

Cel ćwiczenia: praktyczne zapoznanie się z mechanizmem dziedziczenia.

Należy napisać klasę `vect`, która będzie reprezentować wektor liczb zmiennopozycyjnych. Wewnątrz klasy wektor ma być przechowywany w dynamicznie alokowanej tablicy liczb `double` utworzonej pod wskaźnikiem, który jest zmienną klasy, długość wektora jest przechowywana w drugiej zmiennej składowej klasy. Klasa ma dysponować:

1. konstruktorem domyślnym,
2. konstruktorem dwu argumentowym inicjalizującym wektor tablicą typu `double` o danej długości,
3. konstruktorem jedno argumentowym tworzącym wektor o zadanej długości,
4. konstruktorem kopiującym,
5. destruktor,em,
6. przeładowanym operatorem przypisania,
7. przeładowanym operatorem `[]` umożliwiającym dostęp (odczyt/zapis) pojedynczych elementów wektora o podanym indeksie,
8. bezargumentową funkcją `get_size` zwracającą długość wektora,
9. jedno argumentową funkcją `set_size` zmieniającą długość wektora (zawartość wektora nie jest niszczone, elementy dodawane na końcu są ustawiane na wartość 0),
10. jedno argumentową funkcją `dodaj` dodającą nowy element na końcu wektora, długość wektora wzrasta o 1,
11. przeładować jako zaprzyjaźnione funkcje globalne operatory `==` oraz `!=` porównujące dwa wektory,
12. przeładować jako zaprzyjaźnione funkcje globalne operatory `<<` oraz `>>` obsługujące wejście i wyjście (format wej/wyj ma mieć postać: `[N var1 var2 var3 ... varN]`).

Następnie należy utworzyć klasę `svect` dziedziczącą z klasy `vect`, która reprezentuje posortowany wektor co oznacza że liczby przechowywane wewnątrz wektora w klasie `svect` **zawsze** są posortowane. Pod względem funkcjonalności klasa `svect` ma odpowiadać dokładnie klasie `vect` (ma dysponować tymi samymi funkcjami i operatorami ale ich działanie musi być takie, że zawsze (po wywołaniu jakiegokolwiek funkcji lub operatora) wektor pozostaje posortowany). Klasa `svect` dysponuje jedną dodatkową prywatną bezargumentową funkcją `sort` sortującą wektor (przy jej pisaniu można wykorzystać funkcję biblioteczną `qsort`). Przy tworzeniu klasy `svect` należy skorzystać z faktu że, dziedziczy ona z klasy `vect` i napisać dla niej tylko kod niezbędny do jej poprawnej realizacji.

Program należy podzielić na pięć plików:

1. plik `zad06v.h` zawierający definicję klasy `vect` ,
2. plik `zad06sv.h` zawierający definicję klasy `svect` ,

3. plik zad06v.cpp zawierający definicje funkcji składowych klasy vect ,
4. plik zad06sv.cpp zawierający definicje funkcji składowych klasy svect ,
5. plik zad06main.cpp zawierający testową funkcję główną (podaną poniżej)

Poniżej podany jest kod funkcji main, który ma posłużyć jako test poprawności klas vect i svect oraz rzut ekranu przedstawiający poprawny rezultat zadziałania tej funkcji main:

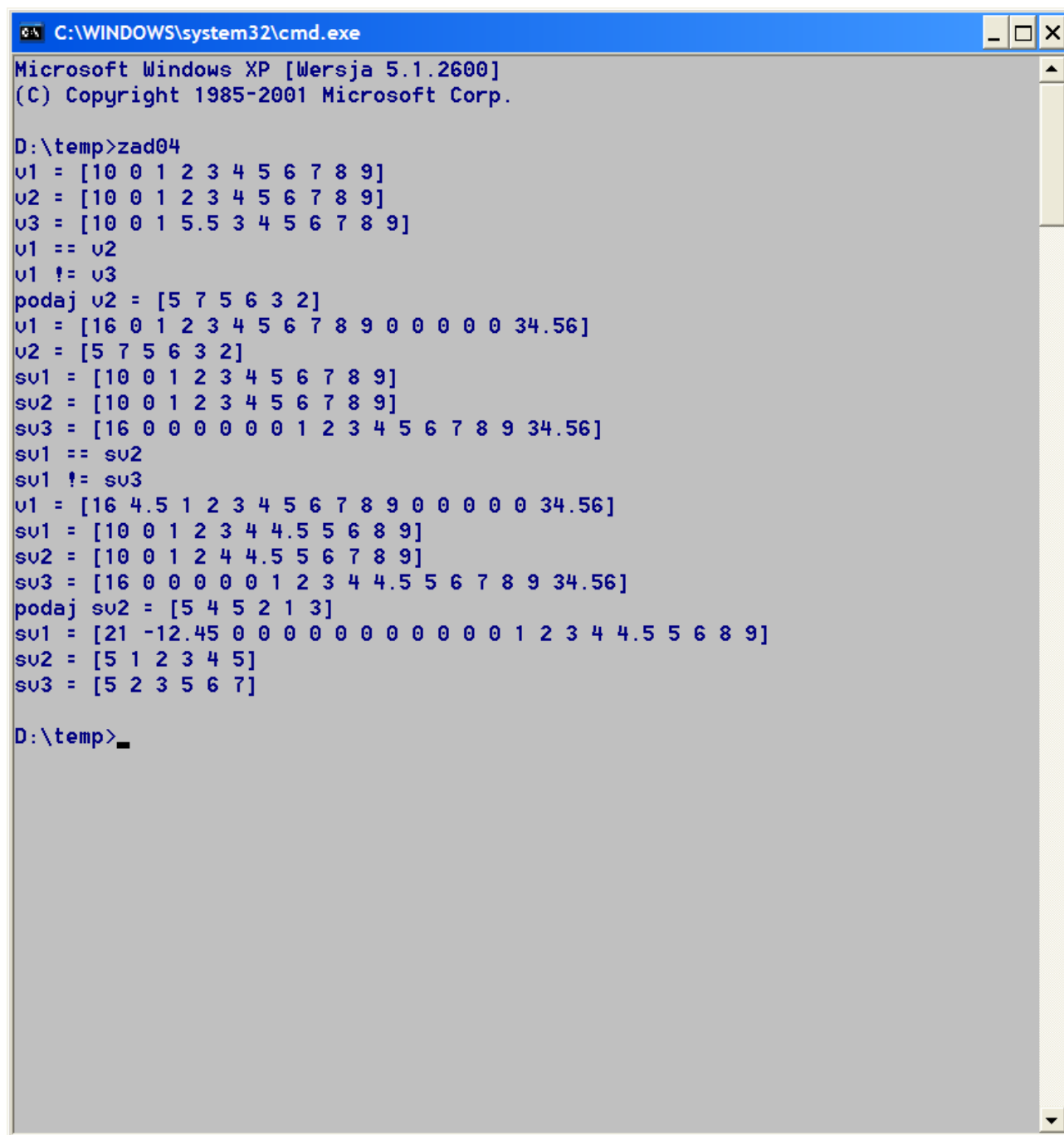
```
// Zadanie szóste - Ćwiczenie z dziedziczenia
// Testowa kompilacja w systemie Linux:
// g++ -O3 -g0 -s -pedantic -ansi -Wall zad06main.cpp zad06sv.cpp zad06v.cpp -o
// zad06.exe
```

```
#include <iostream>
#include "zad06v.h"
#include "zad06sv.h"

using namespace std;

int main(){
//test klasy vect
    int i;
    vect v1(10);
    for (i=0; i<v1.get_size(); i++) v1[i] = i;
    vect v2(v1), v3;
    v3 = v2;
    v3[2] = 5.5;
    cout << "v1 = " << v1 << endl << "v2 = " << v2
        << endl << "v3 = " << v3 << endl;
    if (v1 == v2) cout << "v1 == v2" << endl;
    if (v1 != v3) cout << "v1 != v3" << endl;
    v1.set_size(15);
    v1.dodaj(34.56);
    cout << "podaj v2 = ";
    cin >> v2;
    cout << "v1 = " << v1 << endl << "v2 = " << v2 << endl;
//test klasy svect
    svect sv1(10);
    for (i=0; i<sv1.get_size(); i++) sv1[i] = i;
    svect sv2(sv1);
    svect sv3(v1);
    cout << "sv1 = " << sv1 << endl << "sv2 = " << sv2
        << endl << "sv3 = " << sv3 << endl;
    if (sv1 == sv2) cout << "sv1 == sv2" << endl;
    if (sv1 != sv3) cout << "sv1 != sv3" << endl;
    v1[0] = sv1[7] = sv2[3] = sv3[4] = 4.5;
    cout << "v1 = " << v1 << endl << "sv1 = " << sv1 << endl
        << "sv2 = " << sv2 << endl << "sv3 = " << sv3 << endl;
    sv3 = v2;
    sv1.set_size(20);
    sv1.dodaj(-12.45);
    cout << "podaj sv2 = ";
    cin >> sv2;
    cout << "sv1 = " << sv1 << endl << "sv2 = " << sv2
        << endl << "sv3 = " << sv3 << endl;
    return 0;
}
```

Poniżej rezultat poprawnego zadziałania powyższego kodu:



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Wersja 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

D:\temp>zad04
v1 = [10 0 1 2 3 4 5 6 7 8 9]
v2 = [10 0 1 2 3 4 5 6 7 8 9]
v3 = [10 0 1 5.5 3 4 5 6 7 8 9]
v1 == v2
v1 != v3
podaj v2 = [5 7 5 6 3 2]
v1 = [16 0 1 2 3 4 5 6 7 8 9 0 0 0 0 0 34.56]
v2 = [5 7 5 6 3 2]
sv1 = [10 0 1 2 3 4 5 6 7 8 9]
sv2 = [10 0 1 2 3 4 5 6 7 8 9]
sv3 = [16 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 34.56]
sv1 == sv2
sv1 != sv3
v1 = [16 4.5 1 2 3 4 5 6 7 8 9 0 0 0 0 0 34.56]
sv1 = [10 0 1 2 3 4 4.5 5 6 8 9]
sv2 = [10 0 1 2 4 4.5 5 6 7 8 9]
sv3 = [16 0 0 0 0 0 1 2 3 4 4.5 5 6 7 8 9 34.56]
podaj sv2 = [5 4 5 2 1 3]
sv1 = [21 -12.45 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 4.5 5 6 8 9]
sv2 = [5 1 2 3 4 5]
sv3 = [5 2 3 5 6 7]

D:\temp>
```