hex.cpp

```
//-----------------------wskazniki, tablice
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

void main(void)
{
 clrscr();
 int liczba_osemkowa=017;  int liczba_szesnastkowa=0x0f;  int liczba_dziesietna=15;
 cout<<"\n"<<liczba_osemkowa;
 cout<<"\n"<<liczba_szesnastkowa;
 int a=15; int b=24; cout<<"\n\n"<<(a&b);
 a=a<<1;   cout<<"\n\n"<<a;
 getch();
}
```

wsk_zref.cpp

```
//-----------------------wskazniki, tablice
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

int i;
int &ri=i;             //alias name
int *w;

void main(void)
{
 clrscr();
 i=10;   cout<<"\n"<<ri;
 w=&i;                 //wyluskanie wskaznika
 cout<<"\n"<<*w;
 getch();
}
```

wsk_tab

```
//-----------------------wskazniki, tablice
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

int *w;
int i;
int tab[10];

void main(void)
{
 clrscr();
 for(i=0;i<10;i++) tab[i]=i;
 w=tab;
 for(i=0;i<10;i++) {cout<<"\n"<<*w; w++;}
 getch();
 //rownowazne sa zapisy tab[i], *(tab+i) i *(w+i)
 //tab++    :jest bledem, gdyz nazwa tablicy jest stala
}
```

wsk_tab2.cpp

```
//-----------------------wskazniki, tablice
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

int i;
int *tab[10];  //tablica 10 wskaznikow do int
int *w;

void main(void)
{
 clrscr();
 for(i=0;i<10;i++) {tab[i]=new(int); *tab[i]=i;}
 w=tab[3]; //wskaznik do czwartego elementu tablicy (rownego 3)
 cout<<*w;
 getch();
}
```

wsk_blad.cpp

```
//----------------------wskazniki, tablice
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

int tab[10];

void main(void)
{
 clrscr();
 tab[15]=8;
 cout<<tab[15];             //poprawny wynik dzialania zlego programu
 getch();                   //brak kontroli indeksu w jezyku C !!!
}
```

stringi.cpp

```
//----------------------lancuchy tekstowe
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

char tekst[10];
char *wsk;

void main(void)
{
 clrscr();

 tekst[0]='\n';
 tekst[1]='A';
 tekst[2]='l';
 tekst[3]='a';
 tekst[4]='\n ';
 tekst[5]='i';
 tekst[6]=' ';
 tekst[7]='A';
 tekst[8]='s';
 tekst[9]='\0';          //znak konca lancucha
 cout<<tekst;
 wsk="\nAla\ni As";      //tu znak konca lancucha dodaje sie automatycznie
 if(wsk[9]=='\0') cout<<"\n       Jest automatycznie dodany znak '\\0'";
 cout<<wsk;
 *(wsk+1)='O';
 cout<<wsk;
 wsk=tekst;
 cout<<wsk;
 getch();
}
```

przekwar.cpp

```
//----------------------metody przekazywania parametrow do funkcji
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

void fun(int i) {i++;} //przekazywanie przez wartosc (domyslne-jedyne)

void funwsk(int *w) {(*w)++;}  //bez nawiasow priorytet mialby operator ++
                               //przekazanie wskaznika ("wartosci wskaznika")

void funref(int &l) {l++;}    //przekazanie referencji (aliasu)


void main(void)
{
 clrscr();
 int liczba=5;
 fun(liczba);       //parametrem wywolania jest wartosc 5 (brak kontaktu ze zmienna 'liczba')
 cout<<"\nfun(): "<<liczba;  //brak efektu, zmienna 'i' ginie po zakonczeniu fun()
 funwsk(&liczba);        //&-wyluskanie, parametrem wywolania jest wskaznik do zm. 'liczba'
 cout<<"\nfunwsk(): "<<liczba;
 funref(liczba);             //parametrem jest zmienna 'liczba' (kopiowana do zm. 'l')
 cout<<"\nfunref(): "<<liczba;
 getch();
}
```

wsk_fun.cpp

```
//----------------------wskazniki do funkcji
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

void f1(void){cout<<"\nFunkcja f1.\n";}
void f2(void){cout<<"\nFunkcja f2.\n";}
void uruchom(void (*funkcja)(void)) {funkcja();}

void main()
{clrscr();
 uruchom(f1);
 uruchom(f2);
 getch();
}
```

przecfun.cpp

```
//----------------------przeciazanie funkcji (overloading)
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

void fun(int i) {cout<<"\nfun1: i= "<<i;}
void fun(double d) {cout<<"\nfun2: d= "<<d;}

void main(void)
{clrscr();
 fun(5);
 fun(5.0);
 getch();
}
```

szabfun.cpp

```
//--------------------------------szablon funkcji
#include <iostream.h>
#include <conio.h>
//#include <cstring.h>

template <class T>
T gre(T x, T y)
{
 if(x>y) return(x);               //funkcja gre() zadziala dla wszystkich
 else return(y);                  //typow (klas) dla ktorych zdefiniowany jest
}                                 //operator porownania >

void main()
{  clrscr();
        int i=0,j=1;
        cout<<gre(i,j);
        double x=0.22,y=0.88;
        cout<<"\n"<<gre(x,y);
        getch();
}
```

mac.cpp

```
//----------------------tablica dwuwymiarowa
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

int **w;

void main(void)
{
 clrscr();
 int wierszy=3;
 int kolumn=2;
 w=new int* [wierszy];
 for(int i=0;i<wierszy;i++) w[i]=new int [kolumn];
 w[2][1]=8;
 cout<< *(*(w+2)+1);    //w+2 wskazuje na wskaznik do 3 wiersza (jego drugiego elementu)
 getch();               //*(w+2) jest wskaznikiem do 3 wiersza
                        //*(w+2)+1 wskazuje na 2-gi el. w 3 wierszu
}
```

strukt.cpp

```
//----------------------struktury, operatory wskazujace '->' i '.'
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

struct osoba
{
 char *imie;
 char *nazwisko;
 int numer;
};

osoba tab[2];

void main(void)
{
 clrscr();
 osoba o;  osoba *wo;
 o.imie="Jan";o.nazwisko="Kos";o.numer=1;
 wo=&o;
 cout<<wo->nazwisko<<"\n";
 tab[1]=o;
 wo=tab;  wo++;                    //przesuniecie o sizeof(osoba)
 cout<<wo->numer;
 getch();
}
```

rekur.cpp

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

unsigned int silnia(unsigned int n)
{
 if(n==0) return(1);
 else return( n*silnia(n-1) );
}

void main(void)
{
 cout<<silnia(5);
}
```

vector.cpp

```
//-------------------- szablon klasy
#include <iostream.h>
#include <conio.h>

template <class T>
class Vector
{
       T *data;
       int size;
public:
       Vector(int);
       ~Vector( ) { delete[ ] data; }
       T& operator[ ] (int i) { return data[i]; }  //przeciazany operator []
};

template <class T> Vector<T>::Vector(int n)
{
   data = new T[n];
   size = n;
};

int main()
{  clrscr();
       Vector<int> x(3);   // skonstruowanie wektora 3-elementowego typu int
               x[0] = 1; x[1] = 2; x[2] = 3;
               cout<<"\n"<<x[0]<<"\n"<<x[1]<<"\n"<<x[2];
               getch();
       return 0;
}
```

lista.cpp

```
//-----------------------lista jednokierunkowa
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

struct elem {int dane; elem *wn;};
elem *WL;   // WL - wskaznik listy, uchwyt calej listy

void dopisz(int d, elem* &wskaznik_listy)  //funkcja dopisujaca d na koncu listy
{
 elem *wskaznik_nowego;               //utworzenie nowego elementu
 wskaznik_nowego=new(elem);
 wskaznik_nowego->dane=d;
 wskaznik_nowego->wn=NULL;

 elem *wskaznik_pomocniczy;           //dopisanie go na koncu
 wskaznik_pomocniczy=wskaznik_listy;
 if(wskaznik_pomocniczy==NULL) {wskaznik_listy=wskaznik_nowego;return;}
 while(wskaznik_pomocniczy->wn!=NULL) wskaznik_pomocniczy=wskaznik_pomocniczy->wn;
 wskaznik_pomocniczy->wn=wskaznik_nowego;
}

void wyswietl(elem* wskaznik_listy)
{
 elem *wskaznik_pomocniczy;
 wskaznik_pomocniczy=wskaznik_listy;
 while(wskaznik_pomocniczy!=NULL)
  {
        cout<<"\n"<<wskaznik_pomocniczy->dane;
        wskaznik_pomocniczy=wskaznik_pomocniczy->wn;
  }
}

void main(void)
{
 clrscr();
 WL=NULL;
 dopisz(1,WL);  dopisz(2,WL);  dopisz(3,WL);
 wyswietl(WL);
 getch();
}
```

klasa.cpp

```
//-----------------------definicja prostej klasy
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

class prostokat
{
 float a,b;
public:
 prostokat() {cout<<"\nWywolal sie konstruktor nr 1.";a=0;b=0;};
 prostokat(float x,float y) {cout<<"\nWywolal sie konstruktor nr 2.";a=x;b=y;}
 ~prostokat() {cout<<"\nWywolal sie destruktor.";}
 inline float bok_a() {return(a);};
 float bok_b();
 float pole() {return (a*b);};
};

float prostokat::bok_b()
{
 return(b);
}

void main(void)
{
 clrscr();
 prostokat p1; //konstruktor 1
 cout<<"\nProstokat p1:\n"<<p1.bok_a()<<"\n"<<p1.bok_b()<<"\n"<<p1.pole();
 prostokat p2(2,3.4);//konstruktor 2
 cout<<"\nProstokat p2:\n"<<p2.bok_a()<<"\n"<<p2.bok_b()<<"\n"<<p2.pole();
 getch();
} //dwa destruktory
```

prepro.cpp

```
//----------------------dyrektywy preprocesora
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

#define A 1
#define B 0

#define MAX(a,b) (a) > (b) ? (a) : (b)

#pragma argsused    //wylacza ostrzezenie: "parameter x is never used"
int test(void)
{
 int x;
 #ifdef A
 return(1);//jesli A nie jest zdefiniowane - ten fragment sie nie skompiluje
 #endif
 #ifndef A
 return(0);//jesli A jest zdefiniowane - ten fragment sie nie skompiluje
 #endif
}

void main(void)
{
 clrscr();
 int i=MAX(A,B);
 cout<<i;                    //1
 cout<<"\n"<<test();         //1
 cout<<"\n"<<A;              //1
 getch();
}
```

prot_fun.cpp

```
//-----------------------prototypy funkcji
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

void main(void)
{
 clrscr();
 int maks(int,int);  //prototyp funkcji (deklaracja wewnatrz bloku)
 cout<<maks(1,2);
 getch();
}

int maks(int i,int j) //definicja funkcji
{
 return((i>j)?i:j);
}
```

inline.cpp

```
//----------------------- funkcje inline
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <time.h>

inline double fun(double x)
{
 return(x*x);
}

void main(void)
{
 clrscr();
 cout<<fun(2);// bez kladzenia 2 na stos, bez skoku do skompil. f.
 getch();
}
```

arg_opc.cpp

```
//----------------------argumenty opcjonalne, wartosci domyslne
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <time.h>

int sum(int a, int b, int c=10, int mnoznik=1)
{
 return( mnoznik*(a+b+c) );
}

void main(void)
{
 clrscr();
 cout<<"\n"<< sum(0,0);
 cout<<"\n"<< sum(0,0,1);
 cout<<"\n"<< sum(1,1,100,2);
 getch();
}
```

par_main.cpp

```
//----------------------parametry wywolania programu
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <time.h>

void main(int argc, char *argv[])
{
 clrscr();
 cout<<"\nNazwa programu: "<< argv[0];
 cout<<"\nLiczba argumentow: "<<argc-1;
 cout<<"\nArgumenty:";
 for(int i=1;i<argc;i++) cout<<"\n"<<argv[i];
 getch();
}
```

wsk_fun2

```
//----------------------tablica wskaznikow do funkcji
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <time.h>

int (*tabfun[2])();

int fun0() {return(1);}
int fun1() {return(2);}


void main(void)
{
 clrscr();
 tabfun[0]=fun0;
 tabfun[1]=fun1;
 cout<<"\n"<< (*tabfun[0])();
 cout<<"\n"<< (*tabfun[1])();
 getch();
}
```

scope.cpp

```
//-----------------------scope resolution
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

int i=1;

void main(void)
{
  int i=2;
  cout<<i;      //2  - lokalne (z funkcji main)
  cout<<::i;    //1  - glogalne
}
```

void.cpp

```
//-----------------------wykorzystanie void
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

int fun(int i) {cout<<i; return(i);}

void main(void)
{
(void)fun(3);        //nie potrzebna wartosc funkcji - szkoda czasu na jej przekazywanie
}
```

static.cpp

```
//-----------------------pola statyczne klasy (Borland Builder C++ 6.0)
#include <iostream.h>
#include <conio.h>

class punkt{
 int x,y;
 static l_p;  //cecha calej klasy, nie obiektu
public:
 punkt(int ax, int ay);
 ~punkt() {l_p--;}
 static void ile_was_jest(void) {cout<<"Liczba punktow: "<<l_p;}  //f. statyczna
};

punkt::punkt(int ax, int ay)
{
 x=ax; y=ay; l_p++;
}

void main(void)
{
 cout<<"Jeszcze nic sie nie dzialo. ";punkt::ile_was_jest();
 cout<<"Tworzymy punkt";punkt p(1,1); punkt::ile_was_jest();
}
```

try.cpp

```
//-----------------------obsluga wyjatkow (MS Visual C++ 6.0)
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

int i=0;

void main(void)
{
 //clrscr();
 try
 {
  cout<<"Kuku\n"<<6/i;        //Kuku pojawi sie na ekranie
  cout<<"Kuku2";                      //Kuku2 juz sie nie pojawi
 }
 catch(...)
 {cout<<"Nie udało sie";}     //Ten napis tez sie pojawi

 //getch();
}
```

try2.cpp

```
//----------------------obsluga wyjatkow 2 (MS Visual C++ 6.0)
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

int i=0;

class blad
{
public:
    blad() {DZIEL_PRZEZ_ZERO=0;BRAK_PAMIECI=0;};
       int DZIEL_PRZEZ_ZERO;
    int BRAK_PAMIECI;
    void obsluga(void);
};

void blad::obsluga(void)
{
        if(BRAK_PAMIECI==1) cout<<"\nBrak pamieci\n";
        if(DZIEL_PRZEZ_ZERO==1) cout<<"\nDzielenie przez zero\n";
     if((BRAK_PAMIECI==0)&&(DZIEL_PRZEZ_ZERO==0)) cout<<"\nWszystko w porzadku\n";
}

void main(void)
{
 blad B;
 try {
  if(i==0) {B.DZIEL_PRZEZ_ZERO=1; throw(B);}
  cout<<"Kuku\n "<<6/i;
  throw(B);
 }
 catch(blad &BBB)
 { BBB.obsluga();}
 catch(...)
 {cout<<"Nieznany blad";}
}
```

kolejka.cpp

```
//---------------------------kolejka (lista) obiektowa
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>

enum bool{false,true};

template <class element>      // szablon klasy
class Lista
{
 struct ElemListy  //struktura do stworzenia listy
  {
       element wartosc;
       ElemListy *nast;
  } *start;
 public:
  Lista();               //konstruktor
  ~Lista();              //destruktor
  void zKolejki();       //kasuje 1-szy element z kolejki-f pomocnicza do destr

  bool jestPusta() const;     //sprawdzenie czy jest pusty nic nie znienia
  void doKolejki(const element &elem);//dopisuje element na koncu
  element jestPierwszy() const;   //odczytuje elem pocz, nic nie zmienia
};
//*****************METODY (FUNKCJE SKtADOWE KLASY) Lista*************

template <class element>
Lista<element>::Lista()  // konstruktor
{
 start=NULL;
}

template <class element>
Lista<element>::~Lista()      // destruktor
{
  ElemListy *pomoc1=start;
  ElemListy *pomoc2;
```

```
  while(pomoc1->nast!=NULL)
        {
         pomoc2=pomoc1;
         pomoc1=pomoc1->nast;
         delete pomoc2;
        };
  start=NULL;
}

template <class element>
element Lista<element>::jestPierwszy() const
{
 if (start!=NULL) return start->wartosc;    //zwraca wartosc pierwszego elem
 else return 0;
}

template <class element>
bool Lista<element>::jestPusta() const              // sprawdzenie czy jest pusta
{
 if (start==NULL) return true;
 else return false;
}

template <class element>
void Lista<element>::zKolejki()
{
 if (start!=NULL)
 {
  ElemListy *pomoc=start;
  start=start->nast;
  delete pomoc;
 }
}

template <class element>
void Lista<element>::doKolejki(const element &dolaczany)
{
 ElemListy *pomoc;
 ElemListy *nowy = new ElemListy;
 nowy->nast=NULL;
 nowy->wartosc=dolaczany;
 if (start==NULL) start=nowy;
 else
 {
  pomoc=start;
  while (pomoc->nast!=NULL) pomoc=pomoc->nast;
  pomoc->nast=nowy;
 }
}

void main(){.......   //------funkcje main pominieto
```

this.cpp

```
#include <iostream.h>   //-----------------------slowo kluczowe "this"
#include <stdio.h>
#include <conio.h>
class cpx
{
 float re,im;
public:
 cpx() {};
 cpx(float a, float b) {re=a;im=b;};
 cpx(float x) {re=x;im=0;};           //konstruktor jako konwerter
 cpx operator+(cpx zz);
 void wyswietl(void) {cout<<"\n"<<re<<" +j "<<im;};
};
cpx cpx::operator+(cpx zz)
{
 cpx temp;
 temp.re=this->re+zz.re;
 temp.im=this->im+zz.im;
 return(temp);
}
void main(void)
{
 clrscr();
 cpx z1(10,10),z2(20,20),z3;
 z3=z1+z2;
 z3.wyswietl();
 getch();
}
```

konkop.cpp

```
#include <iostream.h>        //----------------------konstruktory kopiujace
#include <stdio.h>
#include <conio.h>
class cpx
{
 float re,im;
public:
 cpx(float a, float b) {re=a;im=b;};
 cpx(cpx &zz) {re=zz.re;im=zz.im;};
 void wyswietl(void) {cout<<"\n"<<re<<" +j "<<im;};
};
void main(void)
{
 clrscr();
 cpx z1(10,10);
 cpx z2(z1);
 z2.wyswietl();
 getch();
}
```

konwer.cpp

```
#include <iostream.h>         //----------------------konwertery typow
#include <stdio.h>
#include <conio.h>

class cpx
{
 float re,im;
public:
 cpx(float a, float b) {re=a;im=b;};
 cpx(float x) {re=x;im=0;};           //konstruktor jako konwerter
 void wyswietl(void) {cout<<"\n"<<re<<" +j "<<im;};
};
void main(void)
{
 clrscr();
 cpx z1(10,10);
 z1.wyswietl();
 float f=5;
 z1=f;                  //konwersja z typu float do typu cpx
 z1.wyswietl();
 getch();
}
```

friend.cpp

```
#include <iostream.h>

class cpx
{
 float re,im;
public:
 cpx(float x, float y) {re=x;im=y;};
 friend void wyswietl(cpx zz);
};

void wyswietl(cpx zz)
{cout<<zz.re<<" +j "<<zz.im;};

void main(void)
{
 cpx z(1,1);
 wyswietl(z);
}
```

mat_obi.cpp

```
/************** JEZYK C++ I JEGO ZASTOSOWANIA ************************/

//Program do wykonywania operacji na macierzach, wierszach i kolumnach

//z wykorzystaniem dziedziczenia.

// Autor: Grzegorz Mzyk
/*-------------------------------------------------------------------------*/
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

/*---------------------Deklaracje-------------------------------------*/
class matrix;    //klasa podstawowa
class kolumna;   //klasa pochodna
class wiersz;    //klasa pochodna

/*----------------Definicja klasy podstawowej--------------------------*/
class matrix {
       double **p;
       int s1,s2;    //ilosc wierszy, ilosc kolumn
public:
       int ub1,ub2;
       matrix(int d1,int d2);    //konstruktor dwuparametrowy
       matrix(matrix&);          //konstrukyor kopiujacy
       matrix () {};             //tego potrzebowaly klasy pochodne
       ~matrix();                //destruktor
       double& element(int i,int j);   //zwraca element, mozna jej cos przypisac
       friend ostream& operator<<(ostream& wy,matrix A);//przeladowanie operatora cout<<
       friend void operator>>(istream& we,matrix& A);//przeladowanie operatora cin>>
       matrix operator=(matrix A);//operator przypisania
       matrix operator*(matrix);//operator mnozenia macierzy przez macierz
       matrix operator+(matrix&);//operator dodawania 2 macierzy
       matrix operator-(matrix&);//odejmowanie macierzy od macierzy
       matrix operator*(double); //mnozenie macierzy przez liczbe
       friend matrix operator*(double,matrix);//mnozenie liczby przez macierz
       friend matrix operator/(double,matrix);//dzielenie liczby przez tablice
       matrix operator/(double);//dzielenie macierzy przez liczbe
       friend wiersz row(matrix,int);//wyciaganie dowolnego wiersza z macierzy
       friend kolumna col(matrix,int);//wyciaganie dowolnej kolumny z macierzy
};
/*---------------Konstruktory i destruktor-------------------------------*/
matrix::matrix(int d1,int d2)
{
       int i,j;
       if(d1<=0||d2<=0) exit(1);
       s1=d1;
       s2=d2;
       p=new double* [s1];
       for(i=0; i<s1;i++) p[i]=new double[s2];
       ub1=s1-1;
       ub2=s2-1;
       //cout<<"\nKonstruktor";
}
```

```
matrix::matrix(matrix& A)
       :s1(A.s1),s2(A.s2)
{       int i,j;
       p=new double* [s1];
       for(i=0; i<s1;i++) p[i]=new double[s2];

       for(i=0;i<s1;i++)
       for(j=0;j<s2;j++)
               p[i][j]=A.p[i][j];

       ub1=s1-1;
       ub2=s2-1;
       //cout<<"\nDziala konstruktor kopiujacy";
}

matrix::~matrix()
{
       int i;
       for(i=0;i<s1;i++) delete p[i];
       delete p;
       //cout<<"\nDestruktor";
}
/*-----------------Funkcja zwracajaca element-----------------------*/
double& matrix::element(int i,int j)
{
       if(i<0||i>ub1||j<0||j>ub2) exit(1);
       return (p[i][j]);
}
/*-------------Definicja operatora mnozacego macierze------------------*/
matrix matrix::operator*(matrix A)
{
       int i,j,k;
       if(s2!=A.s1) {cout<<"\nMnozenie macierzy niemozliwe.\n";
                     exit(1);
                                }
       matrix C(s1,A.s2);
       for(i=0;i<=C.ub1;i++)
       {
        for(j=0;j<=C.ub2;j++)
               C.p[i][j]=0;
       }
       for(i=0;i<=C.ub1;i++)
       {
        for(j=0;j<=C.ub2;j++)
           {
            for(k=0;k<=ub2;k++)
            C.p[i][j]+=p[i][k]*A.p[k][j];
           }
       }
       return(C);
}
/*---------Funkcje mnozenia macierzy przez liczbe i odwrotnie--------*/
matrix matrix::operator*(double A)
{
 matrix temp(s1,s2);
 int i,j;
 for(i=0;i<s1;i++)
   for(j=0;j<s2;j++)
      temp.element(i,j)=element(i,j)*A;
 return(temp);
}

matrix operator*(double A,matrix B)
{
 return(B*A);
}
/*-------------Funkcja dzielenia liczby przez tablice ----------------*/
matrix operator/(double A,matrix B)
{
 int i,j;
 matrix temp(B.s1,B.s2);
 for(i=0;i<B.s1;i++)
   for(j=0;j<B.s2;j++)
      temp.element(i,j)=A/temp.element(i,j);
 return(temp);
}
```

```
/*-------------Definicja dzielenia macierzy przez liczbe-------------*/
matrix matrix::operator/(double A)
{
 matrix temp(s1,s2);
 int i,j;
 for(i=0;i<s1;i++)
   for(j=0;j<s2;j++)
      temp.element(i,j)=element(i,j)/A;
 return(temp);
}
/*--------------Definicja sumowania macierzy-------------------------*/
matrix matrix::operator+(matrix& A)
{
       int i,j;
       if((s1!=A.s1)||(s2!=A.s2)) {cout<<"\nDodawanie macierzy niemozliwe.\n";
                                   exit(1);}
       matrix C(s1,s2);
       for(i=0;i<=C.ub1;i++)
       {
        for(j=0;j<=C.ub2;j++)
           C.p[i][j]=p[i][j]+A.p[i][j];}
       return(C);}

/*---------------Definicja roznicy macierzy---------------------------*/
matrix matrix::operator-(matrix& A)
{
       int i,j;
       if((s1!=A.s1)||(s2!=A.s2)) {cout<<"\nDodawanie macierzy niemozliwe.\n";
                                   exit(1);
                        }
       matrix C(s1,s2);
       for(i=0;i<=C.ub1;i++)
       {
        for(j=0;j<=C.ub2;j++)
           C.p[i][j]=p[i][j]-A.p[i][j];
       }
       return(C);
}
/*---------------Definicja operatora przypisania----------------------*/
matrix matrix::operator=(matrix A)
{
 s1=A.s1;
 s2=A.s2;
 int i,j;
 p=new double* [s1];
 for(i=0; i<s1;i++) p[i]=new double[s2];
  for(i=0;i<s1;i++)
    for(j=0;j<s2;j++)
           p[i][j]=A.p[i][j];
 ub1=s1-1;
 ub2=s2-1;
 return *this;
}
/*---------------cout<<- do wyprowadzania na ekran---------------------*/
ostream& operator<<(ostream& wy,matrix A)
{
    int i,j;
    wy<<"\n";
    for(i=0;i<A.s1;i++)
    { for(j=0;j<A.s2;j++)
      {
      wy<<A.p[i][j]<<"\t";
      }
    wy<<"\n";
    }
   return(wy);
}
/*---------------cin>> - do wprowadzania z klawiatury-------------------*/
void operator>>(istream& we,matrix& A)
{
    int i,j;
    for(i=0;i<A.s1;i++)
    for(j=0;j<A.s2;j++)
    {
    cout<<"\nElement["<<i+1<<","<<j+1<<"]= ";we>>A.p[i][j];
    }
}
```

```
/*--------------KLASA POCHODNA WIERZSZ---------------------------------*/
class wiersz:public matrix
{
public:
                wiersz(int d2);//bo konstruktory nie dziedzicza sie
                double& element(int j);//z jednoelementowa lista argumentow
                                       //reszta funkcji i operatorow dziedziczy sie
                                       // po klasie podstawowej matrix
};
/*--------------Definicja konstruktora wiersza-------------------------*/
wiersz::wiersz(int d2)
 :matrix(1,d2)
{
}

/*---------------------------------------------------------------------*/

double& wiersz::element(int j)

{
       if(j<0||j>ub2) exit(1);
       return(matrix::element(0,j));
}


/*----------------KLASA POCHODNA KOLUMNA-------------------------------*/
class kolumna:public matrix
{
public:
                kolumna(int d1);
                double& element(int i);
};
/*-------------Definicja konstruktora kolumny-------------------------*/
kolumna::kolumna(int d1)
 :matrix(d1,1)
{}
double& kolumna::element(int i)
{
       if(i<0||i>ub1) exit(1);
       return (matrix::element(i,0));
}
/*------Funkcja wyciaga nr-ta kolumne z macierzy A-----------------------*/
kolumna col(matrix A,int nr)
{
 int i;
 if(nr<0||nr>=A.s2) exit(1);
 kolumna temp(A.s1);
 for(i=0;i<A.s1;i++) temp.element(i)=A.element(i,nr);
 return (temp);
}
/*------Funkcja wyciagajaca nr-ty wiersz z macierzy A-------------------*/
wiersz row(matrix A,int nr)
{
 int i;
 if(nr<0||nr>=A.s1) exit(1);
 wiersz temp(A.s2);
 for(i=0;i<A.s2;i++) temp.element(i)=A.element(nr,i);
 return (temp);
}
```

dziedzi.cpp

```
//--------------------------------dziedziczenie
#include <iostream.h>
#include <conio.h>

#define pi 3.1415925

class figura      //klasa podstawowa (bazowa) - abstrakcyjna
{

public:
 void kuku(void) {cout<<"\nKuku";};
 virtual double pole(void)=0;         //funkcja wirtualna

};

class kwadrat:public figura           //klasa pochodna
{
 double a;
public:
 kwadrat(double x) {a=x;};
 double pole(void) {return(a*a);};
};


class kolo:public figura              //klasa pochodna
{
 double r;
public:
 kolo(double x) {r=x;};
 double pole(void) {return(pi*r*r);};
};



void main(void)
{
 clrscr();
 kwadrat KWADRAT(2);
 cout<<"\n"<<KWADRAT.pole();
 kolo KOLO(2);
 cout<<"\n"<<KOLO.pole();

 KOLO.kuku();
 KWADRAT.kuku();
 getch();
}
```