```cpp
// W2.cpp : Defines the entry point for the console application.
//          Przekazanie obiektu klasy przez liste argumentow formalnych
//          przy wywolani funkcji

#include "stdafx.h"
#include <iostream>
#include <cstdlib>
using namespace std;

#define WYWOLANIE_PO_WSKAZNIKU  //zakomentuj, esli chcesz uzyc wywolanie po wartosci

class myclass;    //to jest referencja zapowiadajaca do klasy
void fun(myclass s);
void fun(myclass *s);

class myclass
{
    double *ptr;
    int it;
public:
    myclass(int items);
    ~myclass();
    void put(double a);
    int getnoitems();
    double * get(int i);
    void disp();
};

myclass::myclass(int items)
{
    cout << "Wywolanie konstruktora\n";
    ptr = (double *)malloc(items*sizeof(double));
    if(!ptr)
    {
        cout << "blad alokacji pamieci\n";
    }

    it = 0;
    memset((void *)ptr, 0, items*sizeof(double));
}

myclass::~myclass()
{
    cout << "wywolanie destruktora:  ";

    if(ptr)
    {
        free(ptr);
        ptr = NULL;
        cout << "  free ptr";
        it = 0;
    }

    cout << "\n";
    system("pause");
}

int myclass::getnoitems()
{
    if(!ptr)
        return -1;

    size_t noitems = _msize(ptr);  //rozmiar tablicy
    return (int)noitems/sizeof(double);
}

void myclass::put(double a)
{
    if(ptr)
    {
        int noitems = getnoitems();  //rozmiar tablicy
        if(it >= noitems)
        {
            cout << "tablica jest wypelniona\n";
```

```
            return;
        }

        ptr[it] = a;
        it++;
    }
}

double * myclass::get(int i)
{
    if(ptr)
    {
        int noitems = getnoitems();
        if(i < 0 || i >= noitems)
            return NULL;

        return &ptr[i];
    }

    return NULL;
}

void myclass::disp()
{
    for(int i=0; i<getnoitems(); i++)
    {
        cout << "i=  " << i << "\t" << ptr[i] << "\n";
    }
}


int _tmain(int argc, _TCHAR* argv[])
{
    myclass ob(10);

    for(int it=0; it<10; it++)
    {
        ob.put((double)(it+1));
    }

#ifdef WYWOLANIE_PO_WSKAZNIKU
    fun(&ob);
#else
    fun(ob);
#endif

    return 0;
    //tu destruktor bedzie wywolany
}

void fun(myclass s)
{
    //kompilator tworzy kopie obiektu myclass s
    //konstruktor sie nie wywoluje, przeciez destruktor bedzie wywolany
    //przy wyjsciu z ciala funkcji

    //W kopji wskaznik do tablicy ptr wskazuje na ten samy obszar pamieci,
    //co oryginal, przeciez ma inny adres. Dla tego destruktor przy drugim wywolani
    //bedzie zwalnial pamiec, ktora juz jest zwolniona - PAGE FAULT
    s.disp();

    //Tu bedzie wywolany destruktor -> pamiec dla ptr pozostanie zwolniona
}

void fun(myclass *s)
{
    //kompilator tworzy kopie wskaznika do obiektu, a nie kopie obiektu
    //Dla tego destruktor nie bedzie wywolany przy wyjsciu z ciala funkcji
    s->disp();
}

#undef BAD_VALUE
```