
 TProdukt1* p1=new TProdukt1("zeszyt", 1.0);      produkty[0] =p1; cout<<*p1;
 TProdukt1* p2=new TProdukt1("zeszyt", 2.0);      produkty[1] =p2; cout<<*p2;
 TProdukt2* p3=new TProdukt2 ("olowek",0.80,7);   produkty[2] =p3; cout<<*p3;
 TProdukt2* p4=new TProdukt2("pioro",4.80,20);    produkty[3] =p4; cout<<*p4;
 TProdukt2* p5=new TProdukt2("dlugopis",6.80,22); produkty[4] =p5; cout<<*p5;

 for (int i=0; i<N;i++)
    cout<<*produkty[i];
 for (int i=0; i<N;i++)
   delete produkty[i];
 cin.get();
}
```

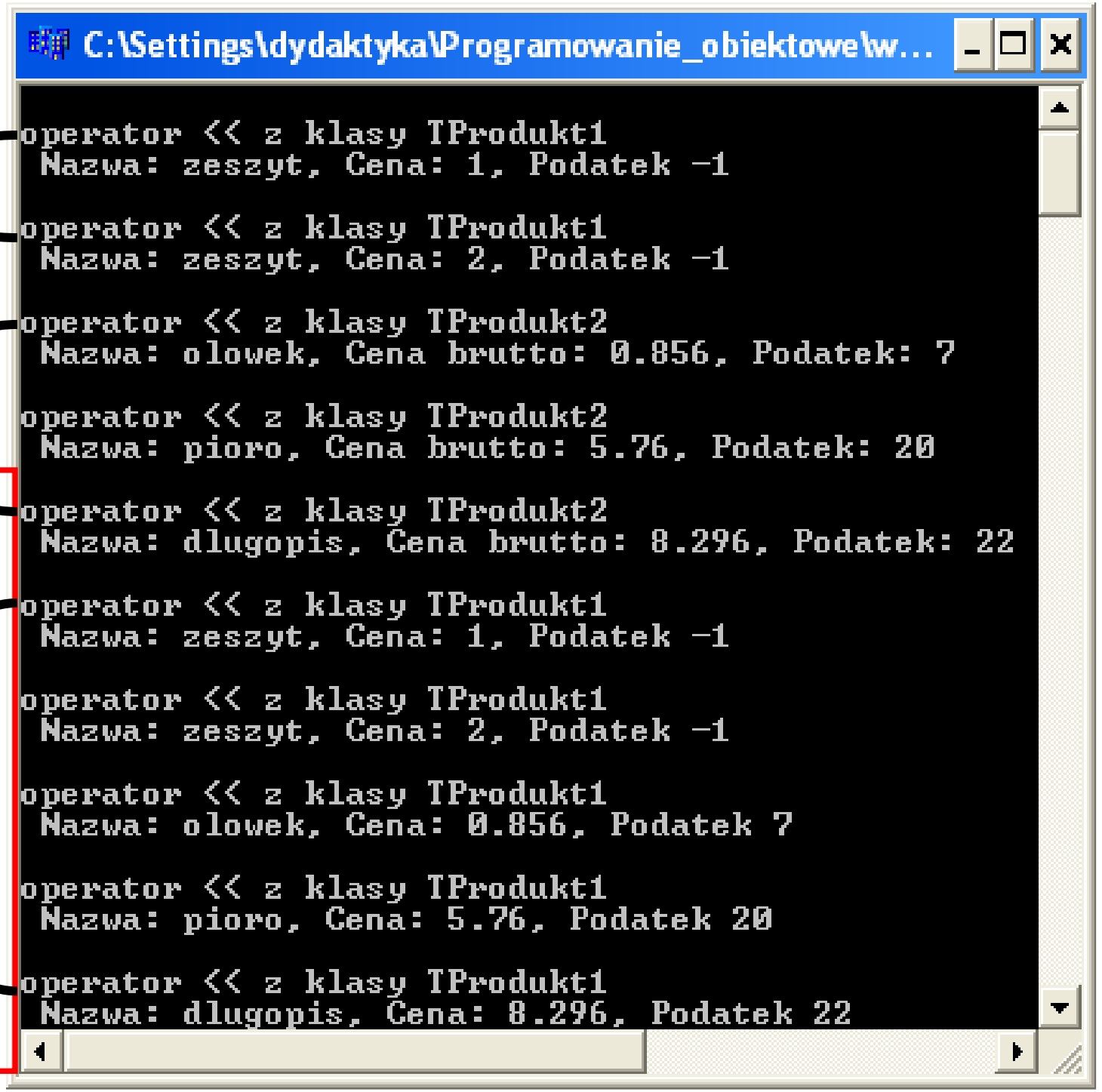
Wywoływany operator<< z klasy **TProdukt1**, natomiast w ciele tej funkcji operatorowej wywoływane są wirtualne metody ***Podaj_cenę*** oraz ***Podaj _podatek***, stąd na ekranie pokazują się poprawne wyniki. Jednak podatek równy -1 przydatny okazał się jedynie w metodzie operatorowej operator==, służącej do porównania obiektów

15: 43 | Modified | Insert

1 – operator << z klasy TProdukt1

2 – operator<< z klasy TProdukt2

1 - operator<< z klasyTProdukt1. Poprawnie drukowana cena i podatek dzięki wirtualnym metodom wywoływanym w ciele tej funkcji. *Jednak podatek równy -1 jest wartością nienaturalną*



Dziedziczenie jednobazowe, poliformizm

1. **Polimorfizm (1) – tablice wskaźników na obiekty**
2. **Polimorfizm (2) – tablice wskaźników na obiekty**

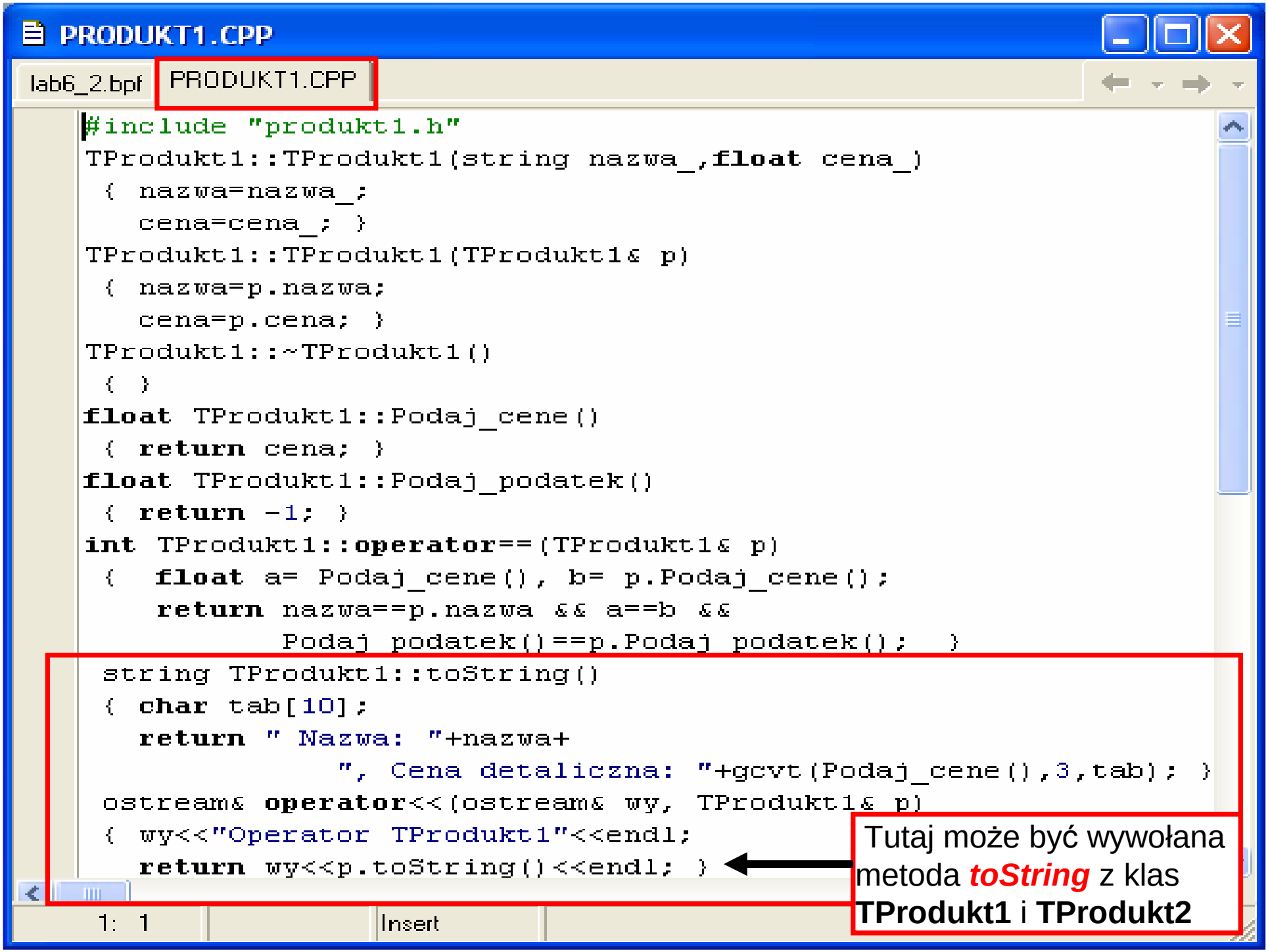
C:\Settings\dydaktyka\Programowanie_obiektowe\w6_2\PRODUKT1.h

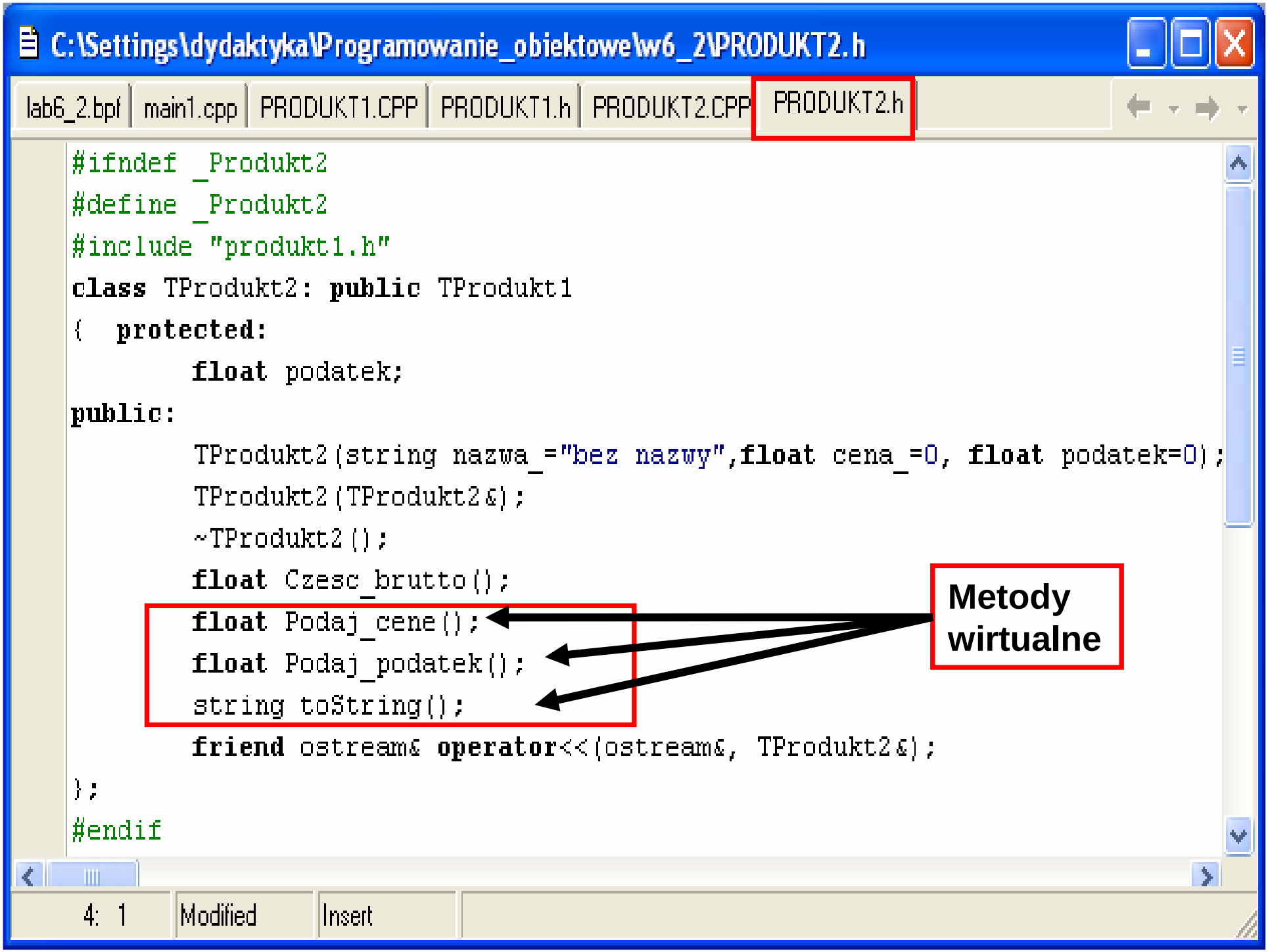
lab6_2.bpf | main1.cpp | PRODUKT1.CPP | PRODUKT1.h | PRODUKT2.CPP | PF

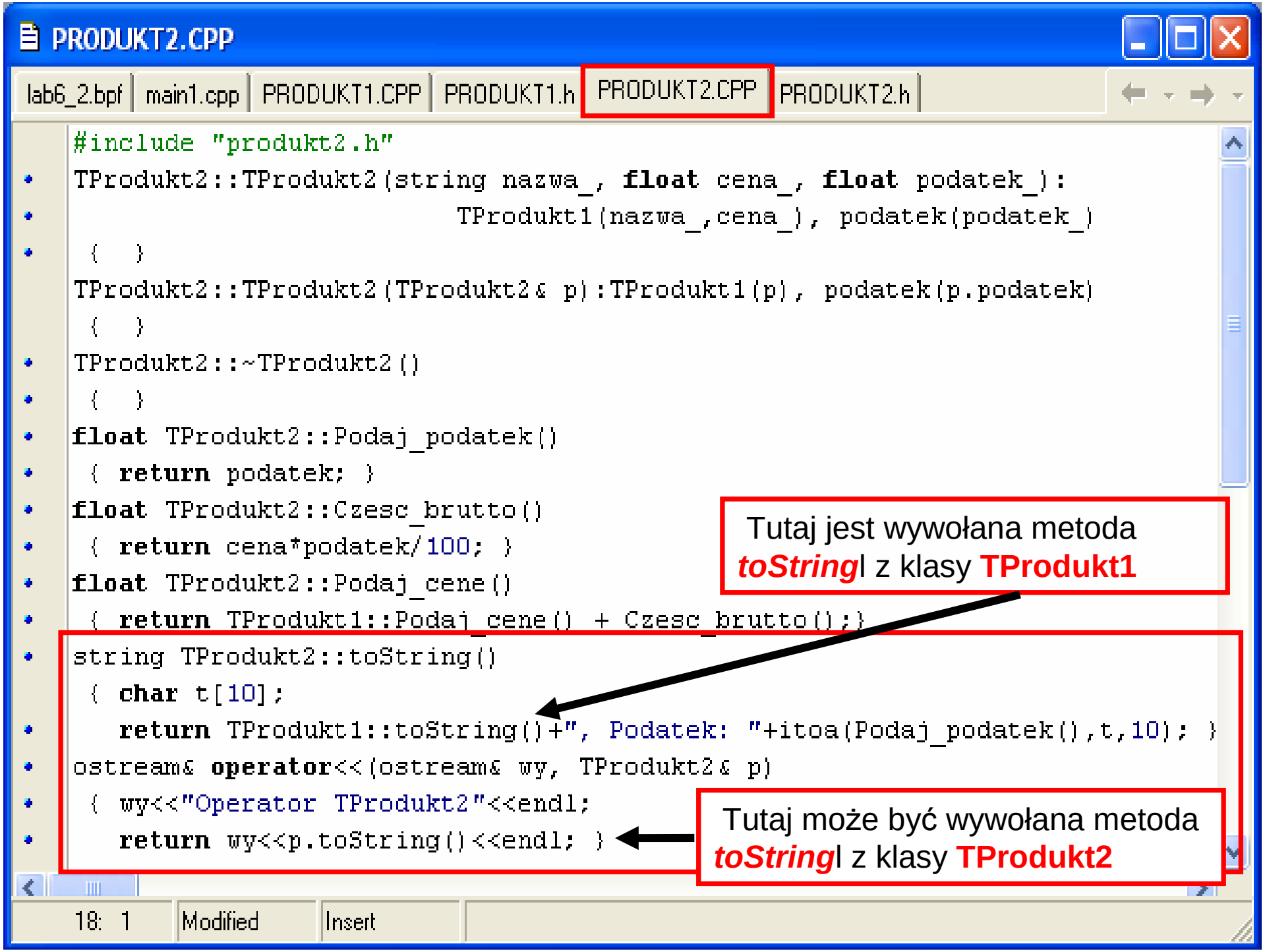
```
#ifndef _Produkt1
#define _Produkt1
#include <iostream.h>
#include <string.h>
#include <iomanip.h>
#include <stdlib.h>
class TProdukt1
{protected:
        string nazwa;
        float cena;
public:
        TProdukt1(string nazwa_="bez nazwy",float cena_=0);
        TProdukt1(TProdukt1&);
        ~TProdukt1();
        virtual float Podaj_cene();
        virtual float Podaj_podatek();
        int operator==(TProdukt1&);
        virtual string toString();
        friend ostream& operator<<(ostream&, TProdukt1&);
};
#endif
```

Metody wirtualne

6: 19 | Modified | Insert


PRODUKT1.CPP
lab6_2.bpf
PRODUKT1.CPP
#include "produkt1.h"
TProdukt1::TProdukt1(string nazwa_,float cena_)
{ nazwa=nazwa_;
cena=cena_; }
TProdukt1::TProdukt1(TProdukt1& p)
{ nazwa=p.nazwa;
cena=p.cena; }
TProdukt1::~TProdukt1()
{ }
float TProdukt1::Podaj_cene()
{ return cena; }
float TProdukt1::Podaj_podatek()
{ return -1; }
int TProdukt1::operator==(TProdukt1& p)
{ float a= Podaj_cene(), b= p.Podaj_cene();
return nazwa==p.nazwa && a==b &&
Podaj_podatek()==p.Podaj_podatek(); }
string TProdukt1::toString()
{ char tab[10];
return " Nazwa: "+nazwa+
", Cena detaliczna: "+gcvt(Podaj_cene(),3,tab); }
ostream& operator<<(ostream& wy, TProdukt1& p)
{ wy<<"Operator TProdukt1"<<endl;
return wy<<p.toString()<<endl; }
1: 1
Insert
Tutaj może być wywołana metoda toString z klas TProdukt1 i TProdukt2


C:\Settings\dydaktyka\Programowanie_obiektowe\w6_2\PRODUKT2.h
lab6_2.bpf
main1.cpp
PRODUKT1.CPP
PRODUKT1.h
PRODUKT2.CPP
PRODUKT2.h
#ifndef _Produkt2
#define _Produkt2
#include "produkt1.h"
class TProdukt2: public TProdukt1
{ protected:
float podatek;
public:
TProdukt2(string nazwa_="bez nazwy",float cena_=0, float podatek=0);
TProdukt2(TProdukt2&);
~TProdukt2();
float Czesc_brutto();
float Podaj_cene();
float Podaj_podatek();
string toString();
friend ostream& operator<<(ostream&, TProdukt2&);
};
#endif
Metody wirtualne
4: 1
Modified
Insert


PRODUKT2.CPP
lab6_2.bpf
main1.cpp
PRODUKT1.CPP
PRODUKT1.h
PRODUKT2.CPP
PRODUKT2.h
#include "produkt2.h"
TProdukt2::TProdukt2(string nazwa_, float cena_, float podatek_):
TProdukt1(nazwa_,cena_), podatek(podatek_)
{ }
TProdukt2::TProdukt2(TProdukt2& p):TProdukt1(p), podatek(p.podatek)
{ }
TProdukt2::~TProdukt2()
{ }
float TProdukt2::Podaj_podatek()
{ return podatek; }
float TProdukt2::Czesc_brutto()
{ return cena*podatek/100; }
float TProdukt2::Podaj_cene()
{ return TProdukt1::Podaj_cene() + Czesc_brutto();}
string TProdukt2::toString()
{ char t[10];
return TProdukt1::toString()+", Podatek: "+itoa(Podaj_podatek(),t,10); }
ostream& operator<<(ostream& wy, TProdukt2& p)
{ wy<<"Operator TProdukt2"<<endl;
return wy<<p.toString()<<endl; }
Tutaj jest wywołana metoda toStringl z klasy TProdukt1
Tutaj może być wywołana metoda toStringl z klasy TProdukt2
18: 1
Modified
Insert

main1.cpp

lab6_2.bpf | main1.cpp

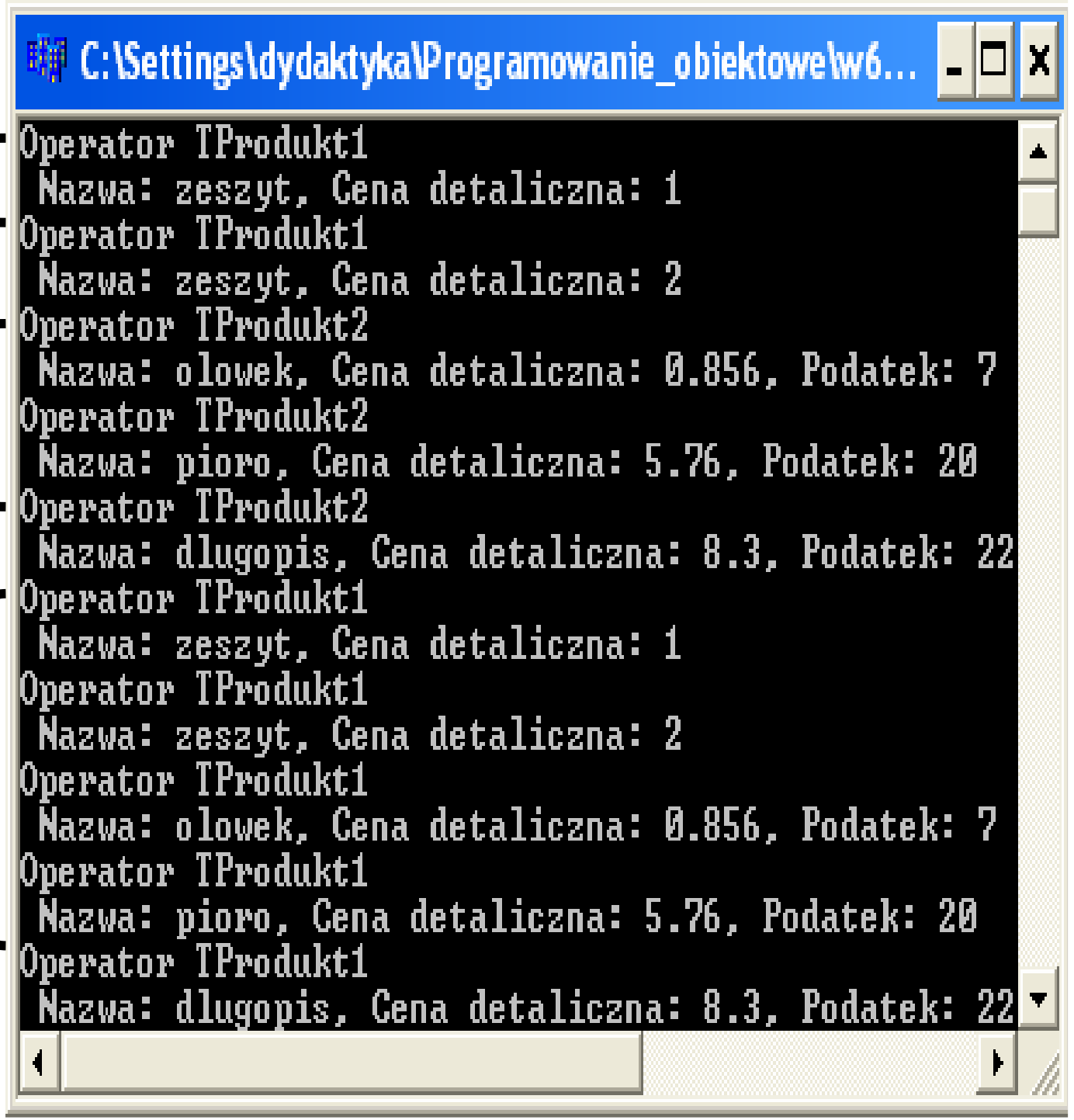
```
#include "Zakup.h"
const int N=5;
void main()
{
 TProdukt1* produkty[N];
 TProdukt1* p1=new TProdukt1("zeszyt", 1.0);       produkty[0] =p1; cout<<*p1;
 TProdukt1* p2=new TProdukt1("zeszyt", 2.0);       produkty[1] =p2; cout<<*p2;
 TProdukt2* p3=new TProdukt2("olowek",0.80,7);     produkty[2] =p3; cout<<*p3;
 TProdukt2* p4=new TProdukt2("pioro",4.80,20);     produkty[3] =p4; cout<<*p4;
 TProdukt2* p5=new TProdukt2("dlugopis",6.80,22); produkty[4] =p5; cout<<*p5;

 for (int i=0;i<N;i++)
   cout<<(*produkty[i]);
 for (int i=0; i<N;i++)
    delete produkty[i];

 cin.get();
}
```

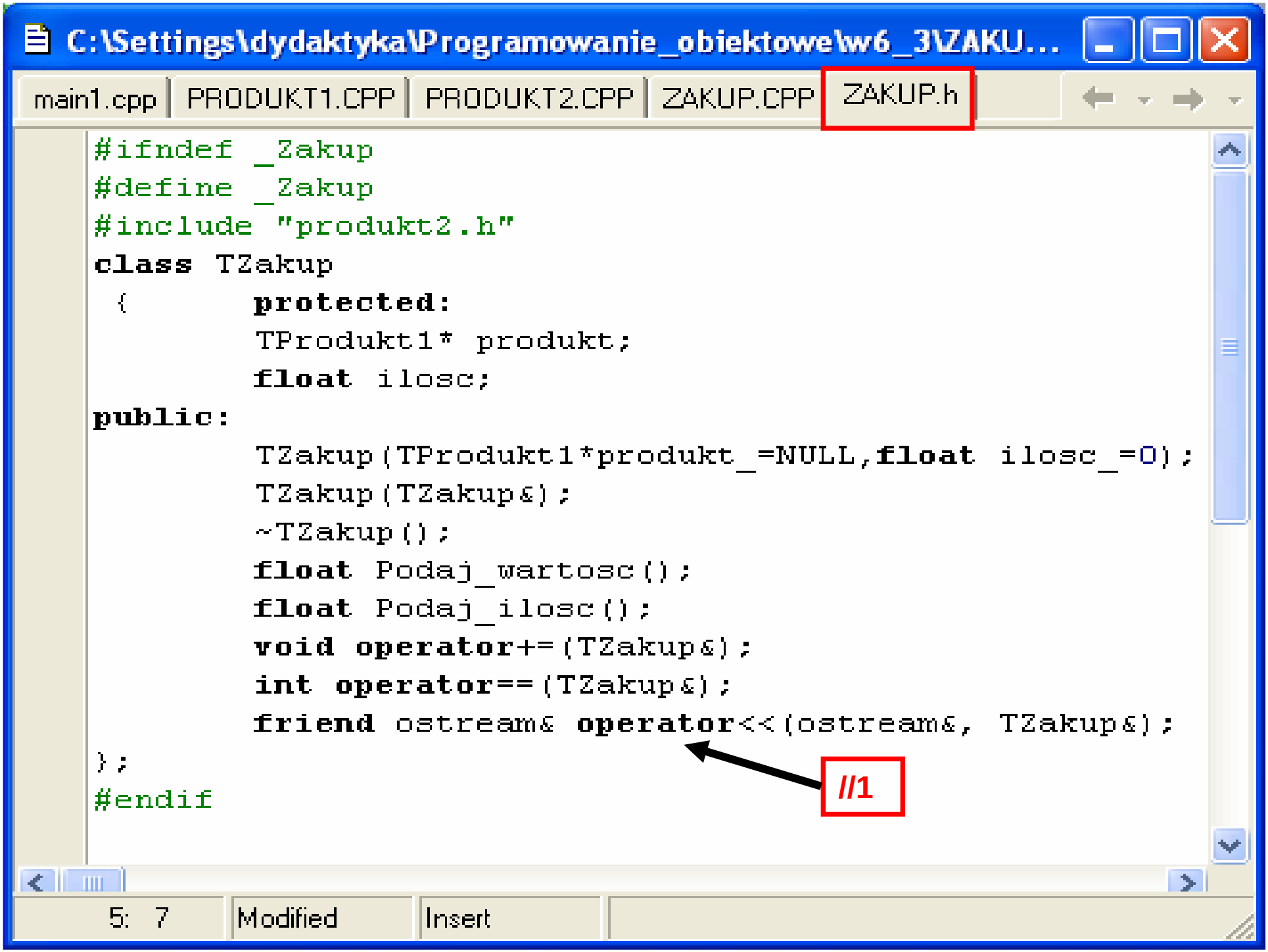
Wywoływany operator<< z klasy **TProdukt1**, natomiast w ciele tej funkcji operatorowej wywoływana jest wirtualna metoda ***toString***, stąd na ekranie pokazują się poprawne wyniki. Teraz podatek jest wyświetlany tylko dla obiektów typu **TProdukt2**

14: 61 | Modified | Insert

1 –
operator <<
z klasy
TProdukt1
2 –
operator<<
z klasy
TProdukt2

1 - operator<<
z klasyTProdukt1.
Poprawnie
drukowana cena i
podatek dzięki
wirtualnej metodzie
toString
wywoływanej w
ciele operatora<<.
Teraz podatek
jest wyświetlany
tyiko dla obiektów
typu TProdukt2




C:\Settings\dydaktyka\Programowanie_obiektowe\w6_3\ZAKU...
main1.cpp
PRODUKT1.CPP
PRODUKT2.CPP
ZAKUP.CPP
ZAKUP.h
#ifndef _Zakup
#define _Zakup
#include "produkt2.h"
class TZakup
{ protected:
TProdukt1* produkt;
float ilosc;
public:
TZakup(TProdukt1*produkt_=NULL,float ilosc_=0);
TZakup(TZakup&);
~TZakup();
float Podaj_wartosc();
float Podaj_ilosc();
void operator+=(TZakup&);
int operator==(TZakup&);
friend ostream& operator<<(ostream&, TZakup&);
};
#endif
//1
5: 7
Modified
Insert

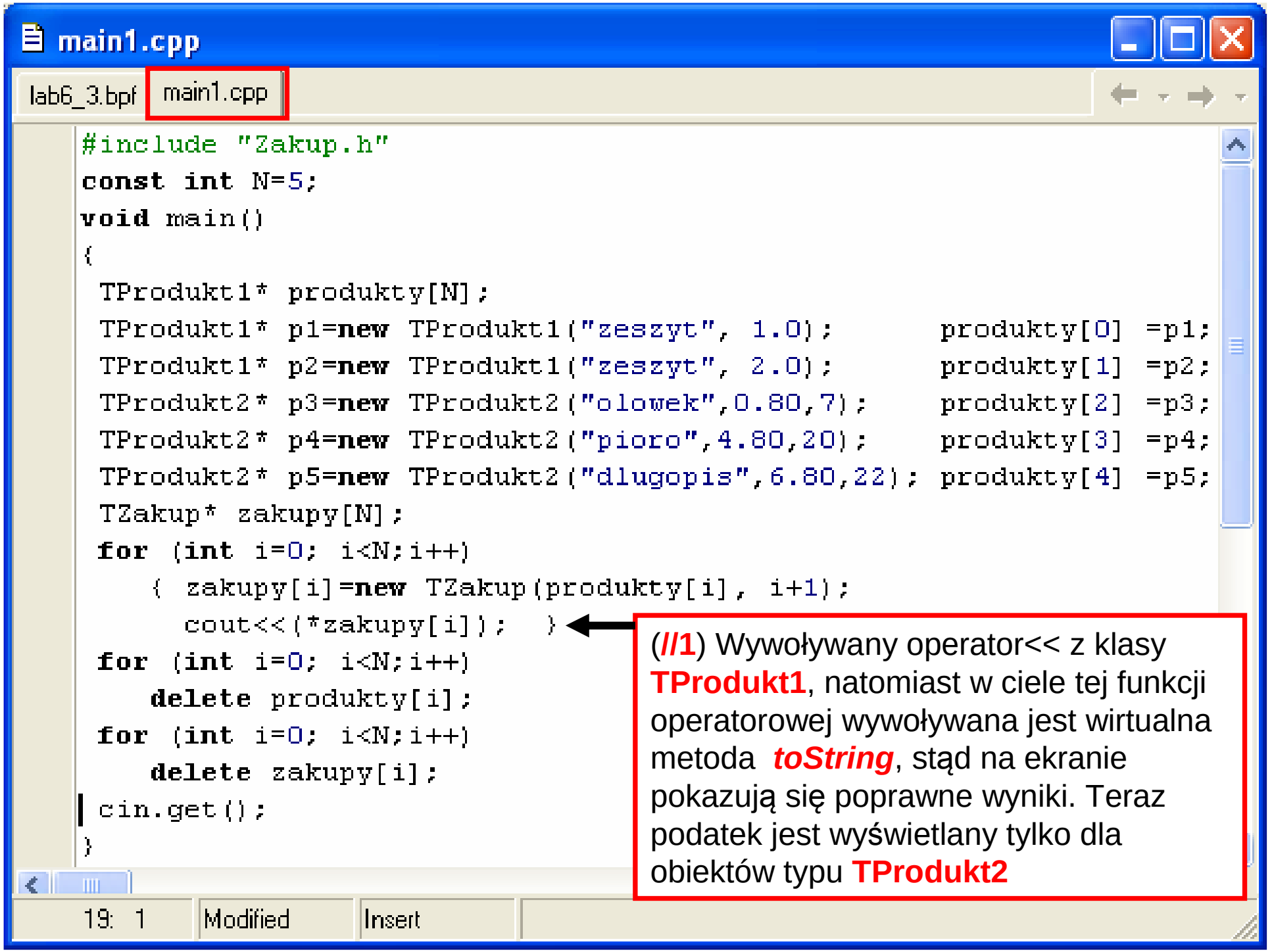
ZAKUP.CPP

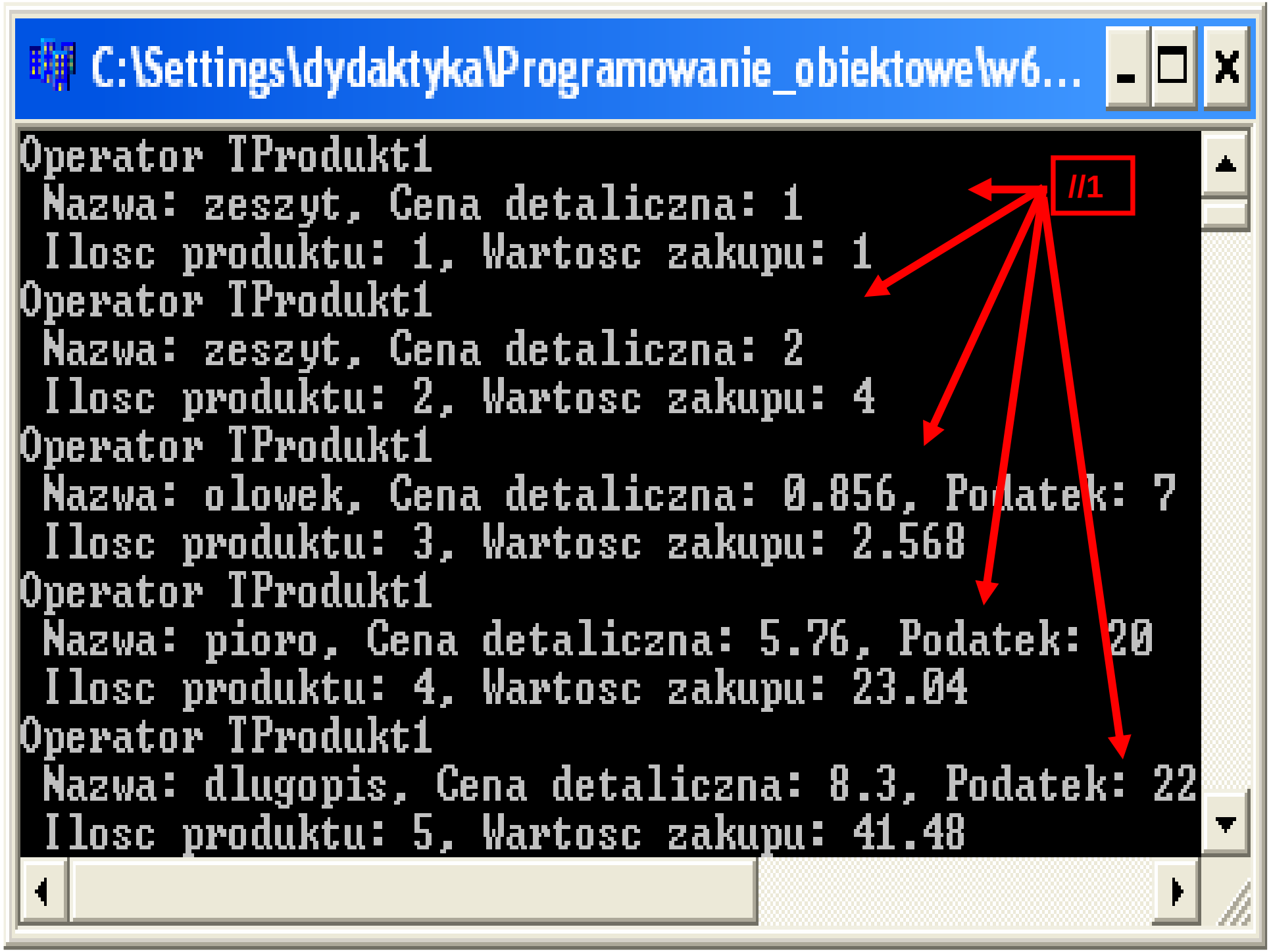
main1.cpp | PRODUKT1.CPP | PRODUKT2.CPP | ZAKUP.CPP

```
#include "Zakup.h"

TZakup::TZakup(TProdukt1*produkt_,float ilosc_)
{ /*cout<<"Konstruktor zwykly z parametrami klasy TZakup"<<endl;*/
  produkt=produkt_;
  ilosc=ilosc_;}
TZakup::TZakup(TZakup& z)
 { produkt=z.produkt;
   ilosc=z.ilosc;
   /*cout<<"Konstruktor kopiujacy klasy TZakup"<<endl;*/  }
TZakup::~TZakup()
 { /*cout<<"Destruktor klasy TZakup"<<endl; */ }
float TZakup::Podaj_wartosc()
 { return ilosc*produkt->Podaj_cene(); }
float TZakup::Podaj_ilosc()
 {  return ilosc;  }
void TZakup::operator+=(TZakup& z)
 { ilosc+=z.ilosc;  }
int TZakup::operator==(TZakup& zakup)
 { return  *produkt==*zakup.produkt; }
ostream& operator<<(ostream& wy, TZakup& zakup)
 { return wy<<*zakup.produkt<<
           " Ilosc produktu: "<<zakup.ilosc<<
           ", Wartosc zakupu: "<<zakup.Podaj_wartosc()<<endl; }
```

//1

1: 1 | Insert


main1.cpp
lab6_3.bpf
main1.cpp
#include "Zakup.h"
const int N=5;
void main()
{
TProdukt1* produkty[N];
TProdukt1* p1=new TProdukt1("zeszyt", 1.0); produkty[0] =p1;
TProdukt1* p2=new TProdukt1("zeszyt", 2.0); produkty[1] =p2;
TProdukt2* p3=new TProdukt2("olowek",0.80,7); produkty[2] =p3;
TProdukt2* p4=new TProdukt2("pioro",4.80,20); produkty[3] =p4;
TProdukt2* p5=new TProdukt2("dlugopis",6.80,22); produkty[4] =p5;
TZakup* zakupy[N];
for (int i=0; i<N;i++)
{ zakupy[i]=new TZakup(produkty[i], i+1);
cout<<(*zakupy[i]); }
for (int i=0; i<N;i++)
delete produkty[i];
for (int i=0; i<N;i++)
delete zakupy[i];
cin.get();
}
(//1) Wywoływany operator<< z klasy TProdukt1, natomiast w ciele tej funkcji operatorowej wywoływana jest wirtualna metoda toString, stąd na ekranie pokazują się poprawne wyniki. Teraz podatek jest wyświetlany tylko dla obiektów typu TProdukt2
19: 1
Modified
Insert


C:\Settings\dydaktyka\Programowanie_obiektowe\w6...
Operator TProdukt1
Nazwa: zeszyt, Cena detaliczna: 1
Ilosc produktu: 1, Wartosc zakupu: 1
Operator TProdukt1
Nazwa: zeszyt, Cena detaliczna: 2
Ilosc produktu: 2, Wartosc zakupu: 4
Operator TProdukt1
Nazwa: olowek, Cena detaliczna: 0.856, Podatek: 7
Ilosc produktu: 3, Wartosc zakupu: 2.568
Operator TProdukt1
Nazwa: pioro, Cena detaliczna: 5.76, Podatek: 20
Ilosc produktu: 4, Wartosc zakupu: 23.04
Operator TProdukt1
Nazwa: dlugopis, Cena detaliczna: 8.3, Podatek: 22
Ilosc produktu: 5, Wartosc zakupu: 41.48
//1

Dziedziczenie jednobazowe, poliformizm

1. **Polimorfizm (1) – tablice wskaźników na obiekty**
2. **Polimorfizm (2) – tablice wskaźników na obiekty**
3. **Polimorfizm (3) – tablice wskaźników na unikatowe obiekty, klasa abstrakcyjna**

Metody czysto wirtualne, klasy abstrakcyjne

Metody czysto wirtualne są deklarowane z inicjalizacją zerem:

virtual void wyswietl () = 0;

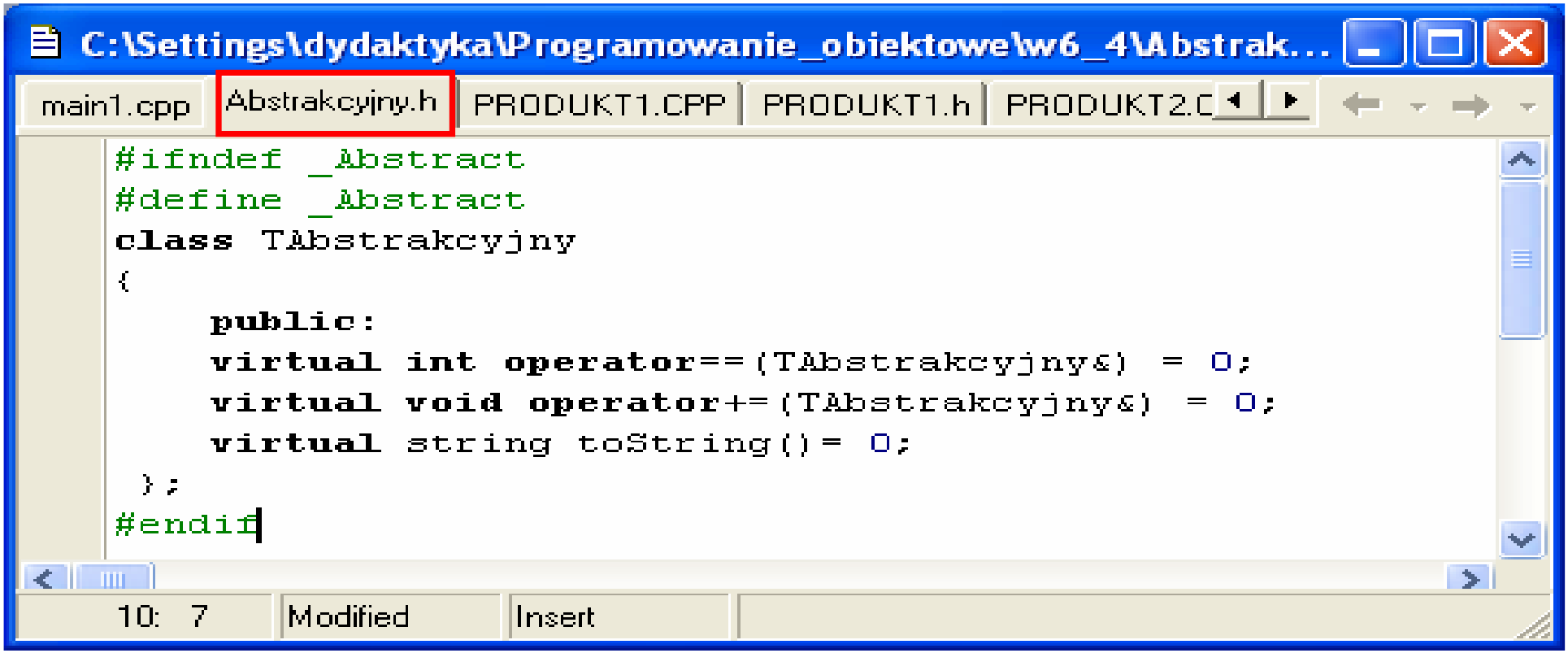
Klasa zawierająca co najmniej jedną taką funkcję jest **klasą abstrakcyjną**. Oznacza to, że nie można tworzyć obiektów jej typu.

Funkcja czysto wirtualna **musi być przedefiniowana w klasie pochodnej**, albo nowo zadeklarowana pochodna klasa będzie ją dziedziczyć, stąd stanie się również *klasą abstrakcyjną*.

Klasa abstrakcyjna służy do ***uogólnienia*** cech wszystkich jej następców w danej rodzinie ***za pomocą metod czysto wirtualnych***. Wystarczy korzystać ze wskaźników lub referencji do klasy abstrakcyjnej, aby można było zastosować metodę czysto wirtualną, która w czasie działania programu zostanie zastąpiona przedefiniowaną metodą wywołaną przez aktualnego następcę klasy abstrakcyjnej.

Stąd dzięki klasie abstrakcyjnej można ograniczyć:

- liczbę innych klas
- funkcji obsługujących całą rodzinę klas dziedziczących od klasy abstrakcyjnej.



Klasa *TAbstrakcyjny* pozwala uogólnić przechowanie wskaźników obiektów np. w tablicach

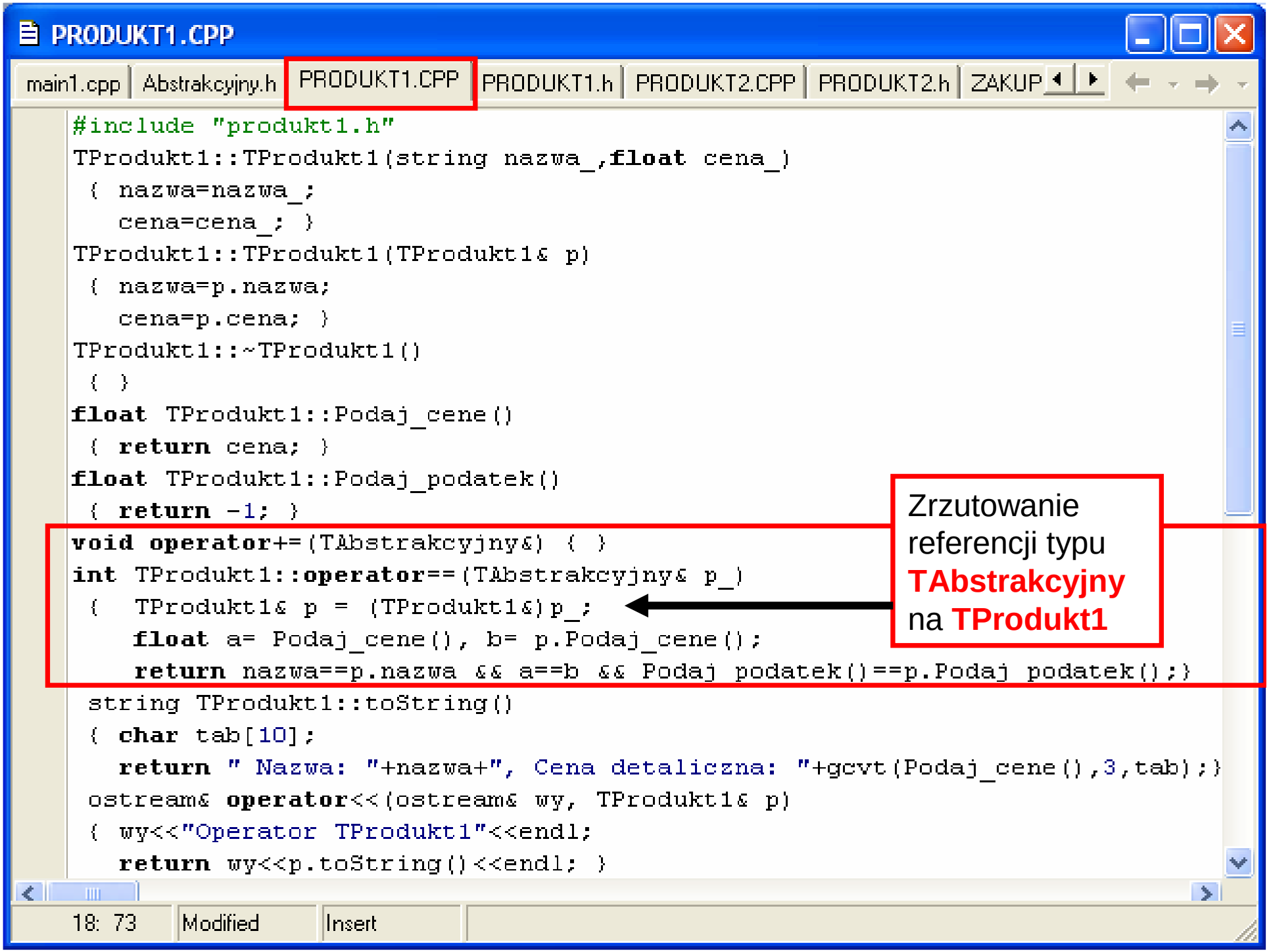
C:\Settings\dydaktyka\Programowanie_obiektowe\w6_4\PRODUKT1.h

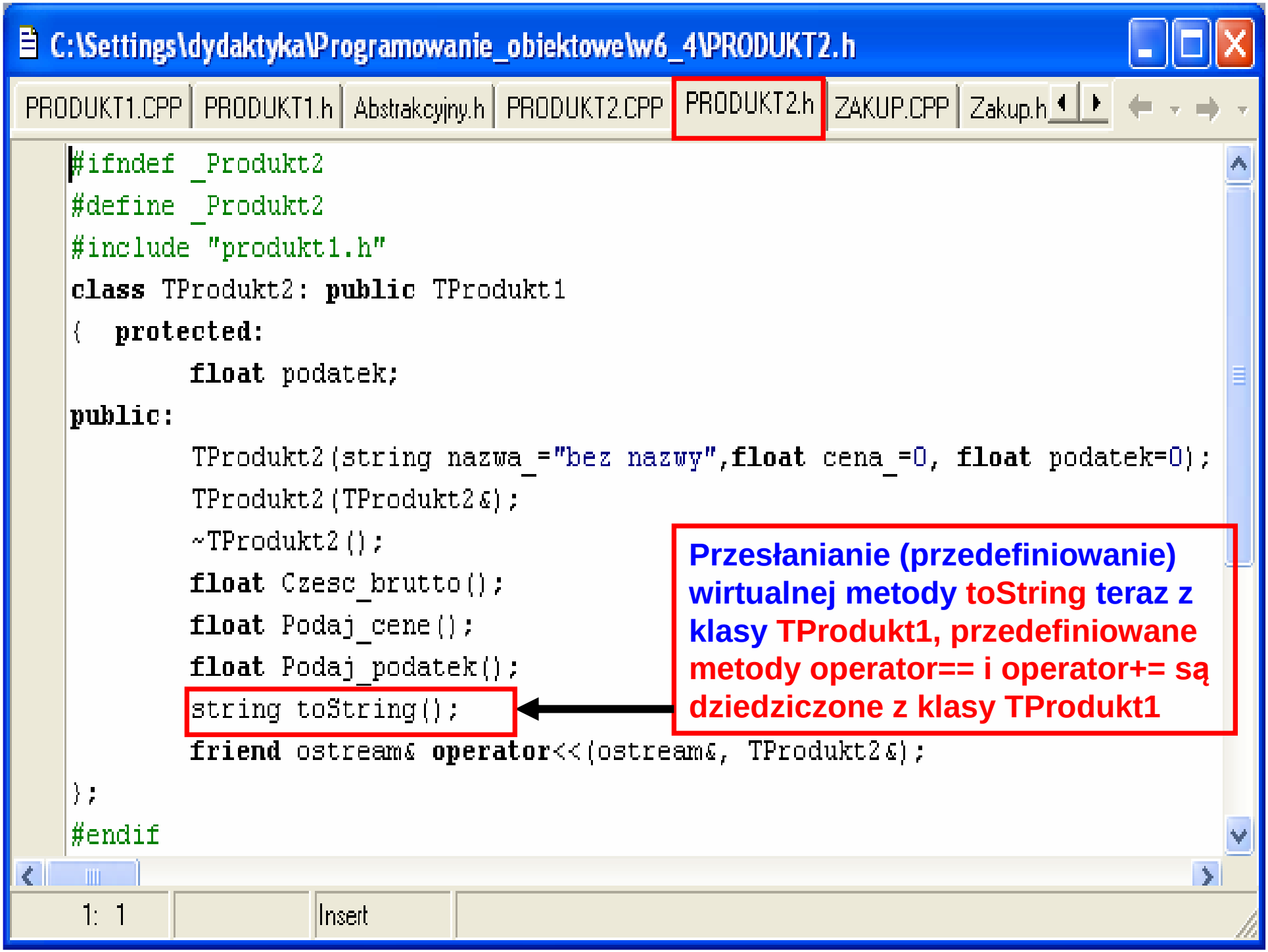
main1.cpp | Abstrakcyjny.h | PRODUKT1.CPP | PRODUKT1.h | PRODUKT2.CPP

```
#ifndef _Produkt1
#define _Produkt1
#include <iostream.h>
#include <string.h>
#include <iomanip.h>
#include <stdlib.h>
#include "Abstrakcyjny.h"
class TProdukt1: public TAbstrakcyjny
{protected:
          string nazwa;
          float cena;
public:
          TProdukt1(string nazwa_="bez nazwy",float cena_=0);
          TProdukt1(TProdukt1&);
          ~TProdukt1();
          virtual float Podaj_cene();
          virtual float Podaj_podatek();
          void operator+=(TAbstrakcyjny&);
          int operator==(TAbstrakcyjny&);
          string toString();
          friend ostream& operator<<(ostream&, TProdukt1&);
};
#endif
```

Przesłanianie (przedefiniowanie) wirtualnych metod abstrakcyjnych z klasy abstrakcyjnej

18: 13 | Modified | Insert


PRODUKT1.CPP
main1.cpp
Abstrakcyjny.h
PRODUKT1.CPP
PRODUKT1.h
PRODUKT2.CPP
PRODUKT2.h
ZAKUP
#include "produkt1.h"
TProdukt1::TProdukt1(string nazwa_,float cena_)
{ nazwa=nazwa_;
cena=cena_; }
TProdukt1::TProdukt1(TProdukt1& p)
{ nazwa=p.nazwa;
cena=p.cena; }
TProdukt1::~TProdukt1()
{ }
float TProdukt1::Podaj_cene()
{ return cena; }
float TProdukt1::Podaj_podatek()
{ return -1; }
void operator+=(TAbstrakcyjny&) { }
int TProdukt1::operator==(TAbstrakcyjny& p_)
{ TProdukt1& p = (TProdukt1&)p_;
float a= Podaj_cene(), b= p.Podaj_cene();
return nazwa==p.nazwa && a==b && Podaj_podatek()==p.Podaj_podatek();}
string TProdukt1::toString()
{ char tab[10];
return " Nazwa: "+nazwa+", Cena detaliczna: "+gcvt(Podaj_cene(),3,tab);}
ostream& operator<<(ostream& wy, TProdukt1& p)
{ wy<<"Operator TProdukt1"<<endl;
return wy<<p.toString()<<endl; }
Zrzutowanie referencji typu TAbstrakcyjny na TProdukt1
18: 73
Modified
Insert

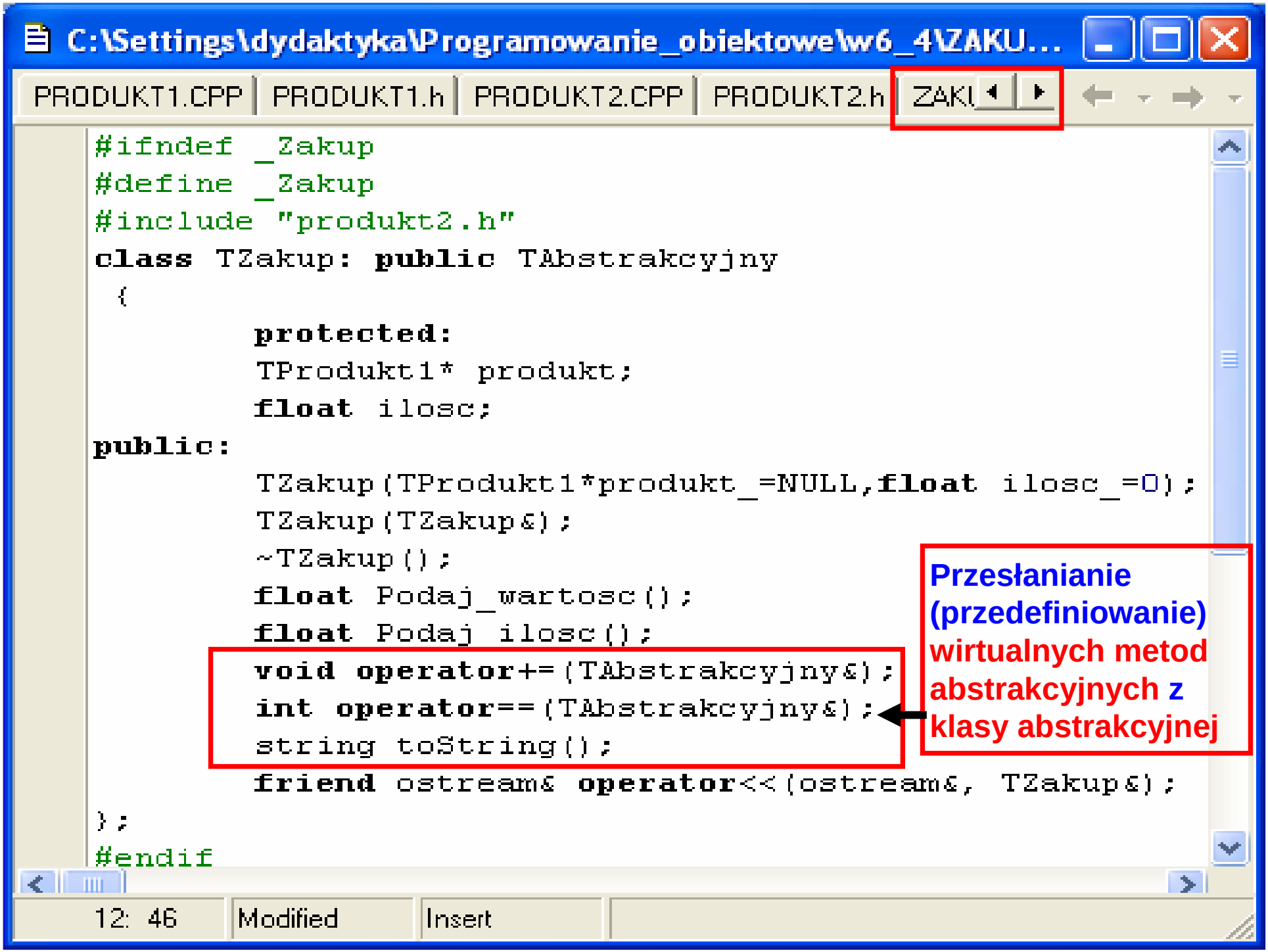

C:\Settings\dydaktyka\Programowanie_obiektowe\w6_4\PRODUKT2.h
PRODUKT1.CPP
PRODUKT1.h
Abstrakcyjny.h
PRODUKT2.CPP
PRODUKT2.h
ZAKUP.CPP
Zakup.h
#ifndef _Produkt2
#define _Produkt2
#include "produkt1.h"
class TProdukt2: public TProdukt1
{ protected:
float podatek;
public:
TProdukt2(string nazwa_="bez nazwy",float cena_=0, float podatek=0);
TProdukt2(TProdukt2&);
~TProdukt2();
float Czesc_brutto();
float Podaj_cene();
float Podaj_podatek();
string toString();
friend ostream& operator<<(ostream&, TProdukt2&);
};
#endif
Przesłanianie (przedefiniowanie) wirtualnej metody toString teraz z klasy TProdukt1, przedefiniowane metody operator== i operator+= są dziedziczone z klasy TProdukt1
1: 1
Insert

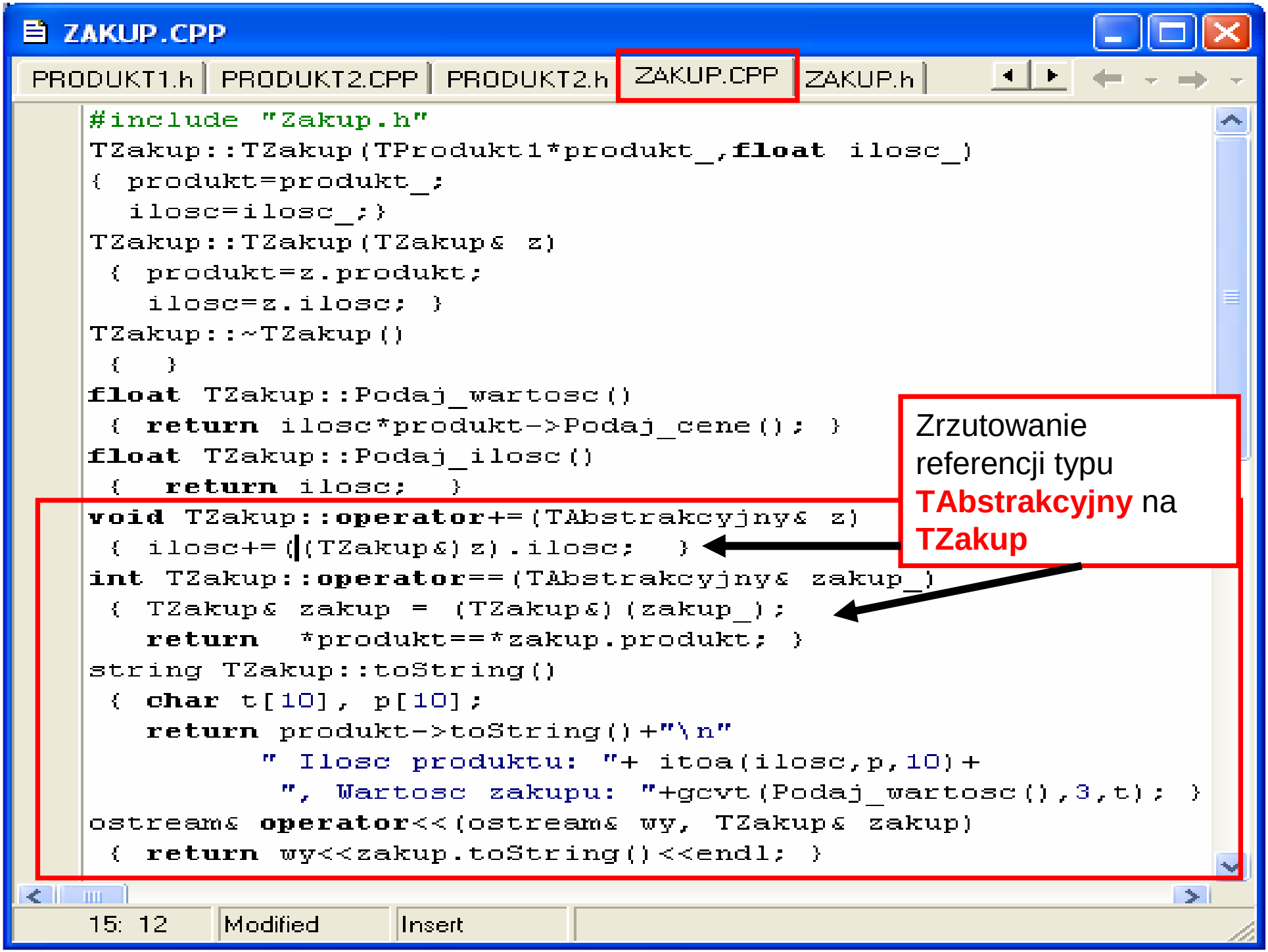
PRODUKT2.CPP

PRODUKT1.CPP | PRODUKT1.h | Abstrakcyjny.h | PRODUKT2.CPP | PRODUKT2.h | ZAKUP.CPP | Zaku

```
#include "produkt2.h"
TProdukt2::TProdukt2(string nazwa_, float cena_, float podatek_):
                          TProdukt1(nazwa_,cena_), podatek(podatek_)
 {  }
TProdukt2::TProdukt2(TProdukt2& p):TProdukt1(p), podatek(p.podatek)
 {  }
TProdukt2::~TProdukt2()
 {  }
float TProdukt2::Podaj_podatek()
 { return podatek; }
float TProdukt2::Czesc_brutto()
 { return cena*podatek/100; }
float TProdukt2::Podaj_cene()
 { return TProdukt1::Podaj_cene() + Czesc_brutto();}
string TProdukt2::toString()
 { char t[10];
   return TProdukt1::toString()+", Podatek: "+itoa(Podaj_podatek(),t,10); }
ostream& operator<<(ostream& wy, TProdukt2& p)
 { wy<<"Operator TProdukt2"<<endl;
   return wy<<p.toString()<<endl; }
```

1: 1 | Insert


C:\Settings\dydaktyka\Programowanie_obiektowe\w6_4\ZAKU...
PRODUKT1.CPP
PRODUKT1.h
PRODUKT2.CPP
PRODUKT2.h
ZAKU
#ifndef _Zakup
#define _Zakup
#include "produkt2.h"
class TZakup: public TAbstrakcyjny
{
protected:
TProdukt1* produkt;
float ilosc;
public:
TZakup(TProdukt1*produkt_=NULL,float ilosc_=0);
TZakup(TZakup&);
~TZakup();
float Podaj_wartosc();
float Podaj_ilosc();
void operator+=(TAbstrakcyjny&);
int operator==(TAbstrakcyjny&);
string toString();
friend ostream& operator<<(ostream&, TZakup&);
};
#endif
Przesłanianie (przedefiniowanie) wirtualnych metod abstrakcyjnych z klasy abstrakcyjnej
12: 46
Modified
Insert


ZAKUP.CPP
PRODUKT1.h
PRODUKT2.CPP
PRODUKT2.h
ZAKUP.CPP
ZAKUP.h
#include "Zakup.h"
TZakup::TZakup(TProdukt1*produkt_,float ilosc_)
{ produkt=produkt_;
ilosc=ilosc_;}
TZakup::TZakup(TZakup& z)
{ produkt=z.produkt;
ilosc=z.ilosc; }
TZakup::~TZakup()
{ }
float TZakup::Podaj_wartosc()
{ return ilosc*produkt->Podaj_cene(); }
float TZakup::Podaj_ilosc()
{ return ilosc; }
void TZakup::operator+=(TAbstrakcyjny& z)
{ ilosc+=((TZakup&)z).ilosc; }
int TZakup::operator==(TAbstrakcyjny& zakup_)
{ TZakup& zakup = (TZakup&)(zakup_);
return *produkt==*zakup.produkt; }
string TZakup::toString()
{ char t[10], p[10];
return produkt->toString()+"\n"
" Ilosc produktu: "+ itoa(ilosc,p,10)+
", Wartosc zakupu: "+gcvt(Podaj_wartosc(),3,t); }
ostream& operator<<(ostream& wy, TZakup& zakup)
{ return wy<<zakup.toString()<<endl; }
Zrzutowanie referencji typu TAbstrakcyjny na TZakup
15: 12
Modified
Insert

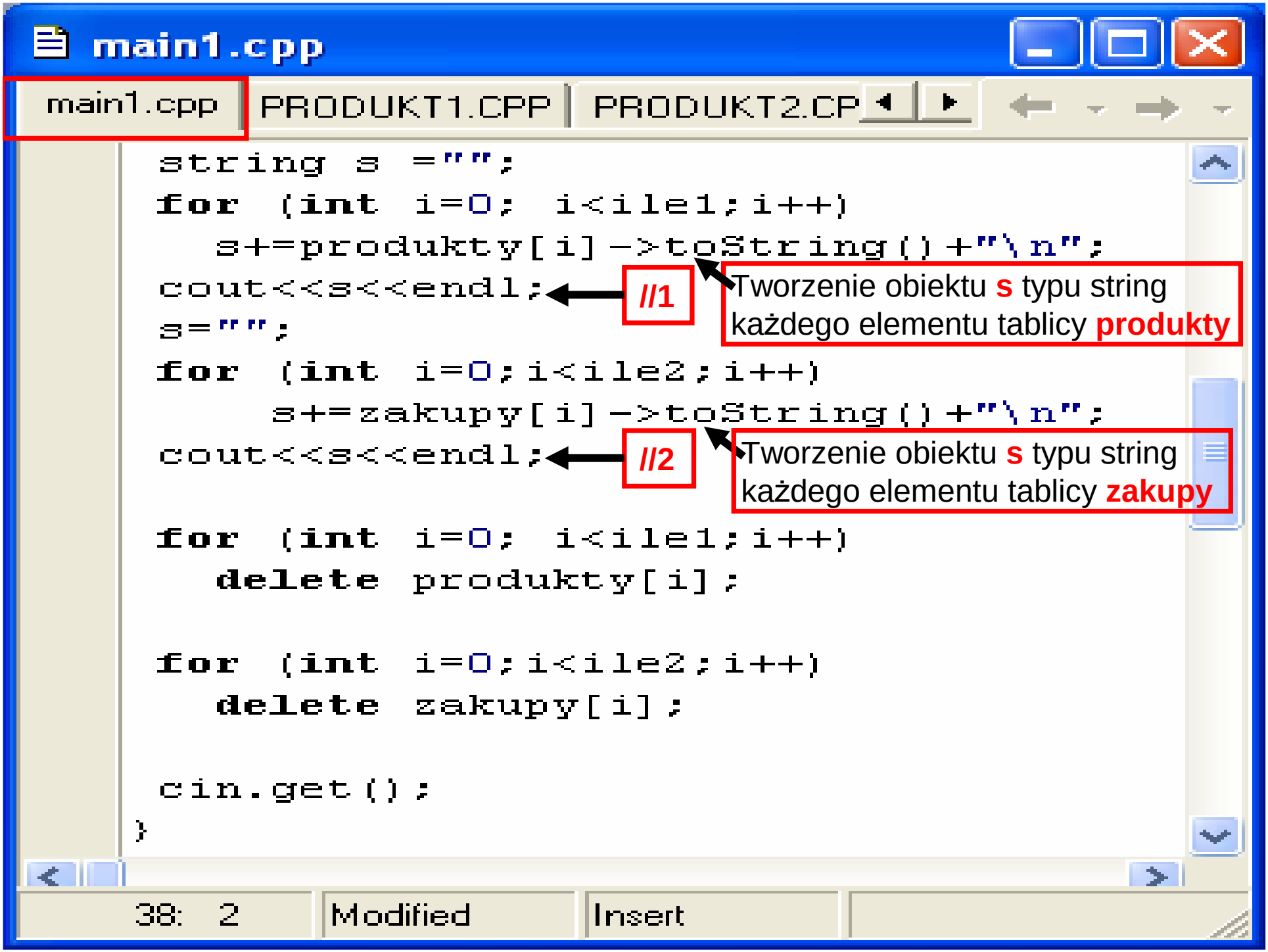
main1.cpp

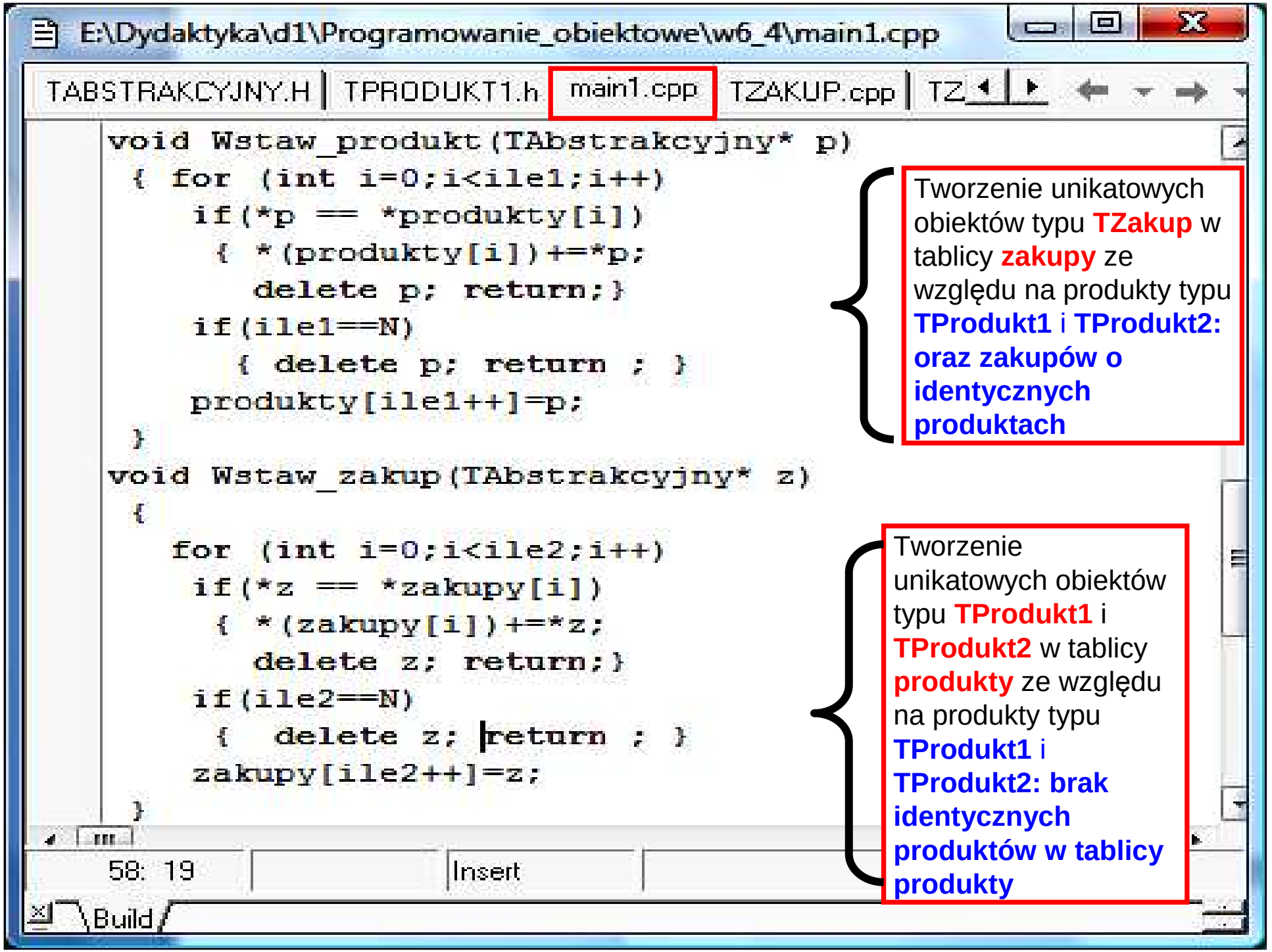
main1.cpp | Abstrakcyjny.h | PRODUKT1.CPP | PRODUKT1.h | PRODUKT2.CPP | PRODUKT2

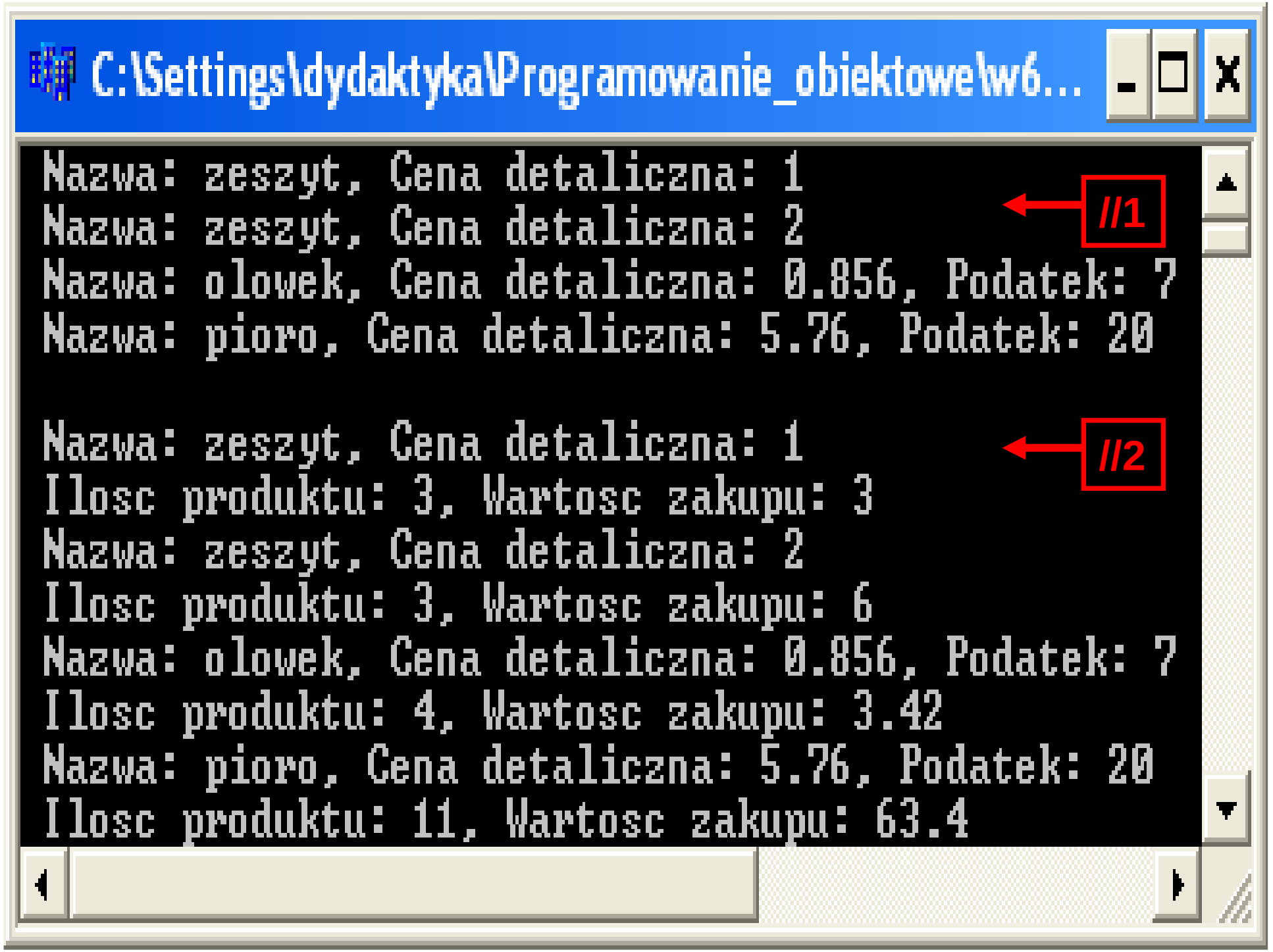
```
#include "Zakup.h"
  const int N=5;
  TAbstrakcyjny* produkty[N];
  TAbstrakcyjny* zakupy[N];
  int ile1,ile2;   //ile1 i ile2 maja wartosci 0
  void Wstaw_produkt(TAbstrakcyjny* p);
  void Wstaw_zakup(TAbstrakcyjny* z);
void main()
{
 TProdukt1* p1=new TProdukt1("zeszyt", 1.0);      Wstaw_produkt(p1);
 TProdukt1* p2=new TProdukt1("zeszyt", 1.0);      Wstaw_produkt(p2);
 TProdukt1* p3=new TProdukt1("zeszyt", 2.0);      Wstaw_produkt(p3);
 TProdukt2* p4=new TProdukt2("olowek",0.80,7);    Wstaw_produkt(p4);
 TProdukt2* p5=new TProdukt2("pioro",4.80,20);    Wstaw_produkt(p5);
 TProdukt2* p6=new TProdukt2("pioro",4.80,20);    Wstaw_produkt(p6);

 TZakup* z1= new TZakup(p1,1); Wstaw_zakup(z1);
 TZakup* z2= new TZakup(p1,2); Wstaw_zakup(z2);
 TZakup* z3= new TZakup(p3,3); Wstaw_zakup(z3);
 TZakup* z4= new TZakup(p4,4); Wstaw_zakup(z4);
 TZakup* z5= new TZakup(p5,5); Wstaw_zakup(z5);
 TZakup* z6= new TZakup(p5,6); Wstaw_zakup(z6);
```

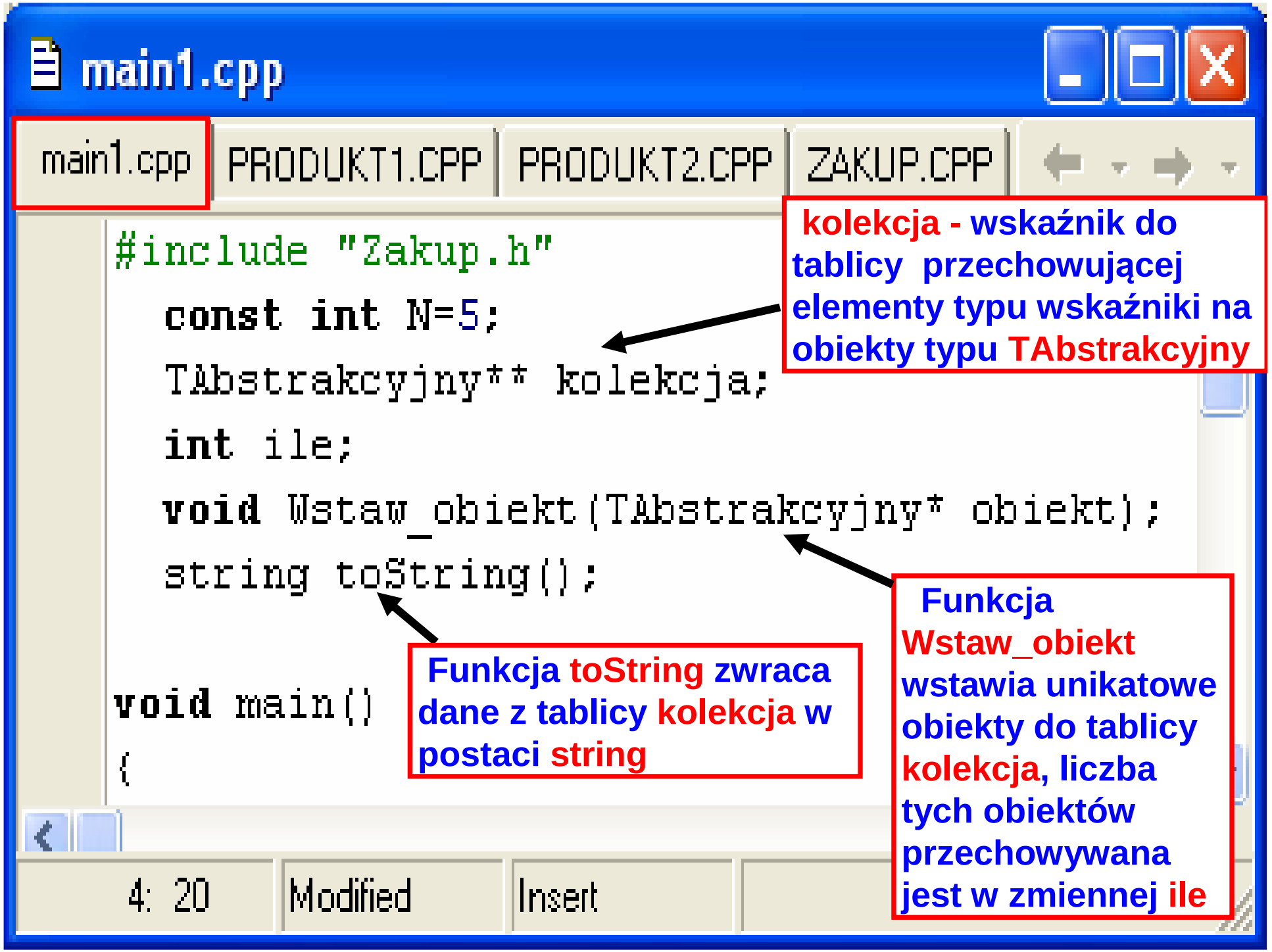
Tworzenie obiektów typu **TZakup** tylko z produktów, które nie powtarzają się, czyli znajdują się w tablicy produkty (produkty **p2, p6** są usunięte z pamięci)- utworzono 4 obiekty typu TZakup

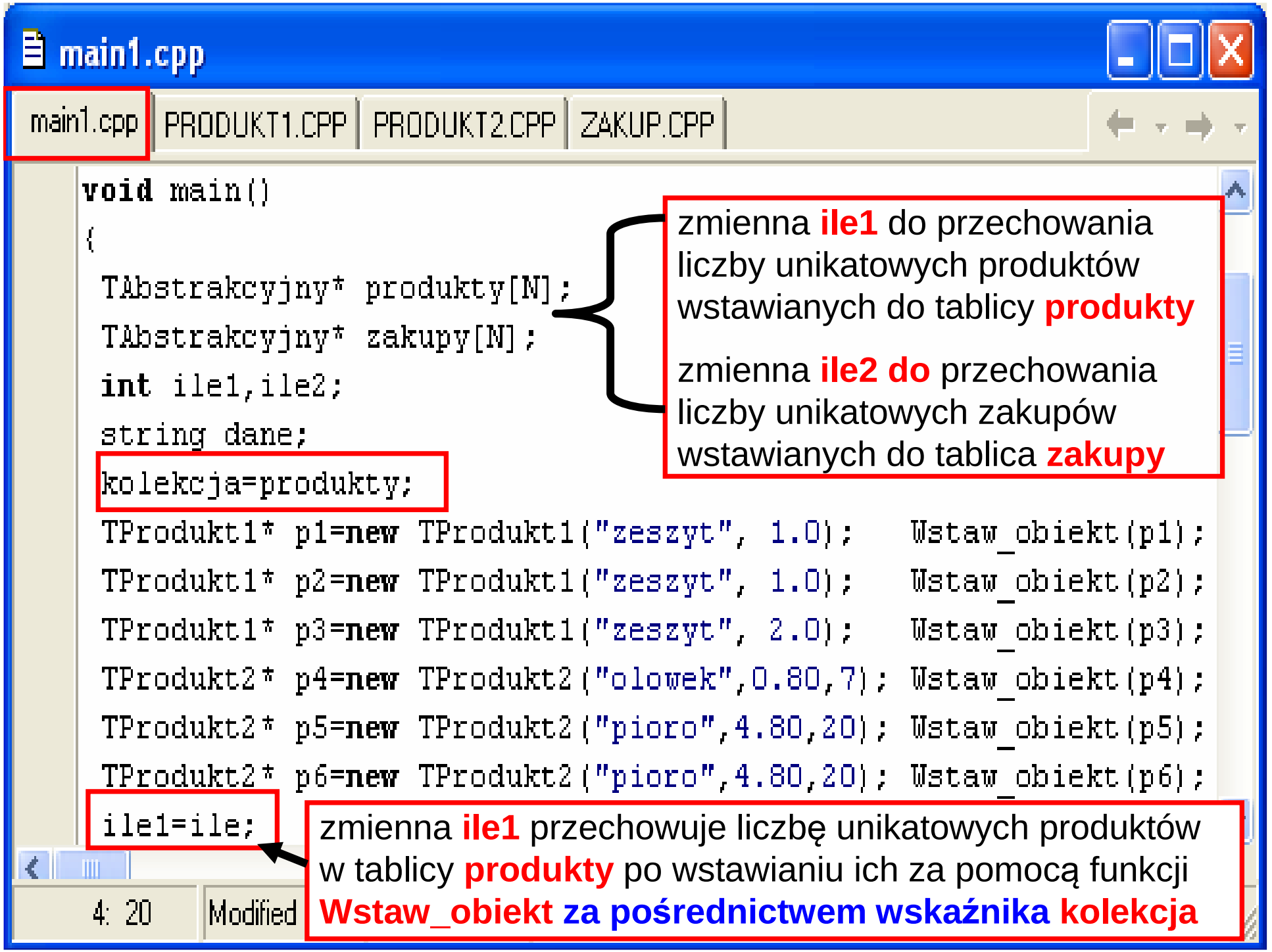
6: 47 | Modified | Insert

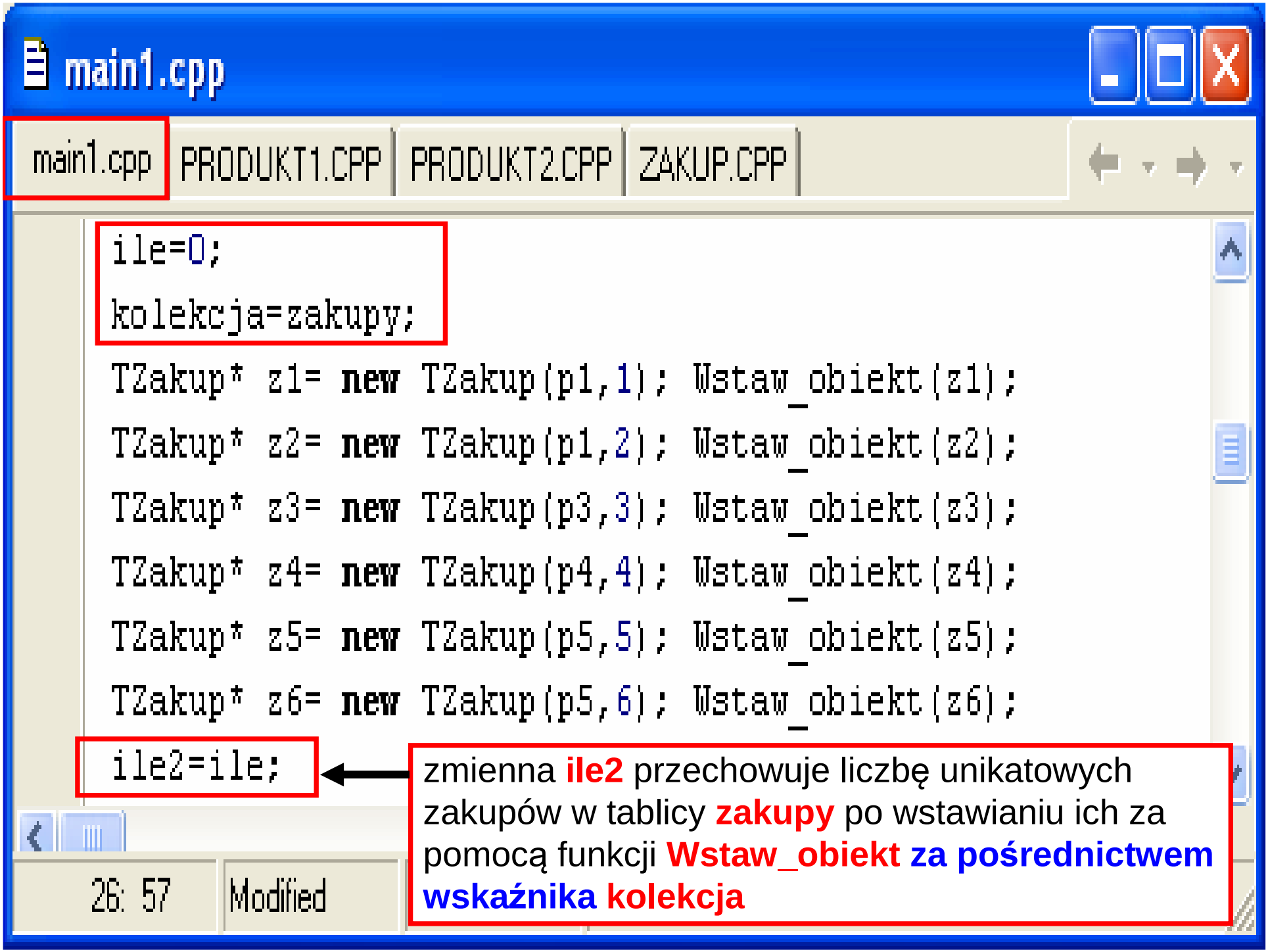

main1.cpp
main1.cpp
PRODUKT1.CPP
PRODUKT2.CP
string s ="";
for (int i=0; i<ile1;i++)
s+=produkty[i]->toString()+"\n";
cout<<s<<endl;
//1
s="";
for (int i=0;i<ile2;i++)
s+=zakupy[i]->toString()+"\n";
cout<<s<<endl;
//2
for (int i=0; i<ile1;i++)
delete produkty[i];
for (int i=0;i<ile2;i++)
delete zakupy[i];
cin.get();
}
Tworzenie obiektu s typu string
każdego elementu tablicy produkty
Tworzenie obiektu s typu string
każdego elementu tablicy zakupy
38: 2
Modified
Insert

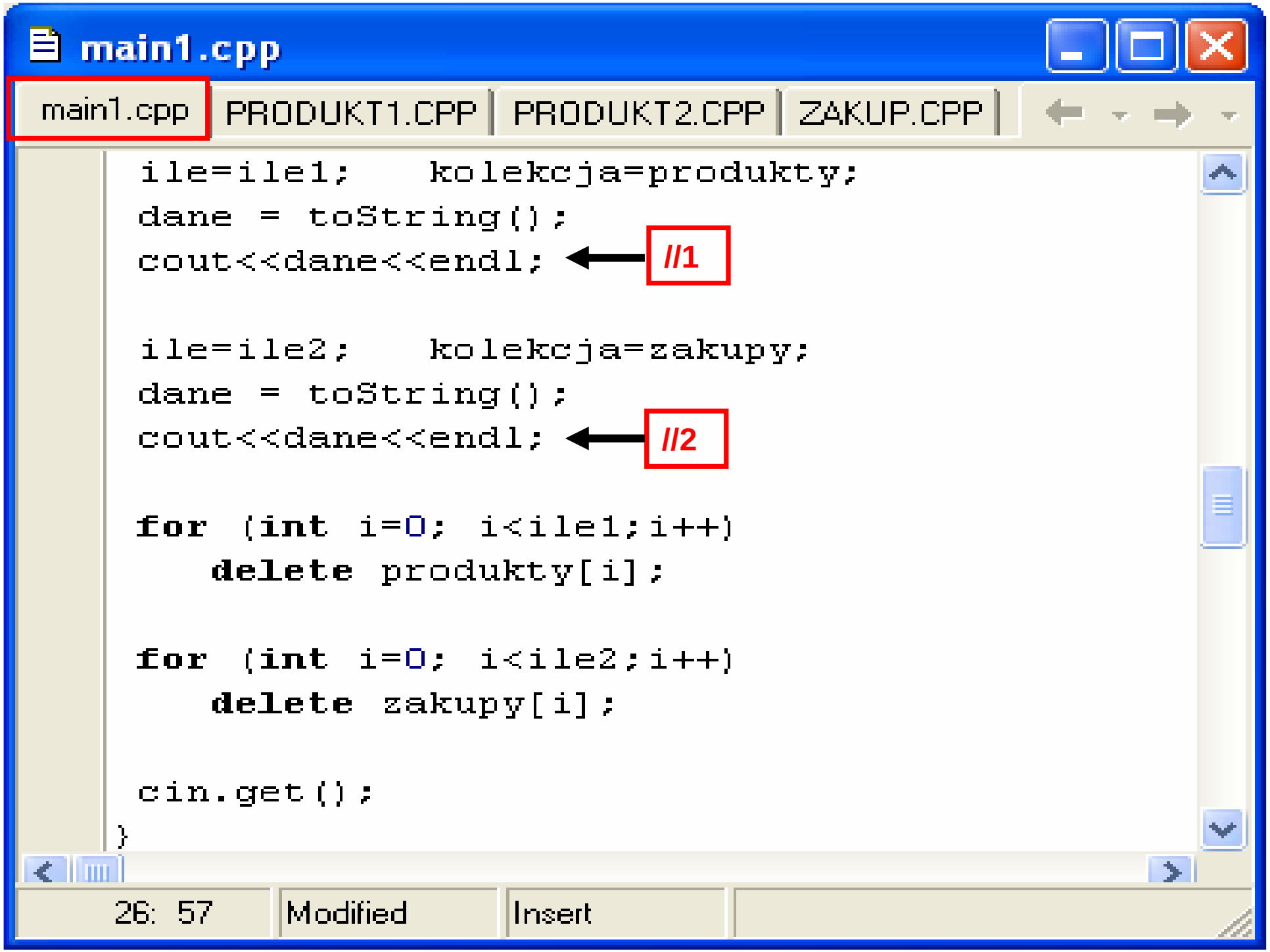

E:\Dydaktyka\d1\Programowanie_obiektowe\w6_4\main1.cpp
TABSTRAKCYJNY.H
TPRODUKT1.h
main1.cpp
TZAKUP.cpp
TZ
void Wstaw_produkt(TAbstrakcyjny* p)
{ for (int i=0;i<ile1;i++)
if(*p == *produkty[i])
{ *(produkty[i])+=*p;
delete p; return;}
if(ile1==N)
{ delete p; return ; }
produkty[ile1++]=p;
}
void Wstaw_zakup(TAbstrakcyjny* z)
{
for (int i=0;i<ile2;i++)
if(*z == *zakupy[i])
{ *(zakupy[i])+=*z;
delete z; return;}
if(ile2==N)
{ delete z; return ; }
zakupy[ile2++]=z;
}
58: 19
Insert
Build
Tworzenie unikatowych obiektów typu TZakup w tablicy zakupy ze względu na produkty typu TProdukt1 i TProdukt2: oraz zakupów o identycznych produktach
Tworzenie unikatowych obiektów typu TProdukt1 i TProdukt2 w tablicy produkty ze względu na produkty typu TProdukt1 i TProdukt2: brak identycznych produktów w tablicy produkty

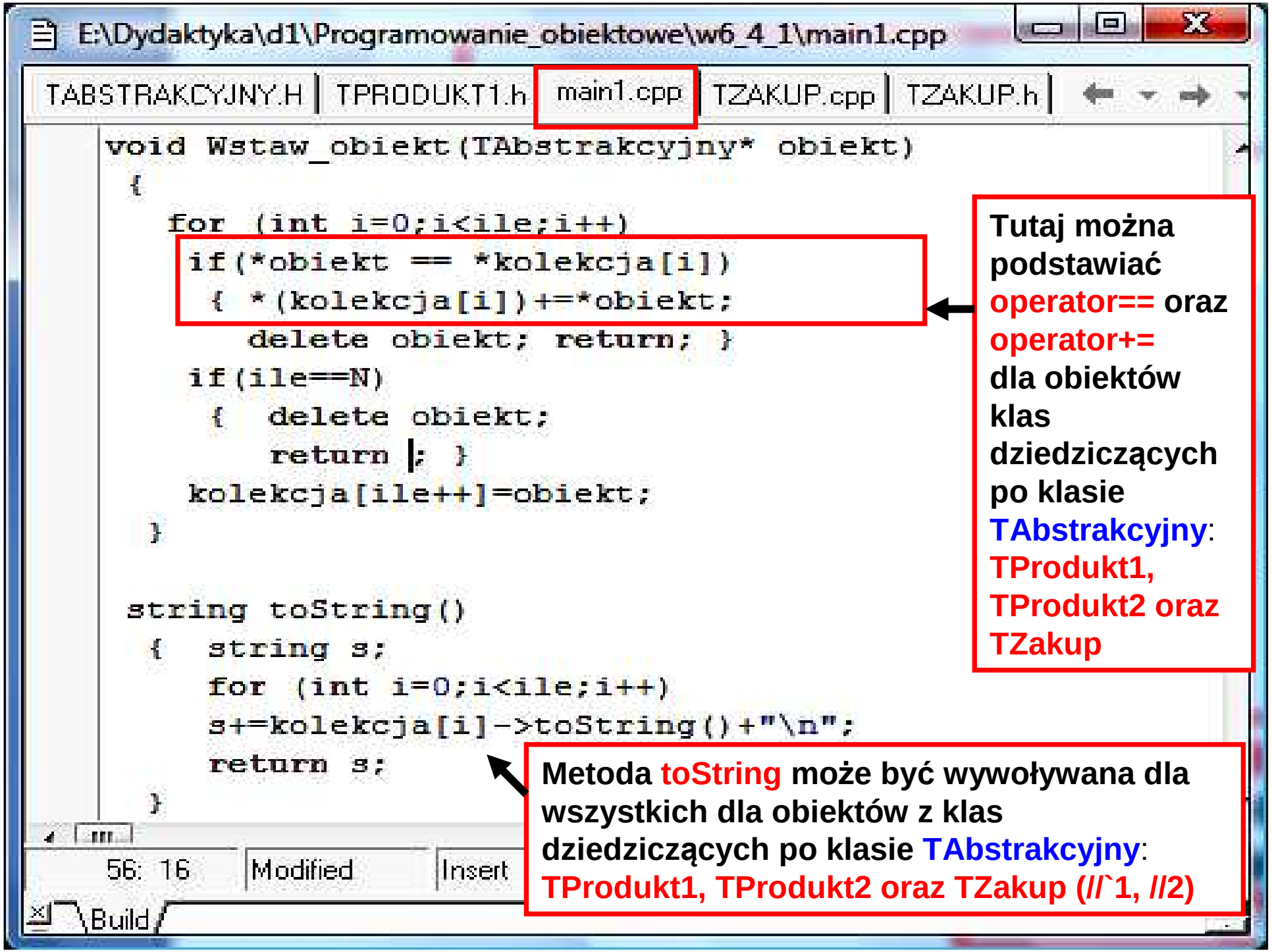

C:\Settings\dydaktyka\Programowanie_obiektowe\w6...
Nazwa: zeszyt, Cena detaliczna: 1
Nazwa: zeszyt, Cena detaliczna: 2
Nazwa: olowek, Cena detaliczna: 0.856, Podatek: 7
Nazwa: pioro, Cena detaliczna: 5.76, Podatek: 20
Nazwa: zeszyt, Cena detaliczna: 1
Ilosc produktu: 3, Wartosc zakupu: 3
Nazwa: zeszyt, Cena detaliczna: 2
Ilosc produktu: 3, Wartosc zakupu: 6
Nazwa: olowek, Cena detaliczna: 0.856, Podatek: 7
Ilosc produktu: 4, Wartosc zakupu: 3.42
Nazwa: pioro, Cena detaliczna: 5.76, Podatek: 20
Ilosc produktu: 11, Wartosc zakupu: 63.4
//1
//2

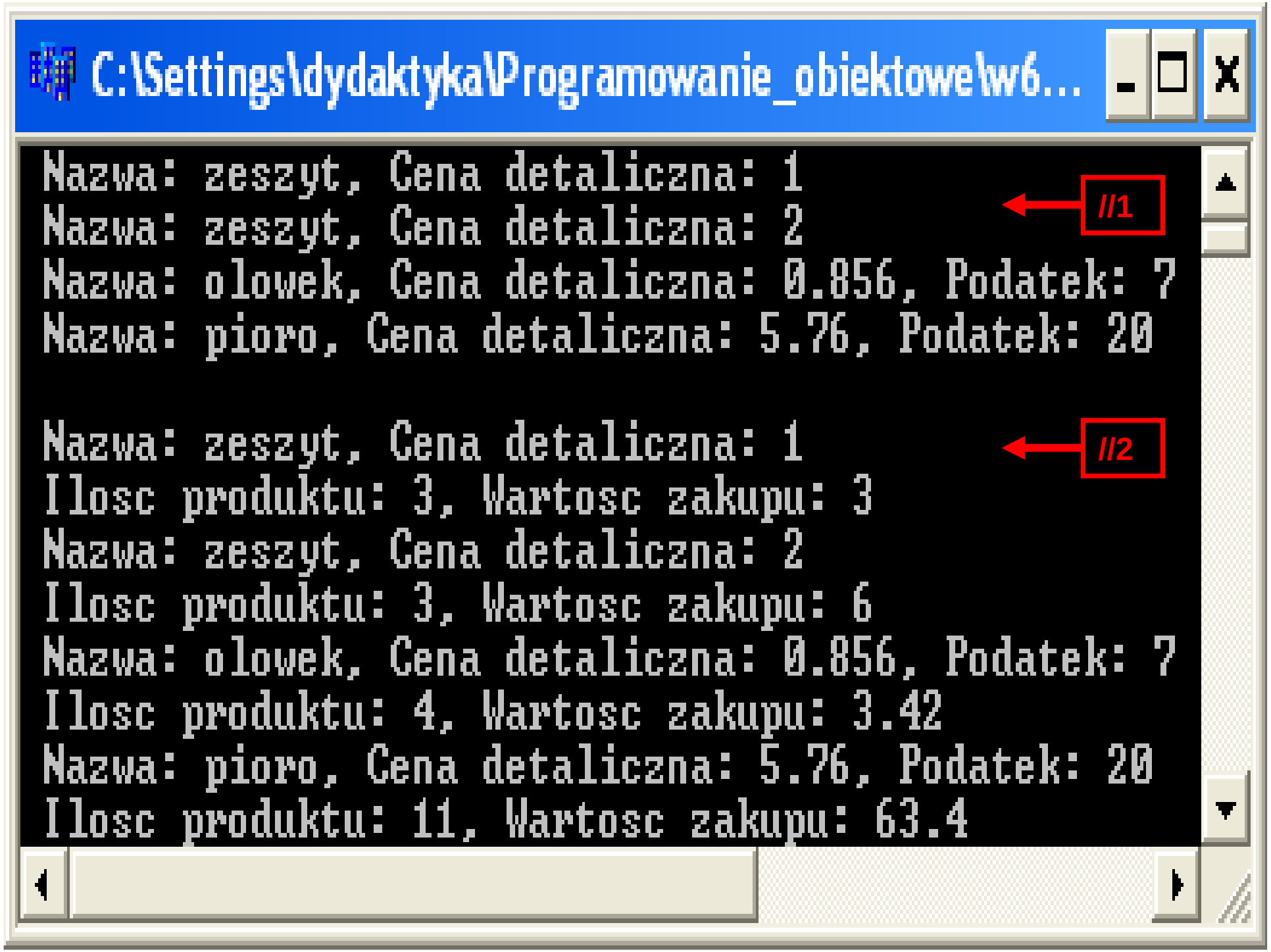

main1.cpp
main1.cpp
PRODUKT1.CPP
PRODUKT2.CPP
ZAKUP.CPP
#include "Zakup.h"
const int N=5;
TAbstrakcyjny** kolekcja;
int ile;
void Wstaw_obiekt(TAbstrakcyjny* obiekt);
string toString();
void main()
{
kolekcja - wskaźnik do tablicy przechowującej elementy typu wskaźniki na obiekty typu TAbstrakcyjny
Funkcja toString zwraca dane z tablicy kolekcja w postaci string
Funkcja Wstaw_obiekt wstawia unikatowe obiekty do tablicy kolekcja, liczba tych obiektów przechowywana jest w zmiennej ile
4: 20
Modified
Insert


main1.cpp
main1.cpp
PRODUKT1.CPP
PRODUKT2.CPP
ZAKUP.CPP
void main()
{
TAbstrakcyjny* produkty[N];
TAbstrakcyjny* zakupy[N];
int ile1,ile2;
string dane;
kolekcja=produkty;
TProdukt1* p1=new TProdukt1("zeszyt", 1.0); Wstaw_obiekt(p1);
TProdukt1* p2=new TProdukt1("zeszyt", 1.0); Wstaw_obiekt(p2);
TProdukt1* p3=new TProdukt1("zeszyt", 2.0); Wstaw_obiekt(p3);
TProdukt2* p4=new TProdukt2("olowek",0.80,7); Wstaw_obiekt(p4);
TProdukt2* p5=new TProdukt2("pioro",4.80,20); Wstaw_obiekt(p5);
TProdukt2* p6=new TProdukt2("pioro",4.80,20); Wstaw_obiekt(p6);
ile1=ile;
zmienna ile1 do przechowania liczby unikatowych produktów wstawianych do tablicy produkty
zmienna ile2 do przechowania liczby unikatowych zakupów wstawianych do tablica zakupy
zmienna ile1 przechowuje liczbę unikatowych produktów w tablicy produkty po wstawianiu ich za pomocą funkcji Wstaw_obiekt za pośrednictwem wskaźnika kolekcja
4: 20
Modified


main1.cpp
main1.cpp
PRODUKT1.CPP
PRODUKT2.CPP
ZAKUP.CPP
ile=0;
kolekcja=zakupy;
TZakup* z1= new TZakup(p1,1); Wstaw_obiekt(z1);
TZakup* z2= new TZakup(p1,2); Wstaw_obiekt(z2);
TZakup* z3= new TZakup(p3,3); Wstaw_obiekt(z3);
TZakup* z4= new TZakup(p4,4); Wstaw_obiekt(z4);
TZakup* z5= new TZakup(p5,5); Wstaw_obiekt(z5);
TZakup* z6= new TZakup(p5,6); Wstaw_obiekt(z6);
ile2=ile;
zmienna ile2 przechowuje liczbę unikatowych zakupów w tablicy zakupy po wstawianiu ich za pomocą funkcji Wstaw_obiekt za pośrednictwem wskaźnika kolekcja
26: 57
Modified


main1.cpp
main1.cpp
PRODUKT1.CPP
PRODUKT2.CPP
ZAKUP.CPP
ile=ile1; kolekcja=produkty;
dane = toString();
cout<<dane<<endl;
//1
ile=ile2; kolekcja=zakupy;
dane = toString();
cout<<dane<<endl;
//2
for (int i=0; i<ile1;i++)
delete produkty[i];
for (int i=0; i<ile2;i++)
delete zakupy[i];
cin.get();
}
26: 57
Modified
Insert


E:\Dydaktyka\d1\Programowanie_obiektowe\w6_4_1\main1.cpp
TABSTRAKCYJNY.H
TPRODUKT1.h
main1.cpp
TZAKUP.cpp
TZAKUP.h
void Wstaw_obiekt(TAbstrakcyjny* obiekt)
{
for (int i=0;i<ile;i++)
if(*obiekt == *kolekcja[i])
{ *(kolekcja[i])+=*obiekt;
delete obiekt; return; }
if(ile==N)
{ delete obiekt;
return; }
kolekcja[ile++]=obiekt;
}
string toString()
{ string s;
for (int i=0;i<ile;i++)
s+=kolekcja[i]->toString()+"\n";
return s;
}
56: 16
Modified
Insert
Build
Tutaj można podstawiać operator== oraz operator+= dla obiektów klas dziedziczących po klasie TAbstrakcyjny: TProdukt1, TProdukt2 oraz TZakup
Metoda toString może być wywoływana dla wszystkich dla obiektów z klas dziedziczących po klasie TAbstrakcyjny: TProdukt1, TProdukt2 oraz TZakup (//`1, //2)


C:\Settings\dydaktyka\Programowanie_obiektowe\w6...
Nazwa: zeszyt, Cena detaliczna: 1
Nazwa: zeszyt, Cena detaliczna: 2
Nazwa: olowek, Cena detaliczna: 0.856, Podatek: 7
Nazwa: pioro, Cena detaliczna: 5.76, Podatek: 20
Nazwa: zeszyt, Cena detaliczna: 1
Ilosc produktu: 3, Wartosc zakupu: 3
Nazwa: zeszyt, Cena detaliczna: 2
Ilosc produktu: 3, Wartosc zakupu: 6
Nazwa: olowek, Cena detaliczna: 0.856, Podatek: 7
Ilosc produktu: 4, Wartosc zakupu: 3.42
Nazwa: pioro, Cena detaliczna: 5.76, Podatek: 20
Ilosc produktu: 11, Wartosc zakupu: 63.4
//1
//2

Dziedziczenie jednobazowe, poliformizm

1. Polimorfizm (1) – tablice wskaźników na obiekty
2. Polimorfizm (2) – tablice wskaźników na obiekty
3. Polimorfizm (3) – tablice wskaźników na unikatowe obiekty, klasa abstrakcyjna
4. Klasa TKol1 obsługująca uniwersalną tablicę zawierającą wskaźniki na unikatowe obiekty

Klasa abstrakcyjna TAbstrakcyjny

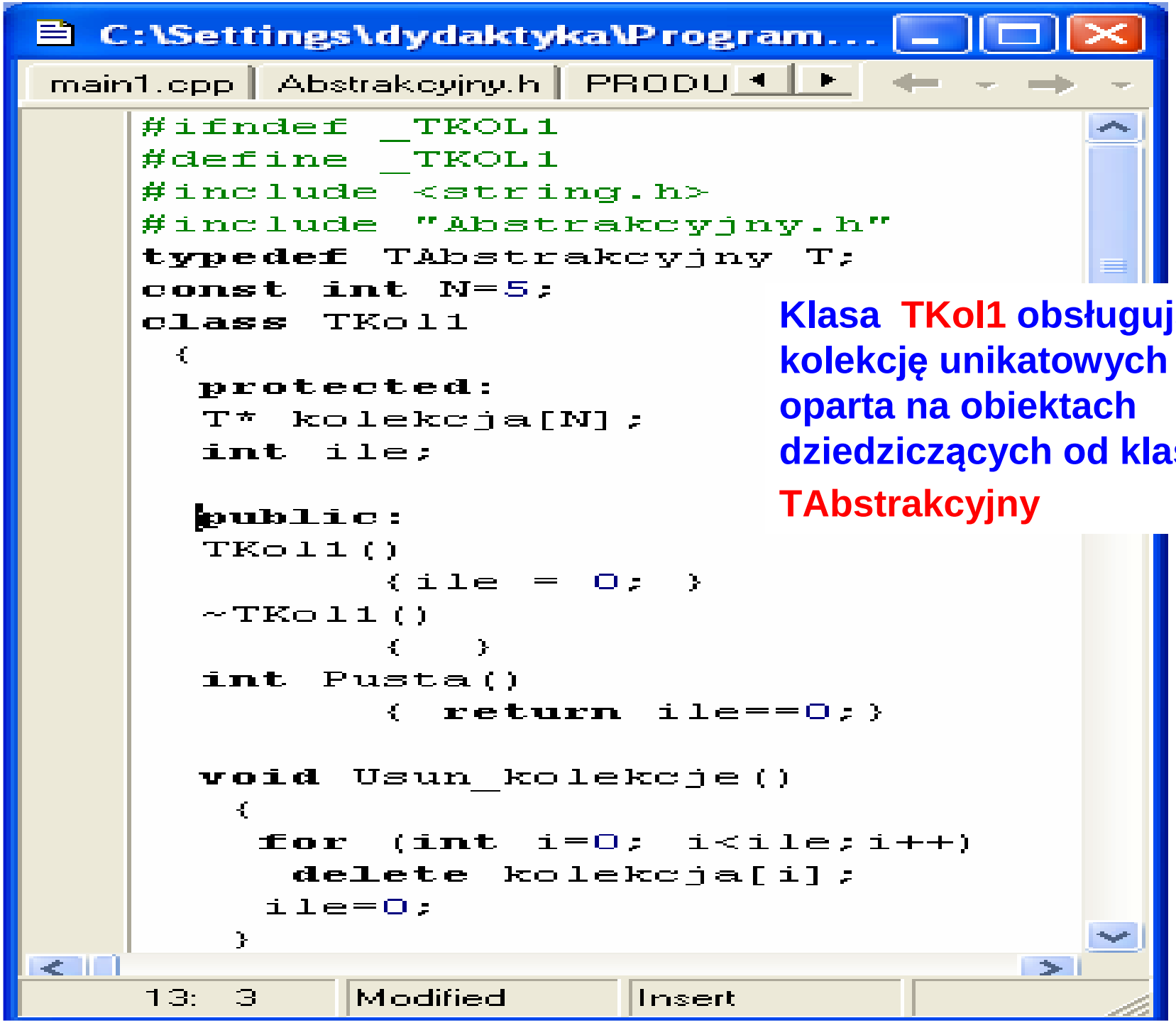
C:\Settings\dydaktyka\Programowanie_obiektowe\w6...

lab6_5.bpf | Abstrakcyjny.h

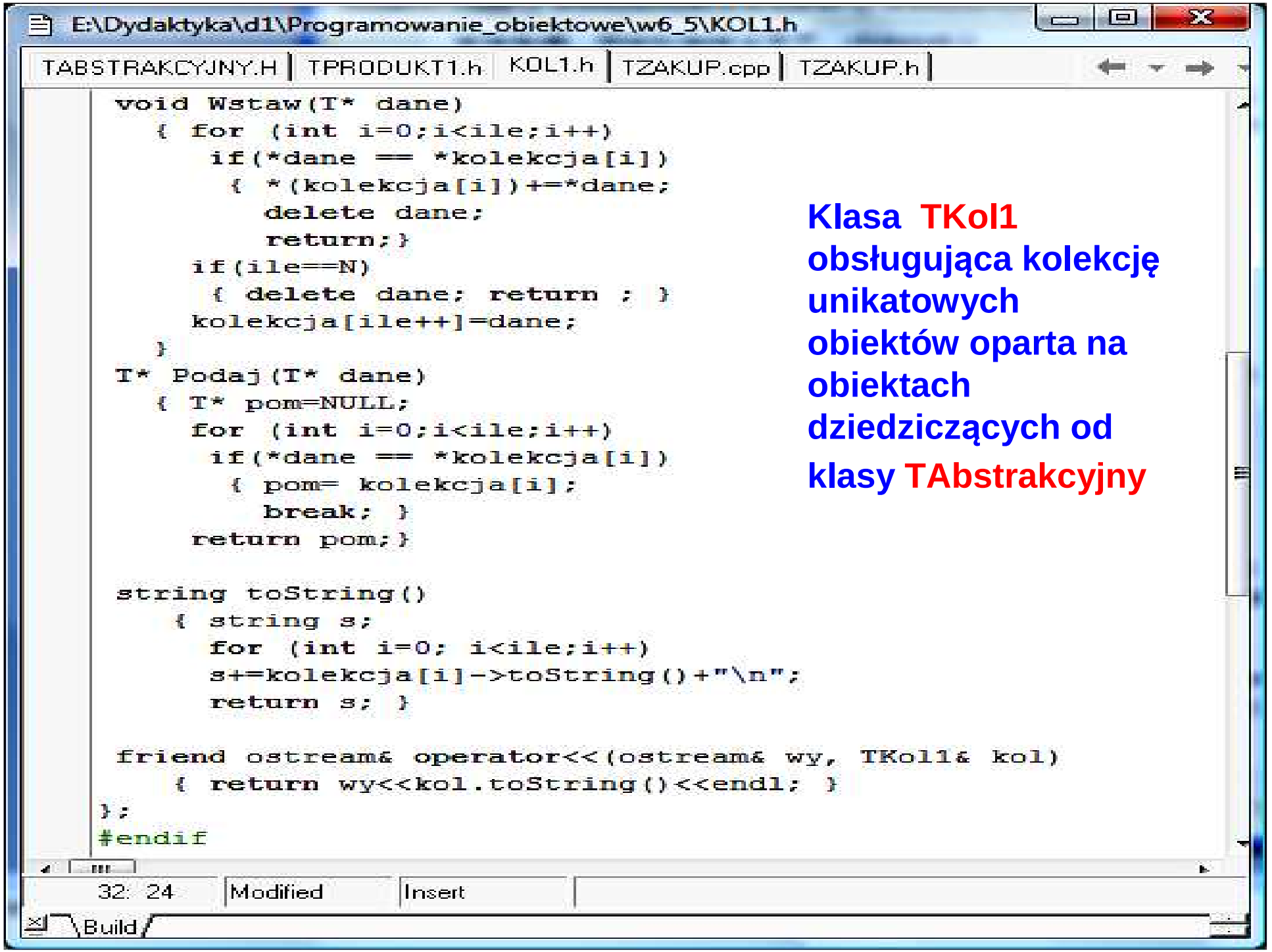
```
#ifndef _Abstract
#define _Abstract
class TAbstrakcyjny
{
    public:
    virtual int operator==(TAbstrakcyjny&) = 0;
    virtual void operator+=(TAbstrakcyjny&) = 0;
    virtual string toString()= 0;
    virtual ~TAbstrakcyjny() {}
 };
#endif
```

2: 49

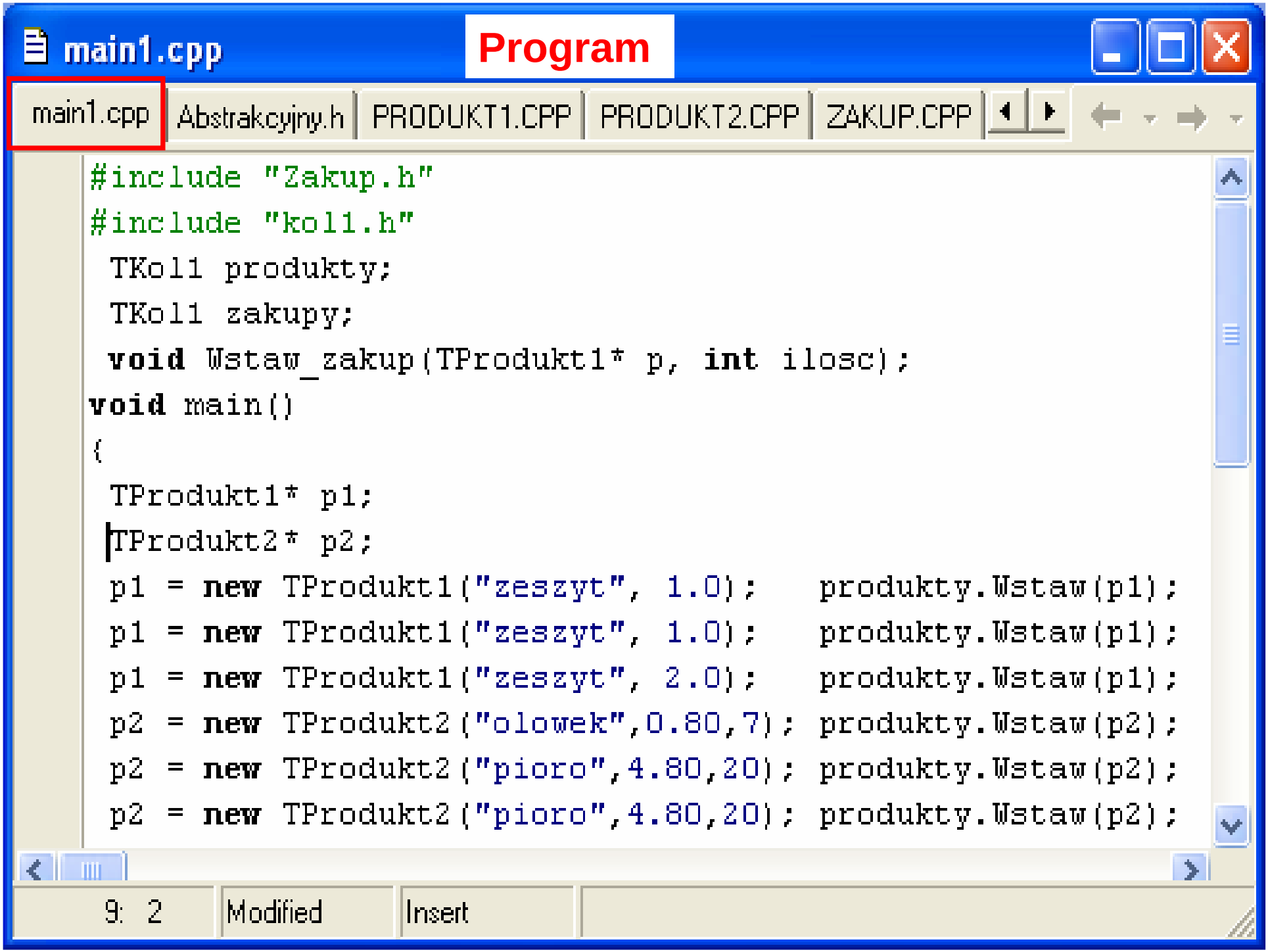
Wirtualny destruktor pozwala przy usuwaniu obiektów dziedziczących po klasie **TAbstrakcyjny wywołać charakterystyczny dla nich algorytm usuwania z pamięci danych powiązanych z usuwanym obiektem, zdefiniowany w destruktorze jego klasy.**

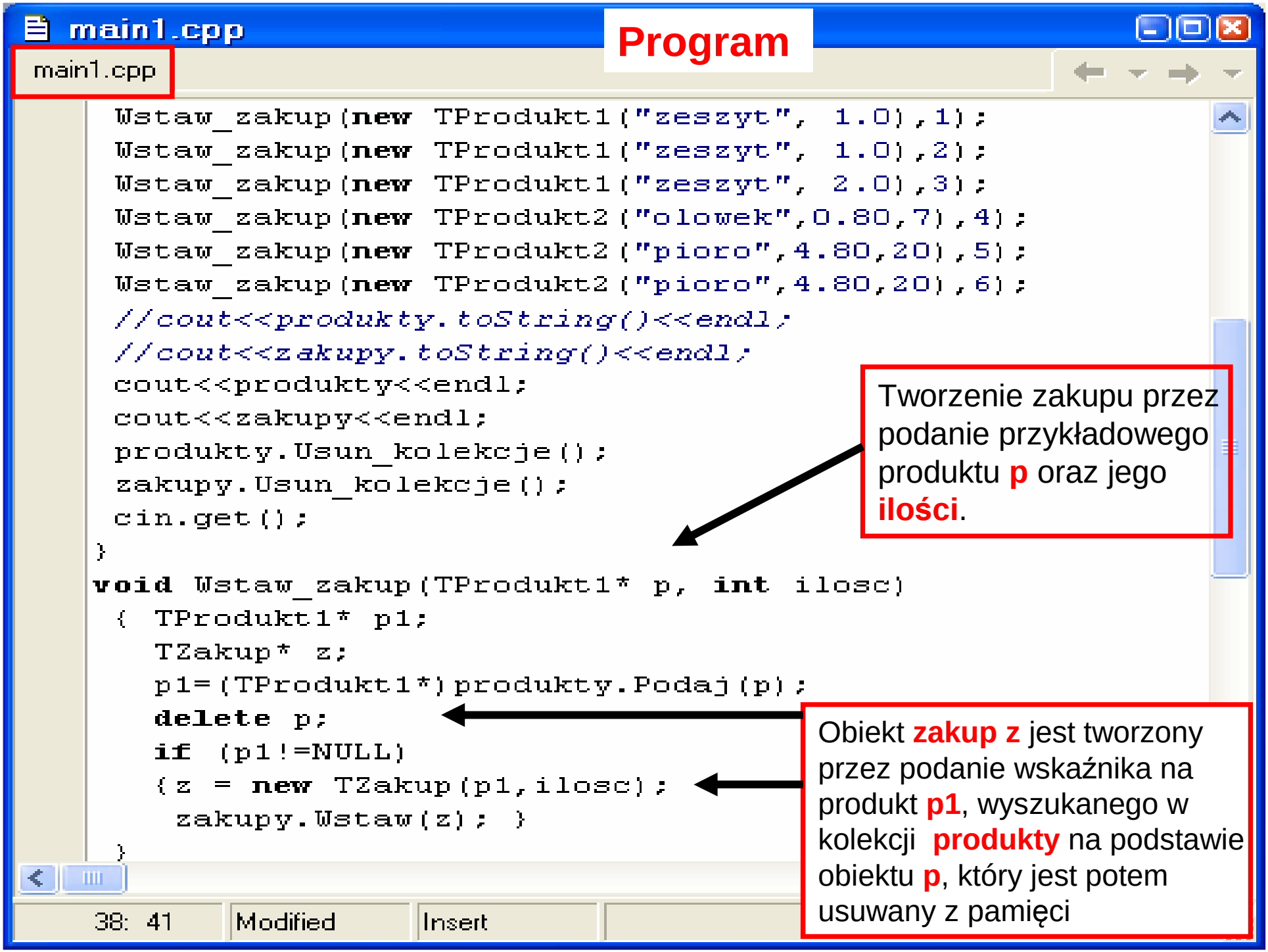


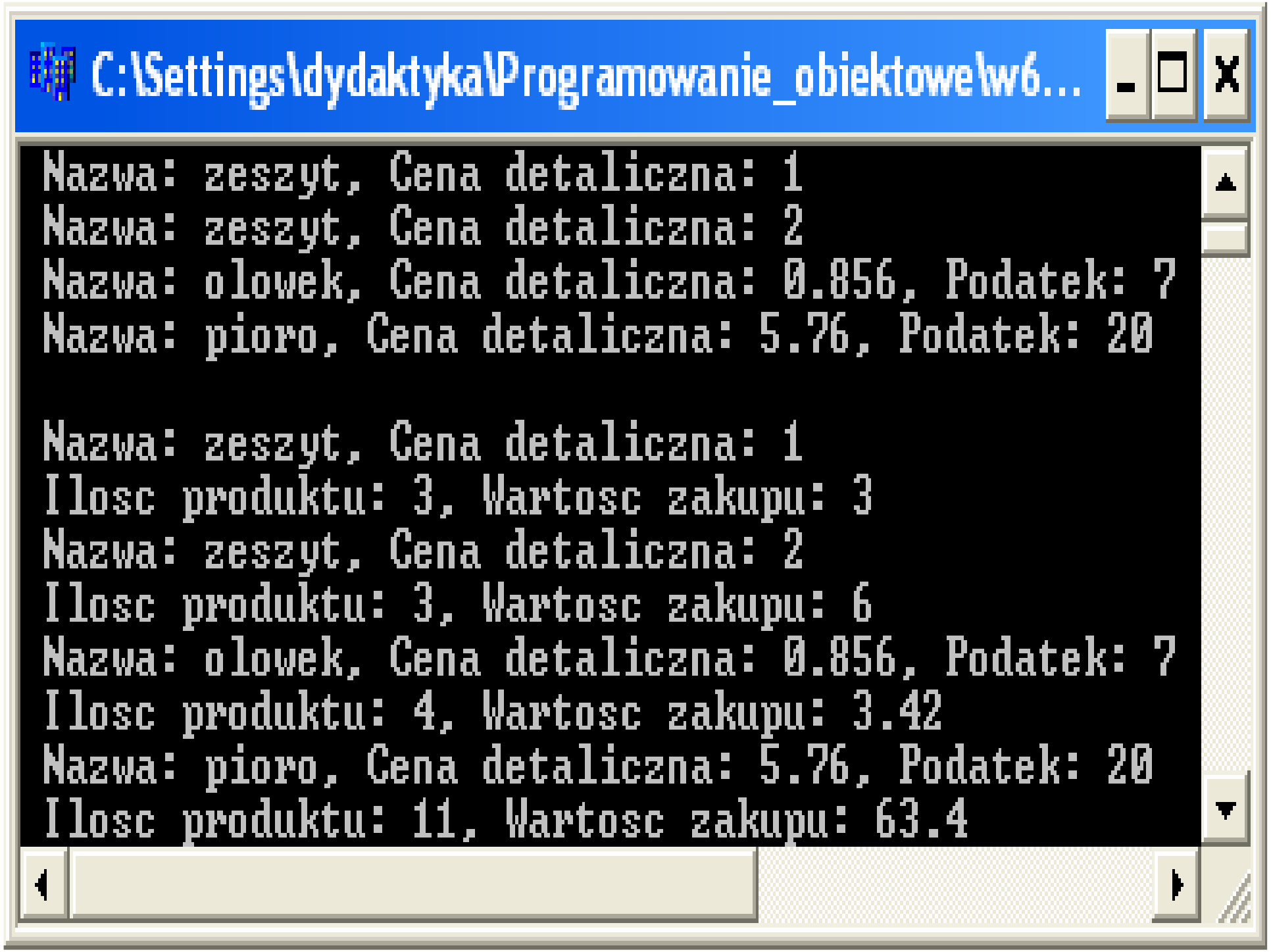
Klasa TKol1 obsługująca kolekcję unikatowych obiektów oparta na obiektach dziedziczących od klasy TAbstrakcyjny



Klasa TKol1 obsługująca kolekcję unikatowych obiektów oparta na obiektach dziedziczących od klasy TAbstrakcyjny


main1.cpp
Program
main1.cpp
Abstrakcyjny.h
PRODUKT1.CPP
PRODUKT2.CPP
ZAKUP.CPP
#include "Zakup.h"
#include "kol1.h"
TKol1 produkty;
TKol1 zakupy;
void Wstaw_zakup(TProdukt1* p, int ilosc);
void main()
{
TProdukt1* p1;
TProdukt2* p2;
p1 = new TProdukt1("zeszyt", 1.0); produkty.Wstaw(p1);
p1 = new TProdukt1("zeszyt", 1.0); produkty.Wstaw(p1);
p1 = new TProdukt1("zeszyt", 2.0); produkty.Wstaw(p1);
p2 = new TProdukt2("olowek",0.80,7); produkty.Wstaw(p2);
p2 = new TProdukt2("pioro",4.80,20); produkty.Wstaw(p2);
p2 = new TProdukt2("pioro",4.80,20); produkty.Wstaw(p2);
9: 2
Modified
Insert


main1.cpp
Program
main1.cpp
Wstaw_zakup(new TProdukt1("zeszyt", 1.0),1);
Wstaw_zakup(new TProdukt1("zeszyt", 1.0),2);
Wstaw_zakup(new TProdukt1("zeszyt", 2.0),3);
Wstaw_zakup(new TProdukt2("olowek",0.80,7),4);
Wstaw_zakup(new TProdukt2("pioro",4.80,20),5);
Wstaw_zakup(new TProdukt2("pioro",4.80,20),6);
//cout<<produkty.toString()<<endl;
//cout<<zakupy.toString()<<endl;
cout<<produkty<<endl;
cout<<zakupy<<endl;
produkty.Usun_kolekcje();
zakupy.Usun_kolekcje();
cin.get();
}
void Wstaw_zakup(TProdukt1* p, int ilosc)
{ TProdukt1* p1;
TZakup* z;
p1=(TProdukt1*)produkty.Podaj(p);
delete p;
if (p1!=NULL)
{z = new TZakup(p1,ilosc);
zakupy.Wstaw(z); }
}
Tworzenie zakupu przez podanie przykładowego produktu p oraz jego ilości.
Obiekt zakup z jest tworzony przez podanie wskaźnika na produkt p1, wyszukanego w kolekcji produkty na podstawie obiektu p, który jest potem usuwany z pamięci
38: 41
Modified
Insert


C:\Settings\dydaktyka\Programowanie_obiektowe\w6...
Nazwa: zeszyt, Cena detaliczna: 1
Nazwa: zeszyt, Cena detaliczna: 2
Nazwa: olowek, Cena detaliczna: 0.856, Podatek: 7
Nazwa: pioro, Cena detaliczna: 5.76, Podatek: 20
Nazwa: zeszyt, Cena detaliczna: 1
Ilosc produktu: 3, Wartosc zakupu: 3
Nazwa: zeszyt, Cena detaliczna: 2
Ilosc produktu: 3, Wartosc zakupu: 6
Nazwa: olowek, Cena detaliczna: 0.856, Podatek: 7
Ilosc produktu: 4, Wartosc zakupu: 3.42
Nazwa: pioro, Cena detaliczna: 5.76, Podatek: 20
Ilosc produktu: 11, Wartosc zakupu: 63.4