Wprowadzenie do programowania obiektowego w C++

1. Główne zasady programowania obiektowego: hermetyzacja, dziedziczenie, polimorfizm
2. Pojęcie klasy: sposoby deklarowania i definiowania składowych klasy
3. Atrybuty dostępu do składowych - **private**, **protected** i **public** wspierające hermetyzację
4. Obiekty automatyczne, statyczne i dynamiczne
5. Konstruktory i destruktory
6. Składowe klasy typu **static**
7. Zmienne referencyjne, przekazywanie parametrów przez referencję, autoreferencja
8. Pliki nagłówkowe, dyrektywy preprocesora, programy wielo-plikowe

Wprowadzenie do programowania obiektowego w C++

1. **Główne zasady programowania obiektowego: hermetyzacja, dziedziczenie, polimorfizm**

1.1. Hermetyzacja (enkapsulacja) - ochrona przed bezpośrednim dostępem do danych zawartych w klasie/obiekcie

Realizacja:

- Do danych umieszczonych w klasie należy się odwołać jedynie za pośrednictwem metod tej klasy/obiektu
- Dostęp do składowych klasy można ograniczać specyfikatorami dostępu: **private, protected, public**

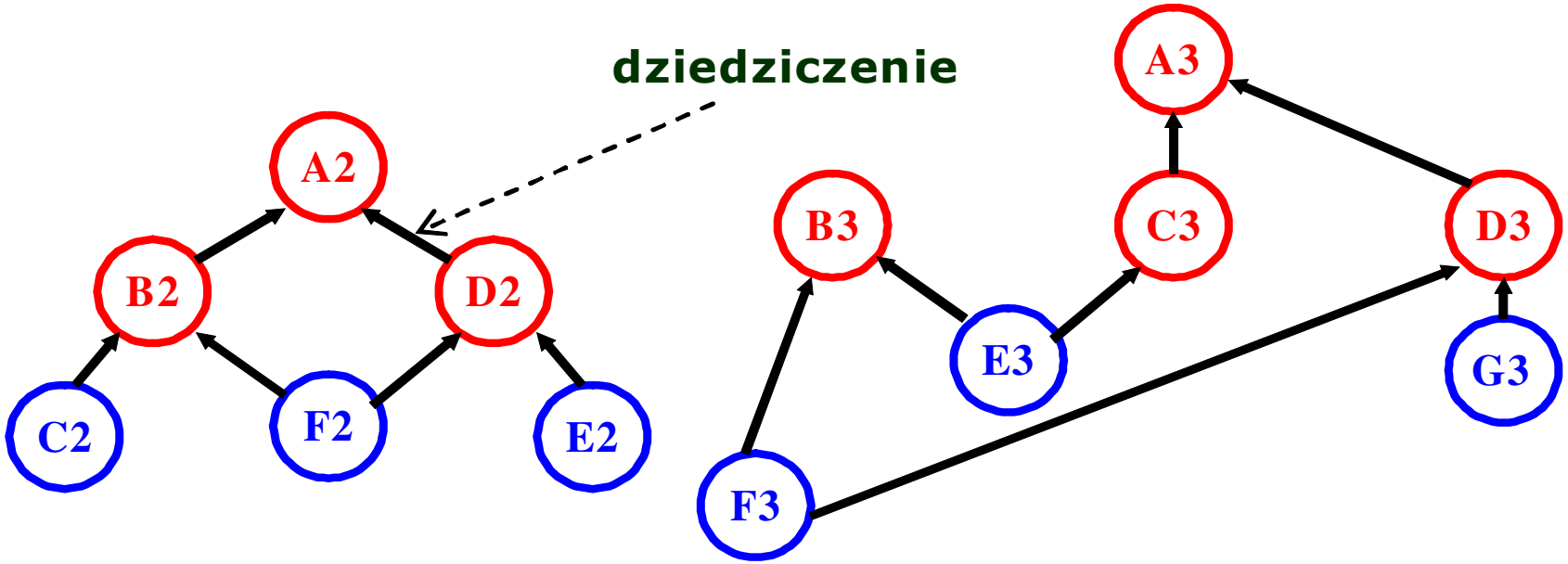
1.2. Dziedziczenie

Budowa nowych klas (nadklas) na podstawie własności klas istniejących (podklas)

Przykłady:

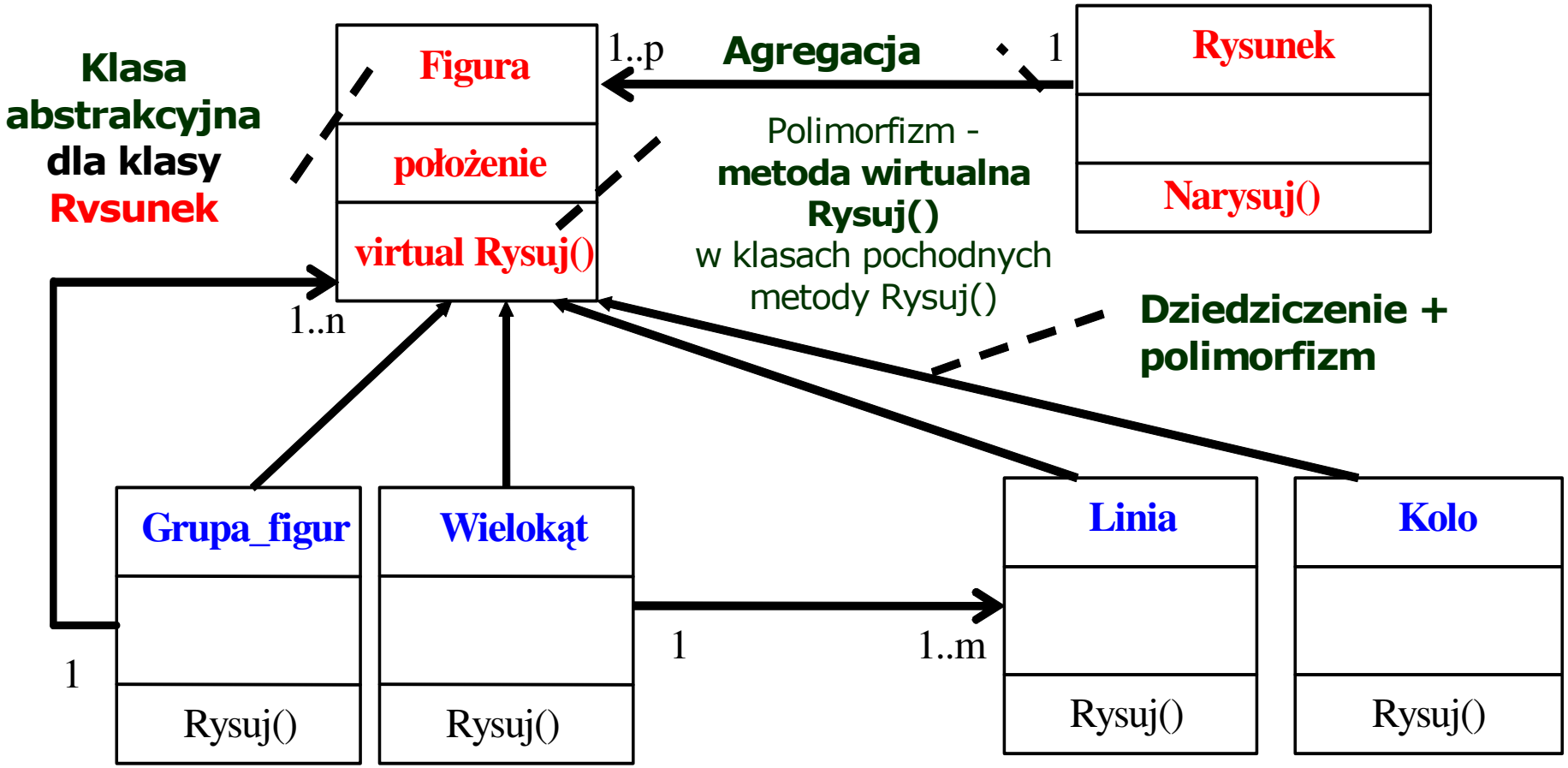
***Drzewo pojazdów*:**
A2-pojazd, B2-samochód,
C2-samochod osobowy,
D2-łódź, E2-motorówka,
F2-amfibia

***Drzewo hierarchii stanowisk*:**
A3-pracownik, C3-projektant, D3-kierownik,
B3-czasowy,
F3-konsultatnt,
E3-czasowy projektant,
G3-dyrektor



1.3. Polimorfizm

Uogólnianie pewnych cech na poziomie korzenia drzewa dziedziczenia w celu uproszczenia obsługi klas/obiektów z całej rodziny



Wprowadzenie do programowania obiektowego w C++

1. Główne zasady programowania obiektowego: hermetyzacja, dziedziczenie, polimorfizm
2. **Pojęcie klasy: sposoby deklarowania i definiowania składowych klasy,**

Klasa jest typem danych definiowanym przez użytkownika

Klasa stanowi połączenie:

1) innych typów danych (predefiniowanych lub definiowanych przez użytkownika)
2) funkcji przeznaczonych do przetwarzania tych danych (metody)

Składnia klasy:

class"|| "struct"|| "union ***nazwa klasy*** **[:*lista klas bazowych*]**

{ *lista składowych klasy* };

lista klas bazowych * - opcjonalna lista klas, od których dziedziczy dana klasa

lista składowych klasy - deklaruje składowe klasy, czyli dane i metody

Przykład klasy
a) Dostęp do składowych private (domyślny), protected i public.
b) Deklaracja składowych wewnątrz klasy.
C) Definicja składowych na zewnątrz klasy – kod metod występuje w jednym miejscu i jest wspólny dla wszystkich obiektów, które są instancją tej klasy

```
//---------------------------------------------------------------------------
 #include <iostream.h>
//---------------------------------------------------------------------------
class TProdukt1
 { //protected:
            string nazwa;
            float cena;
  public:
       float Podaj_cene();
       void Nadaj_cene(float );
       string Podaj_nazwe ();
       void Nadaj_nazwe(string );
       int Porownaj_produkty(TProdukt1& );
       void Wyswietl();
 };
```

składowe są tylko deklarowane wewnątrz ciała klasy

```
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.Podaj_nazwe()  &&
             Podaj_cene() == p.Podaj_cene(); }

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

definicja metod poza ciałem klasy

Wprowadzenie do programowania obiektowego w C++

1. Główne zasady programowania obiektowego: hermetyzacja, dziedziczenie, polimorfizm
2. Pojęcie klasy: sposoby deklarowania i definiowania składowych klasy,
3. **Atrybuty dostępu do składowych - private, protected i public wspierające hermetyzację**

3.1. Deklaracje dostępu do poszczególnych sekcji klasy

lista składowych klasy:
[specyfikator dostępu :] lista składowych klasy

private: **sekcja prywatna** (**domyślna dla class)** - składowe klasy są widoczne tylko w obrębie składowych danej klasy oraz w funkcjach/ klasach zaprzyjaźnionych

protected: **sekcja zabezpieczona** - składowe są widoczne jedynie tylko w obrębie innych składowych klasy, składowych klas dziedziczących oraz w funkcjach/ klasach zaprzyjaźnionych

public: **sekcja publiczna (domyślna dla struct** i **union**)- składowe są widoczne w całym programie

3.2. Deklaracje dostępu do klas podstawowych*

:lista klas bazowych

specyfikator dostępu lista klas bazowych

private: publiczne i chronione składowe klasy podstawowej są prywatnymi składowymi klasy pochodnej, natomiast prywatne stają się niedostępne (wyjątek: klasa pochodna jest zadeklarowana jako **friend** w klasie podstawowej)

public: publiczne oraz chronione składowe klasy podstawowej są publicznymi oraz chronionymi składowymi klasy pochodnej.

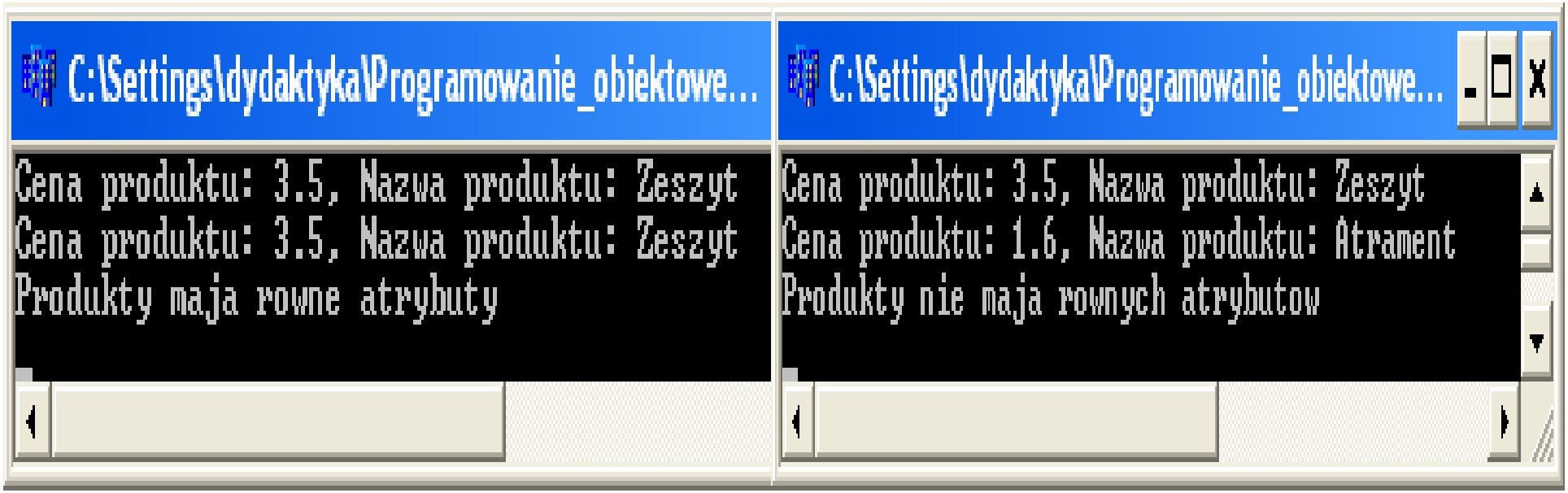
Program 1 - Dostęp do składowych **private** (domyślny), **protected** i **public**

```
//-------------------------------------------------------------------------
 #include <iostream.h>
//-------------------------------------------------------------------------
class TProdukt1
 { //protected:
            string nazwa;
            float cena;
  public:
       float Podaj_cene();
       void Nadaj_cene(float );
       string Podaj_nazwe ();
       void Nadaj_nazwe(string );
       int Porownaj_produkty(TProdukt1& );
       void Wyswietl();
 };
```

```
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.Podaj_nazwe() &&
              Podaj_cene() == p.Podaj_cene(); }

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

```
int main(int argc, char* argv[])
{  TProdukt1 produkt1, produkt2;
    produkt1.Nadaj_cene(3.5);
    produkt1.Nadaj_nazwe("Zeszyt");
    produkt2.Nadaj_cene(1.6);    // 3.5
    produkt2.Nadaj_nazwe("Atrament");  //"Zeszyt"
    produkt1.Wyswietl();
    produkt2.Wyswietl();
    //cout<< produkt1.nazwa<<endl;
    if (produkt1.Porownaj_produkty(produkt2))
            cout<<"Produkty maja rowne atrybuty"<<endl;
    else     cout<<"Produkty nie maja rownych atrybutow"<<endl;
    cin.get();
    return 0; }
```

Można wywoływać składowe typu public w programie. Nie można skompilować programu, gdy atrybut *nazwa* jest typu private lub protected



Wprowadzenie do programowania obiektowego w C++

1. Główne zasady programowania obiektowego: hermetyzacja, dziedziczenie, polimorfizm
2. Pojęcie klasy: sposoby deklarowania i definiowania składowych klasy,
3. Atrybuty dostępu do składowych - **private**, **protected** i **public** wspierające hermetyzację
4. **Konstruktory i destruktory**

Konstruktory - metody o nazwie klasy, wywoływane zawsze podczas tworzenia obiektu

Są to metody domyślne lub jawne zadeklarowane o różnej liście parametrów.

- **Domyślny** zwykły konstruktor bezparametrowy: ***nazwa_klasy();***
- **Domyślny** konstruktor kopiujący: ***nazwa_klasy(nazwa_klasy&);***

- **Jawne** zwykłe konstruktory z parametrami i jeden bez parametrów– albo istnieje jeden domyślny, albo dowolna liczba jawnych konstruktorów
- **Jawny** konstruktor kopiujący jest tylko jeden i posiada identyczny nagłówek jak konstruktor kopiujący domyślny – albo istnieje domyślny albo jawny

***Konstruktor kopiujący* jawny lub domyślny jest wywołany** podczas:

- przekazywania obiektów przez wartość do funkcji/metody
- zwracania obiektu przez wartość jako wyniku funkcji/metody
- inicjowania definiowanego obiektu obiektem tej samej klasy.

***Pozostałe konstruktory* (domyślny lub jawne) są wywoływane** podczas:

- definiowania zmiennych obiektowych
- alokacji pamięci dla obiektów dynamicznych za pomocą operatora **new.**

Destruktor - tylko jedna metoda bezparametrowa
~nazwa_klasy()

- **Może być zdefiniowany jawnie lub jest domyślny.**
  Domyślny destruktor istnieje zawsze, gdy nie zdefiniowanego jawnego destruktora.
  Jawna definicja destruktora likwiduje destruktor domyślny.
  Jawna definicja destruktora jest wymagana do zwolnienia pamięci przeznaczonej na dynamiczne składowe klasy lub wykonania pewnych czynności związanych z usuwanym obiektem.
- **Jest ona wywoływana** zawsze podczas usuwania obiektu z pamięci (podczas wywołania operatora **delete** przy usuwaniu obiektu dynamicznego, dla pozostałych obiektów po wyjściu z bloku ich definicji

Program 2 – konstruktory i destruktor domyślne (dotyczy to także programu 1)

```
//---------------------------------------------------------------------------
 #include <iostream.h>
//---------------------------------------------------------------------------
class TProdukt1
 { // protected:
             string nazwa;
             float cena;
  public:
   float Podaj_cene()
        {  return cena;  }
   void Nadaj_cene(float cena_)
        {  cena = cena_;}
   string Podaj_nazwe ()
        {  return nazwa; }
    void Nadaj_nazwe(string nazwa_)
        {  nazwa = nazwa_;}
    int Porownaj_produkty(TProdukt1& p)
       { return  Podaj_nazwe()==p.Podaj_nazwe() && Podaj_cene()==p.Podaj_cene(); }
    void Wyswietl()
        { cout<<"Cena produktu: "<<cena<<", Nazwa produktu: "<<nazwa<<endl; }
 };
```

Deklaracja i definicja składowych wewnątrz klasy.
Są to domyślnie składowe **typu inline**, których kod „wkleja” się do kodu programu w miejscu ich wywołania – duży program może szybciej działać, ponieważ nie traci czasu na wywołanie metody, ale może mieć większe rozmiary

```
int main(int argc, char* argv[])
{ TProdukt1 produkt1, produkt2;
   produkt1.Nadaj_cene(3.5);
   produkt1.Nadaj_nazwe("Zeszyt");
   produkt2.Nadaj_cene(1.6);    // 3.5
   produkt2.Nadaj_nazwe("Atrament");  //"Zeszyt"
   produkt1.Wyswietl();
   produkt2.Wyswietl();
   //cout<< produkt1.nazwa<<endl;
   if (produkt1.Porownaj_produkty(produkt2))
      cout<<"Produkty maja rowne atrybuty"<<endl;
   else     cout<<"Produkty nie maja rownych atrybutow"<<endl;
   cin.get();
   return 0;
}
```

Można wywoływać składowe typu public w programie. Nie można skompilować programu, gdy atrybut *nazwa* jest typu private lub protected

Program 3 – konstruktory i destruktor jawne -zastosowanie konstruktora kopiującego przy przekazywaniu obiektu przez **wartość** oraz **tworzenia nowego obiektu przez przypisanie** do niego istniejącego obiektu (tworzenie kopii obiektu)

```
//---------------------------------------------------------------------------
 #include <iostream.h>
//---------------------------------------------------------------------------
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 );
      void Wyswietl();
};
```

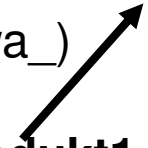
Konstruktor zwykły z parametrami

Konstruktor kopiujący

Destruktor

Tutaj zostanie przekazana kopia obiektu typu TProdukt1 (przekazanie przez wartość) za pomocą wywołanego konstruktora kopiującego – w tym programie jawnego, lecz może to zrobić również konstruktor domyślny

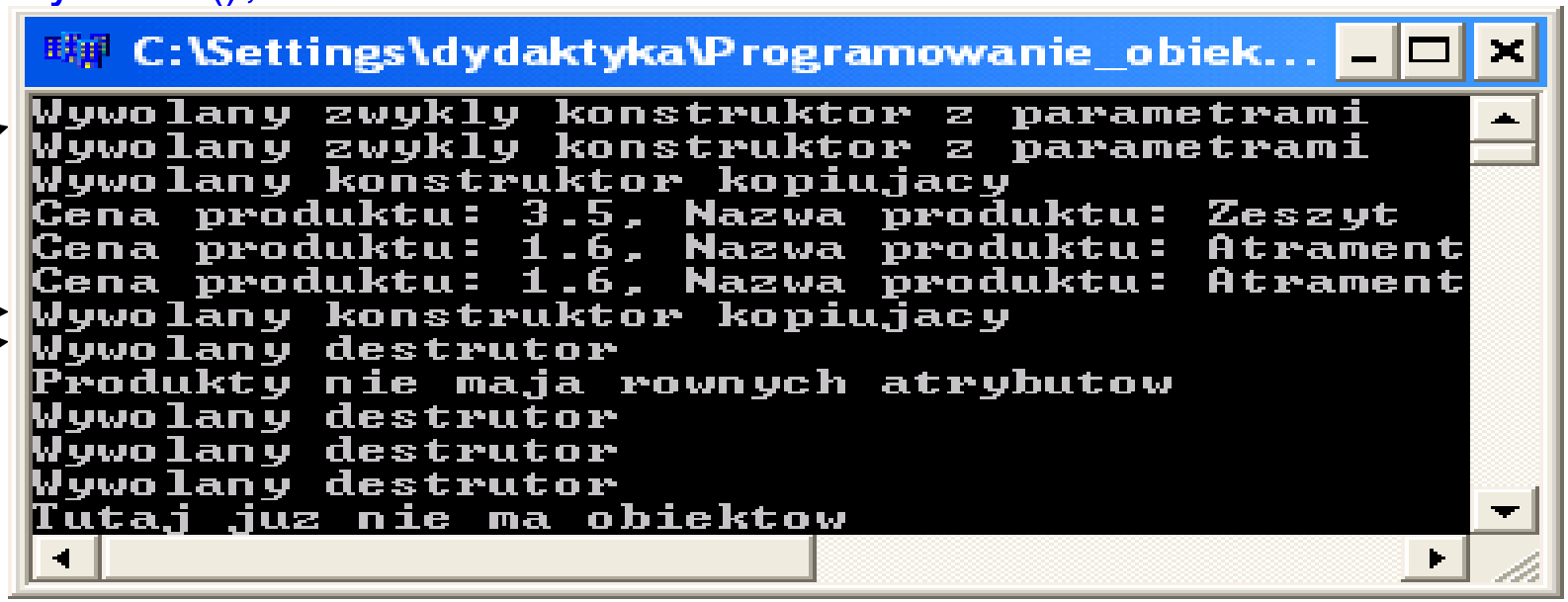
```
TProdukt1::TProdukt1(string nazwa_, float cena_)
 { cout<<"Wywolany zwykly konstruktor z parametrami"<<endl;
   nazwa = nazwa_;
   cena = cena_; }

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.Podaj_nazwe() && Podaj_cene()==p.Podaj_cene(); }

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



Tutaj zostanie przekazana kopia obiektu typu TProdukt1 (przekazanie przez wartość) za pomocą wywołanego konstruktora kopiującego

```
int main(int argc, char* argv[])
{
  { TProdukt1 produkt1("Zeszyt", 3.5), produkt2("Atrament", 1.6);  // 1, 2
    TProdukt1 produkt3 = produkt2; //3
    produkt1.Wyswietl();
    produkt2.Wyswietl();
    produkt3.Wyswietl();
    if (produkt1.Porownaj_produkty(produkt2))      // 4, 5
            cout<<"Produkty maja rowne atrybuty"<<endl;
    else   cout<<"Produkty nie maja rownych atrybutow"<<endl;
   }        // 6, 7, 8
   cout<<"Tutaj juz nie ma obiektow"<<endl;
   //produkt1.Wyswietl();
   cin.get();
   return 0;
}
```

Utworzono nowy obiekt produkt3 jako kopię obiektu produkt2 za pomocą konstruktora kopiującego



Wprowadzenie do programowania obiektowego w C++

1. Główne zasady programowania obiektowego: hermetyzacja, dziedziczenie, polimorfizm
2. Pojęcie klasy: sposoby deklarowania i definiowania składowych klasy,
3. Atrybuty dostępu do składowych - private, protected i public wspierające hermetyzację
4. Konstruktory i destruktory
5. **Obiekty automatyczne, statyczne i dynamiczne**

Rodzaje zmiennych typu obiektowego - obiektów:

Automatyczne - są definiowane (tworzone) wewnątrz dowolnego bloku funkcji. Są usuwane, gdy wychodzi się z danego bloku lub z funkcji.

***Statyczne*:**

- definiowane poza funkcjami jako zmienne globalne (o zasięgu wieloplikowym deklarowane jako **extern**)
- jako globalne zmienne typu **static** o zasięgu jednoplikowym
- wewnątrz dowolnej funkcji, gdy definicja jest poprzedzona słowem **static** (dostępne są tylko wewnątrz działającej funkcji z zachowaniem wartości wyznaczonych podczas poprzedniego wywołania).

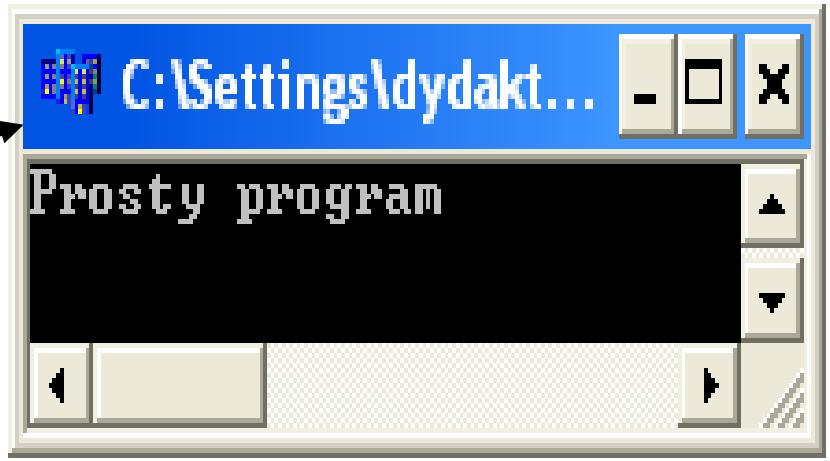
  Istnieją one przez cały czas działania programu i usuwane są po jego zakończeniu.

Dynamiczne* - tworzone jawnie w czasie działania programu na stercie (heap) za pomocą operatora **new** i usuwane za pomocą operatora **delete**

Statyczne, globalne obiekty wejścia **cin** i wyjścia **cout**, operator wejścia >> oraz wyjścia <<

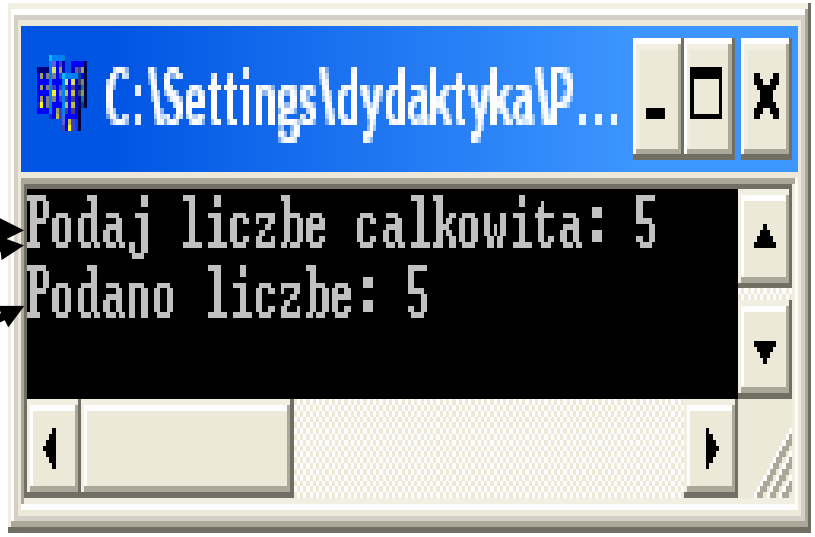
```
//-------------------------------------
#include <iostream.h>
//-------------------------------------
int main(int argc, char* argv[])
{    cout << "Prosty program\n";
     cin.get();
     return 0;
}
//-------------------------------------
```

```
#include <iostream>
extern ostream cout;
ostream cout;
```




```
#include <iostream.h>
//-------------------------------------
int main(int argc, char* argv[])
{
  int liczba;
  cout << "Podaj liczbe całkowita: ";
  cin >> liczba; cin.get();
  cout << "Podano liczbe: " << liczba << endl;
  cin.get();
  return 0;
}
```

```
#include <iostream>
extern istream cin;
istream cin;
```

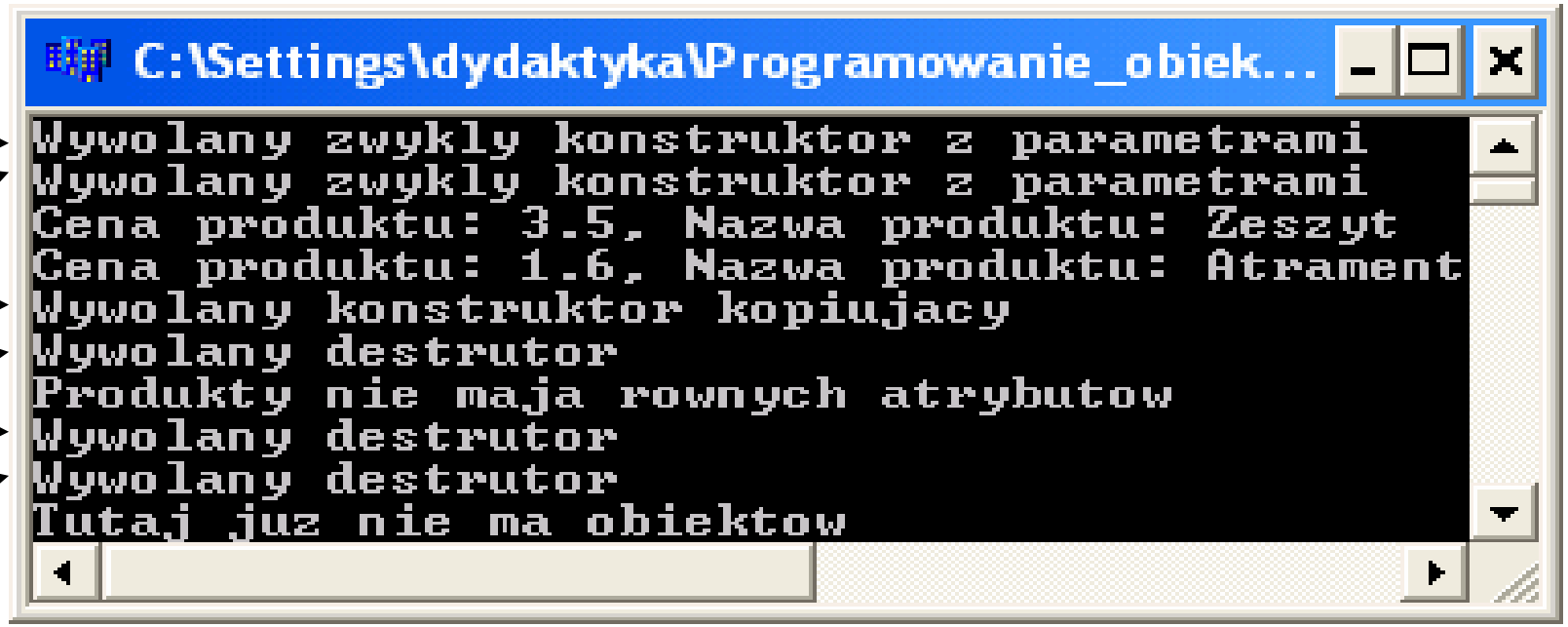


Obiekty automatyczne

```
int main(int argc, char* argv[])    // funkcja main dla klasy TProdukt1 z programu 3
{  //definiowanie automatycznych obiektow
  { TProdukt1 produkt1("Zeszyt", 3.5), produkt2("Atrament", 1.6);  // 1, 2
    produkt1.Wyswietl();
    produkt2.Wyswietl();
    if (produkt1.Porownaj_produkty(produkt2))      // 3, 4
            cout<<"Produkty maja rowne atrybuty"<<endl;
    else   cout<<"Produkty nie maja rownych atrybutow"<<endl;
   }        // 5, 6
   cout<<"Tutaj juz nie ma obiektow"<<endl;
   //produkt1.Wyswietl();
   cin.get();
   return 0;
}
```

W metodzie *Porownaj_produkty* pojawia się obiekt automatyczny

Poza blokiem nie ma obiektów automatycznych

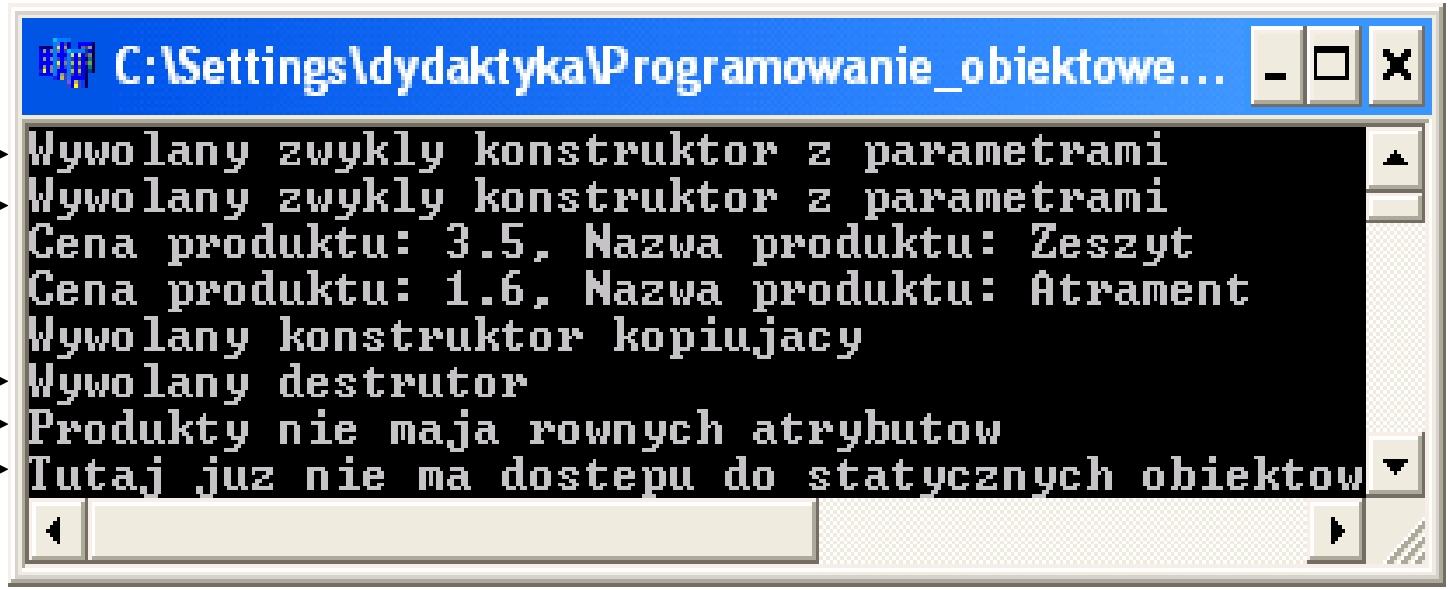


Obiekty statyczne

```
int main(int argc, char* argv[])          // klasa TProduk1 z programu 3
{  //definiowanie statycznych obiektów
  { static TProdukt1 produkt1("Zeszyt", 3.5), produkt2("Atrament", 1.6);    // 1, 2
    produkt1.Wyswietl();
    produkt2.Wyswietl();
    if (produkt1.Porownaj_produkty(produkt2))      // 3, 4
            cout<<"Produkty maja rowne atrybuty"<<endl;
    else   cout<<"Produkty nie maja rownych atrybutow"<<endl;
   }
   cout<<"Tutaj juz nie ma dostepu do obiektow statycznych"<<endl; //5
   //produkt1.Wyswietl();
   cin.get();
   return 0;
}
```

W metodzie Porownaj_produkty pojawia się obiekt automatyczny

Poza blokiem są dalej obiekty statyczne (brak wywołanych destruktorów), jednak nie można wywołać ich metod poza blokiem definicji



Wprowadzenie do programowania obiektowego w C++

1. Główne zasady programowania obiektowego: hermetyzacja, dziedziczenie, polimorfizm
2. Pojęcie klasy: sposoby deklarowania i definiowania składowych klasy,
3. Atrybuty dostępu do składowych - private, protected i public wspierające hermetyzację
4. Konstruktory i destruktory
5. Obiekty automatyczne, statyczne i dynamiczne
6. **Składowe klasy typu static**

Składowe typu static

• W klasie dane i metody mogą być typu **static**. Tylko jedna kopia danych statycznych istnieje dla wszystkich obiektów tej klasy.

• Statyczna **metoda** klasy A klasy globalnej ma zewnętrzny zasięg. Metoda klasy lokalnej nie ma takiego zasięgu. Statyczna metoda jest związana jedynie z klasą, w której jest zadeklarowana. Taka funkcja nie może być wirtualną*.

• Statyczna **metoda** może wywoływać jedynie **inne statyczne składowe typu metody lub dane**. Taka funkcja nie może używać autoreferencji **this**.

•Statyczna **składowa** może być wywołana w metodzie niestatycznej

Program 4 – Zastosowanie składowych typu static

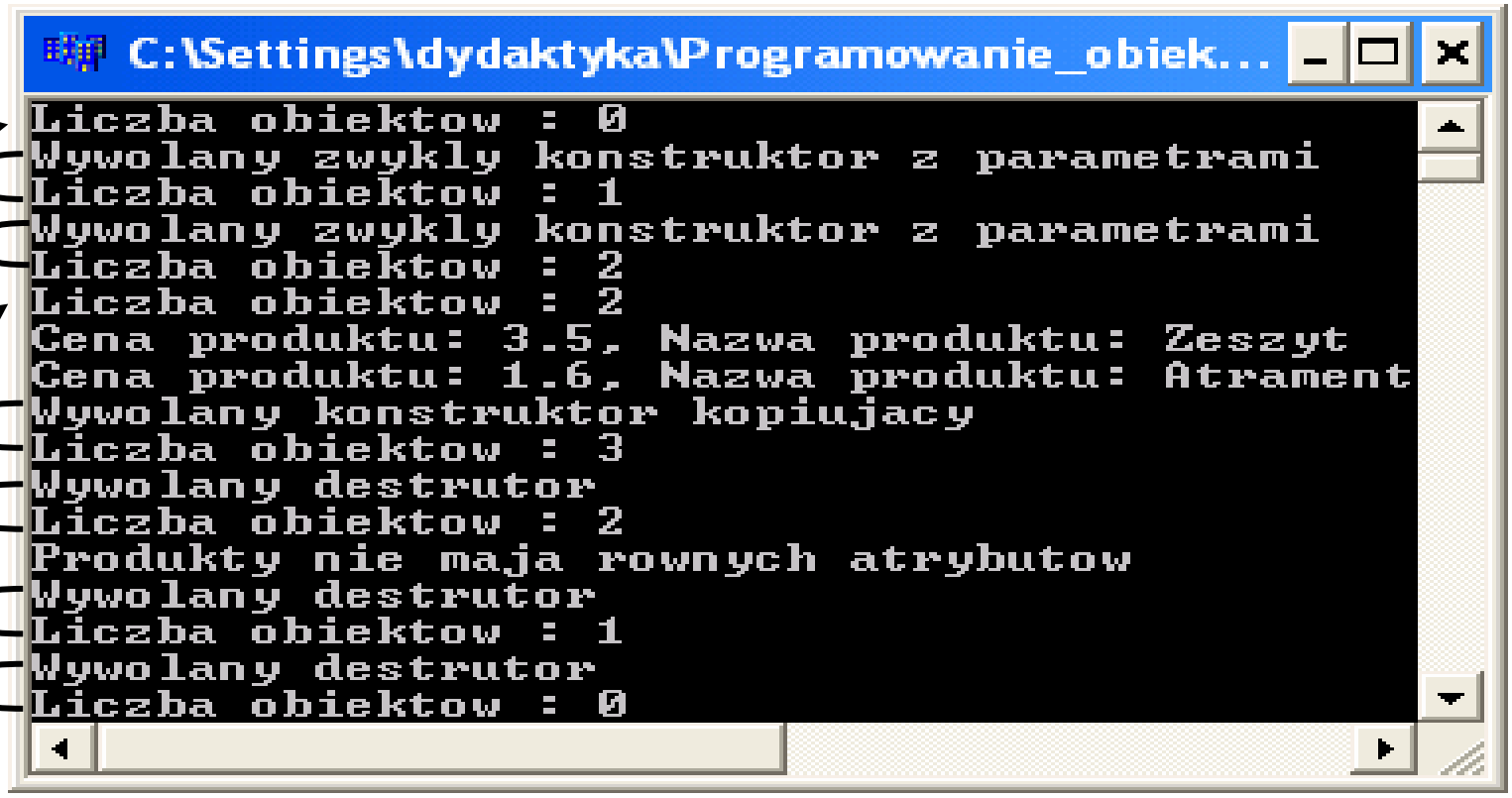
```
//---------------------------------------------------------------------------
 #include <iostream.h>
//---------------------------------------------------------------------------
class TProdukt1
 { protected:
        static int ile_obiektow;
        string nazwa;
        float cena;
  public:
        TProdukt1(string, float );
        TProdukt1(TProdukt1&);
        ~TProdukt1();
        float Podaj_cene();
        void Nadaj_cene(float );
        string Podaj_nazwe ();
        void Nadaj_nazwe(string );
        int Porownaj_produkty(TProdukt1 );
        void Wyswietl();
        static int liczba_obiektow();
 };
```

```
int TProdukt1::ile_obiektow = 0;

int TProdukt1::liczba_obiektow()
  { return ile_obiektow;  }
```

Definicja składowych typu static

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

TProdukt1::TProdukt1(TProdukt1& p)
  { cout<<"Wywolany konstruktor kopiujacy"<<endl;
    nazwa = p.nazwa;
    cena = p.cena;
    ile_obiektow++;
    cout<<"Liczba obiektow : "<<liczba_obiektow()<<endl;  }

TProdukt1::~TProdukt1()
 { cout << "Wywolany destrutor"<<endl;
   ile_obiektow--;
   cout<<"Liczba obiektow : "<<liczba_obiektow()<<endl;  }

  //.....................
}
```

```
int main(int argc, char* argv[])
{  cout <<"Liczba obiektow: "<< TProdukt1::liczba_obiektow()<<endl; // 1
    {  TProdukt1 produkt1("Zeszyt", 3.5), produkt2("Atrament", 1.6);  //2, 3
       cout <<"Liczba obiektow: "<< produkt1.liczba_obiektow()<<endl; // 4
       produkt1.Wyswietl();
       produkt2.Wyswietl();
       if (produkt1.Porownaj_produkty(produkt2))   // 5, 6
              cout<<"Produkty maja rowne atrybuty"<<endl;
       else   cout<<"Produkty nie maja rownych atrybutow"<<endl;
    } // 7, 8
   cin.get();
   return 0;
}
```



Wprowadzenie do programowania obiektowego w C++

1. Główne zasady programowania obiektowego: hermetyzacja, dziedziczenie, polimorfizm
2. Pojęcie klasy: sposoby deklarowania i definiowania składowych klasy,
3. Atrybuty dostępu do składowych - private, protected i public wspierające hermetyzację
4. Konstruktory i destruktory
5. Obiekty automatyczne, statyczne i dynamiczne
6. Składowe klasy typu static
7. Zmienne referencyjne, przekazywanie parametrów przez referencję, autoreferencja

Autoreferencja, zmienne referencyjne, przekazywanie parametrów przez referencję

1) ***Autoreferencja*** **this jest wskaźnikiem do obiektu, istniejącym w momencie wywoływania danej metody.** Autoreferencję this można użyć, aby wskazać składową klasy, gdy istnieje konflikt nazw

2) ***Zmienna referencyjna*** **jest to zmienna tożsama z inną zmienną**

```
int zmienna_1;
int& ref_zmienna_1 = zmienna_1;
```

Zmienna *zmienna_1* jest dostępna pod dwiema nazwami: *zmienna_1* oraz *ref_zmienna_1.* Typ zmiennej referencyjnej **musi być identyczny** z typem zmiennej, z którą jest powiązana.

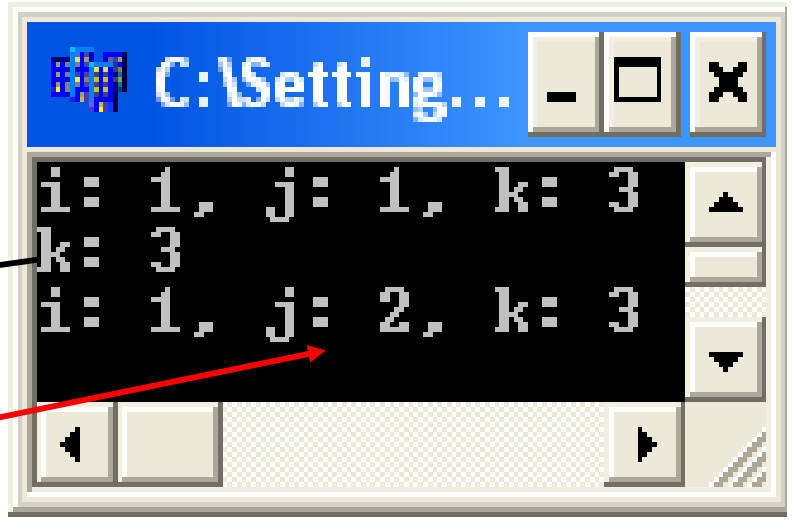
3) **Referencja typu stała**

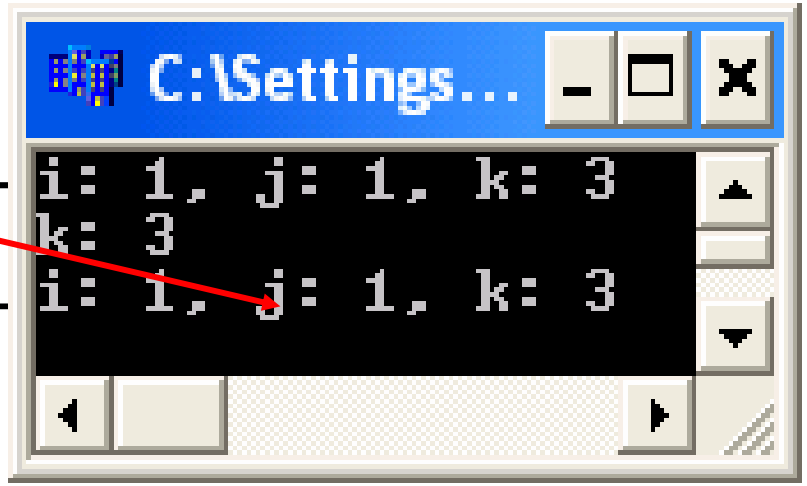
```
const int& ref_stała = 1;
```

Zmienna *ref_stała* ma dostęp do wartości 1 umieszczonej w zmiennej tymczasowej. Nie można zmienić jej wartości.

3) Przekazywanie parametrów w funkcji/metodzie przez wartość i referencję.

```
//---------------------------------------------------------
#include <iostream.h>
//---------------------------------------------------------
  void f (int wart, int& ref, const float& st)
  { wart++;
    ref++;
    cout << "k: "<<st <<'\n';
  }

int main(int argc, char* argv[])
 { int i = 1;
   int j = 1;   // long j =1;
   float k = 3;
   cout<<"i: "<<i<<", j: "<<j<<", k: "<<k<<endl;
   f(i, j, k);
   cout<<"i: "<<i<<", j: "<<j<<", k: "<<k<<endl;
   cin.get();
   return 0;
}
```





Program 5 – Przekazywanie obiektów przez referencję, autoreferencja this

```
//---------------------------------------------------------------------
 #include <iostream.h>
//---------------------------------------------------------------------
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& );
       void Wyswietl();
 };
```

Tutaj zostanie przekazana referencja (adres) do obiektu typu TProdukt1).

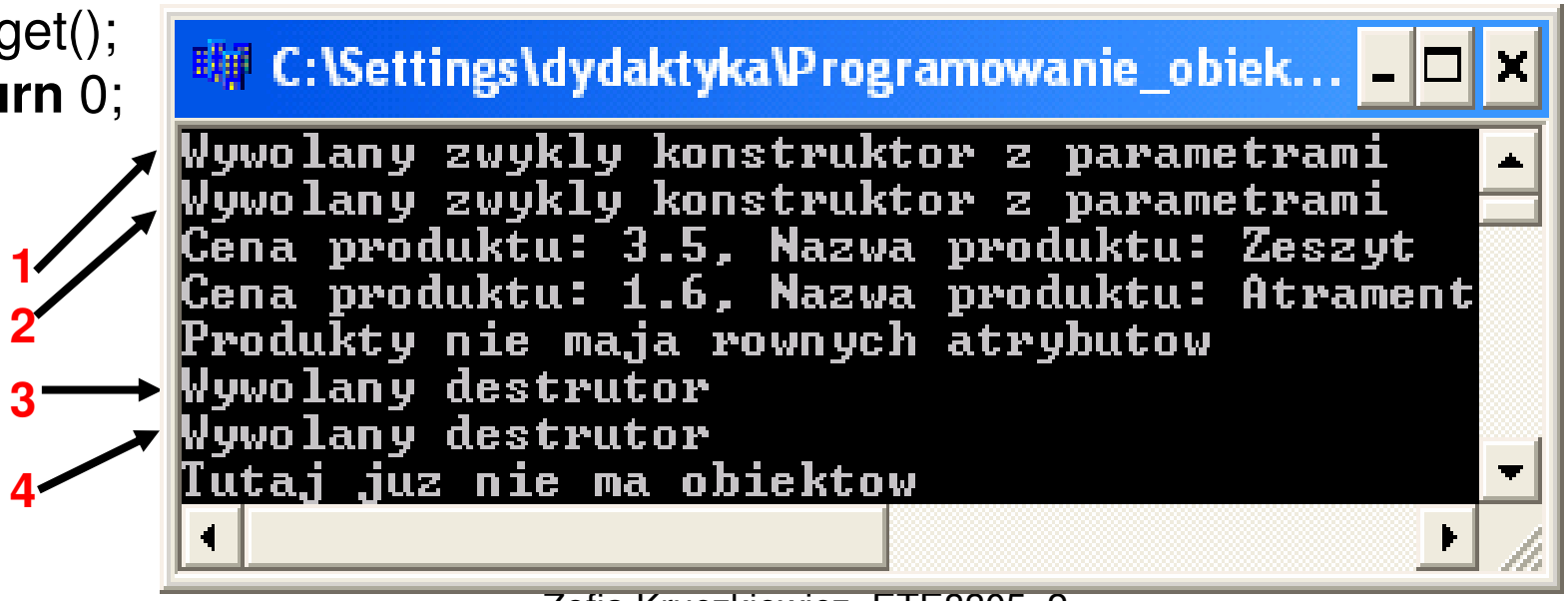
Do tego celu nie wywołuje się konstruktora kopiującego

```
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)
    { this->cena = cena; } // lub (*this).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.Podaj_nazwe() && Podaj_cene()==p.Podaj_cene(); }

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

Autoreferencja this jest wskaźnikiem zawierającym adres obiektu danej klasy, istniejącym w momencie wywołania metody niestatycznej.
this można użyć, aby wskazać składową klasy, gdy istnieje konflikt nazw

Obiekt p jest przekazany przez referencję, dlatego nie jest wywołany konstruktor kopiujący

```
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))
            cout<<"Produkty maja rowne atrybuty"<<endl;
    else   cout<<"Produkty nie maja rownych atrybutow"<<endl;
   } //3, 4
   cout<<"Tutaj juz nie ma obiektow"<<endl;
   //produkt1.Wyswietl();
   cin.get();
   return 0;
}
```



Wprowadzenie do programowania obiektowego w C++

1. Główne zasady programowania obiektowego: hermetyzacja, dziedziczenie, polimorfizm
2. Pojęcie klasy: sposoby deklarowania i definiowania składowych klasy,
3. Atrybuty dostępu do składowych - private, protected i public wspierające hermetyzację
4. Konstruktory i destruktory
5. Obiekty automatyczne, statyczne i dynamiczne
6. Składowe klasy typu static
7. Zmienne referencyjne, przekazywanie parametrów przez referencję, autoreferencja
8. **Pliki nagłówkowe, dyrektywy preprocesora, programy wieloplikowe**

#include <iostrem.h> - dołączenie pliku nagłówkowego z katalogu standardowego
#include „TProdukt1.h” - dołączanie pliku nagłówkowego z katalogu bieżącego

Pliki nagłówkowe powinny zawierać:

- definicje typów np.

```
class punkt
   { float x, y;
     public:
        punkt (float, float);
        void przesun (float, float);
    };
```

- szablony funkcji i klas

```
template <class T>
     class stos { // .....}
```

- prototypy funkcji np.

```
void wyswietl_float(float);
```

- definicje funkcji typu **inline**

```
inline int Większy(int x, int y) {return x > y; }
```

- deklaracje zmiennych

```
extern int zmienna;
```

- definicje stałych

```
const int max = 3;
```

- wyliczenia

```
enum Boolean {False, True};
```

- deklaracje nazw

```
class kolo;
```

- dyrektywy

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

- funkcje makra

```
#define MAX(x, y)    ((x) > (y) ? (x) : (y))
```

- stałe jawne

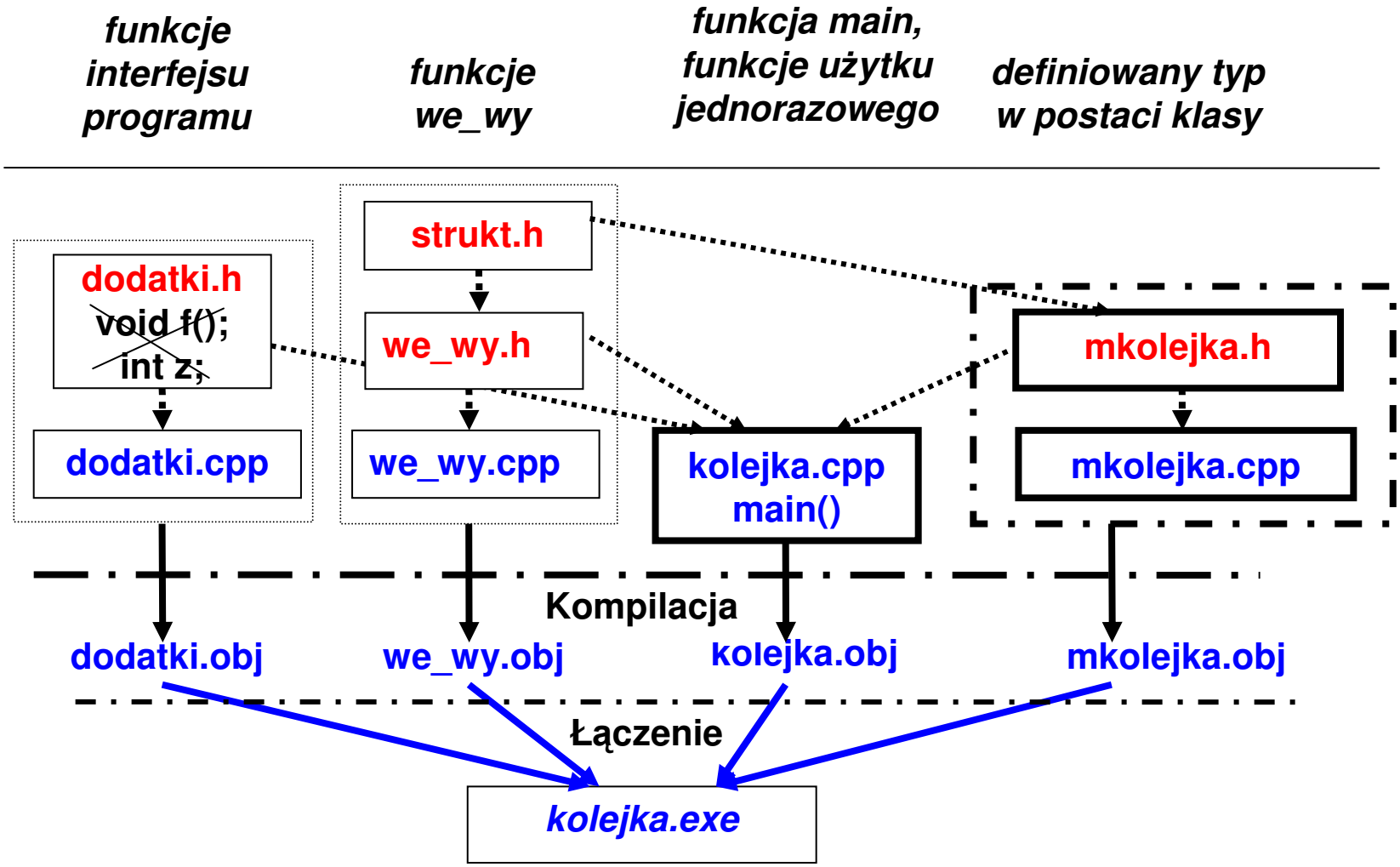
```
#define TRUE 1
```

Uwaga:

Nie należy nigdy wstawiać do pliku nagłówkowego:

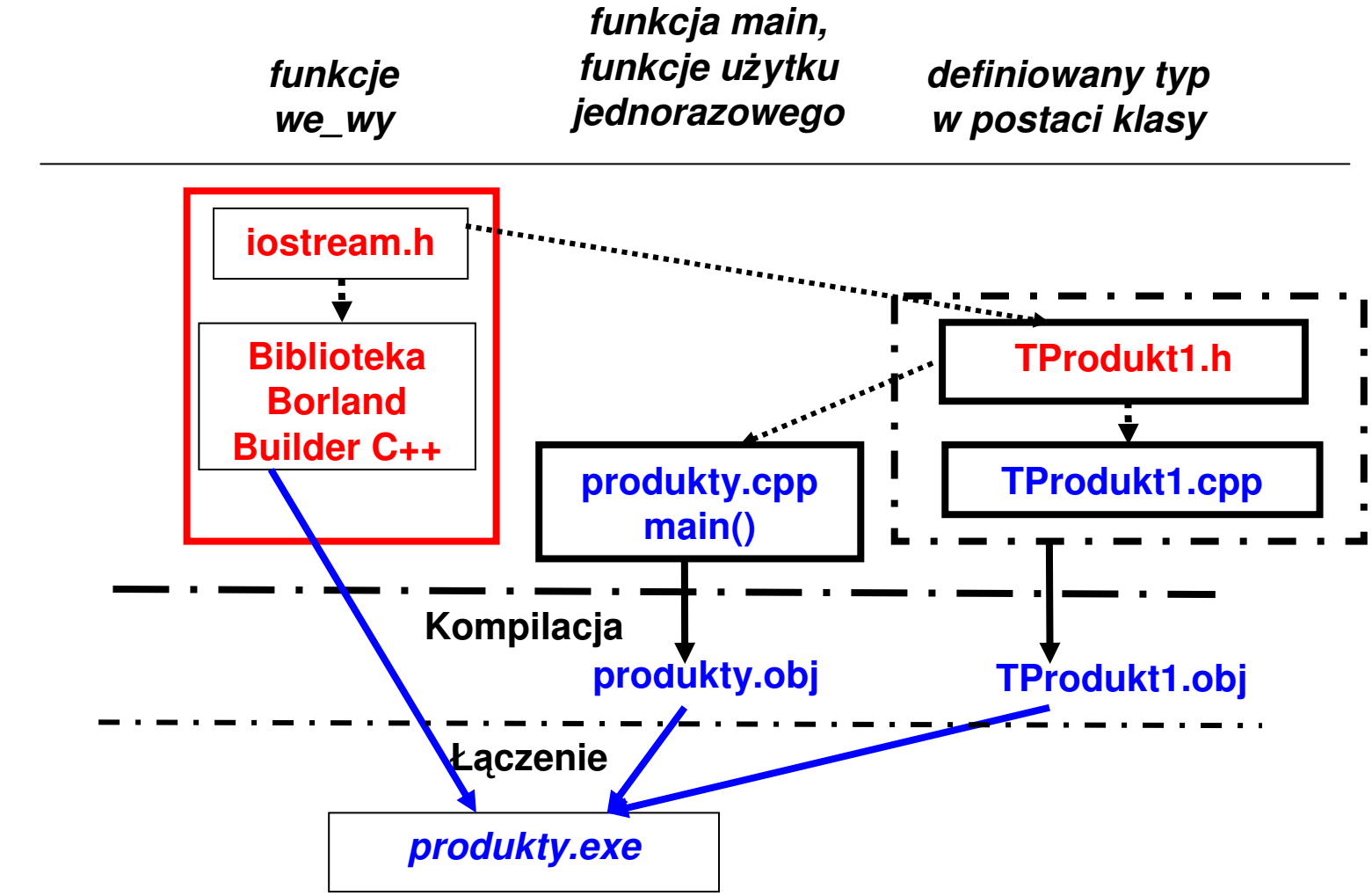
definicji zwykłych funkcji: **int Większy**(**int** x, **int** y) {**return** x > y; }
definicji zmiennych: **int zmienna;**
definicji stałych agregatów np. **const char** tab[] =”aaa”;

Przykład podziału programu na moduły



tworzenie jednego pliku, definiowanie nazw z plików nagłówkowych

Przykład podziału programu prezentowanego na wykładzie na moduły



tworzenie jednego pliku, definiowanie nazw z plików nagłówkowych

Program 6 – program wieloplikowy

```
//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 );
       void Wyswietl();
 };
#endif
```

```
#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.Podaj_nazwe() && Podaj_cene()==p.Podaj_cene(); }

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

```
//---------------------------------------------------------------------------
 #include "TProdukt1.h"          //zawartość pliku z funkcją main
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
  { TProdukt1 produkt1("Zeszyt", 3.5), produkt2("Atrament", 1.6);
    produkt1.Wyswietl();
    produkt2.Wyswietl();
    if (produkt1.Porownaj_produkty(produkt2))
            cout<<"Produkty maja rowne atrybuty"<<endl;
    else   cout<<"Produkty nie maja rownych atrybutow"<<endl;
   }
   cout<<"Tutaj juz nie ma obiektow"<<endl;
   //produkt1.Wyswietl();
   cin.get();
   return 0;
}
```