

## Zadanie 1

Utwórz klasę `Pojemnik` służącą do przechowywania obiektów typu `Wektor2D`

- Atrybuty tej klasy to dynamiczna tablica typu `Wektor2D` `*mdata` i rozmiar tej tablicy `int msize`.
- Utwórz domyślny konstruktor oraz destruktor.
- Utwórz metodę `Dodaj(const Wektor2D& v)` służącą do dodawania elementów do klasy `Pojemnik`. Konieczne jest w niej dynamiczne przealokowanie tablicy `mdata` (używaj operatorów `new` i `delete`). Przealokowanie możesz wykonać według następującego schematu:
  - zadeklaruj wskaźnik do tablicy tymczasowej;
  - przydziel niezbędną pamięć;
  - przepisuj dane ze starej tablicy;
  - zwolnij pamięć starej tablicy;
  - wstaw nowy element na odpowiednie miejsce w tablicy tymczasowej;
  - przypisz tablicę tymczasową do odpowiedniego wskaźnika obiektu `Pojemnik`.
- W programie głównym uruchom następujący fragment kodu

```
Pojemnik poj;  
poj.Dodaj( v1 );  
poj.Dodaj( v2 );  
poj.Dodaj( Wektor2D(3,2) );  
poj.Dodaj( Wektor2D(8,4) );
```

- Sprawdź przy pomocy debuggera zawartość obiektu `poj`. W oknie `watch` użyj trzech różnych wpisów:
  - `poj.mdata[0]`;
  - `poj.mdata`;
  - `poj.mdata,4`;

## Zadanie 2

Dodaj do klasy `Pojemnik` metodę `void DrukujWszystko()` drukującą zawartość wszystkich obiektów `Wektor2D` przechowywanych w tej klasie. Skorzystaj z metody `Drukuj()` z klasy `Wektor2D`.

## Zadanie 3

Dodaj do klasy `Pojemnik` metodę klasy `Wektor2D` `Suma()` liczącą sumę wszystkich wektorów przechowywanych w klasie `Pojemnik`, korzystając z operatorów zdefiniowanych dla klasy `Wektor2D`. Wynik funkcji wyświetl w programie głównym.

- Spróbuj wywołać liczenie sumy i jej wyświetlanie w taki sposób aby były umieszczone w jednej instrukcji.

## Zadanie 4

Dodaj do klasy `Wektor2D` operator `<<`.

- Operator powinien mieć postać:

```
void operator<<( ostream& o, const Wektor2D& w )
```

i powinien wykonywać te same operacje co funkcja `Drukuj()` klasy `Wektor2D`.

- Wyświetl zawartość dowolnego wektora w programie głównym:

```
cout << v1;
```

- Przerób definicję operatora `<<` tak, aby można było wykonać następującą instrukcję:

```
cout << v1 << v2;
```

- Dodaj do klasy `Pojemnik` nową wersję funkcji `DrukujWszystko()` (np. `DrukujWszystko2()`). Nowa funkcja powinna wykorzystywać operator `<<` zamiast funkcji `Drukuj()` klasy `Wektor2D`.

##Zadanie 5 Sprawdź na czym polega „dekorowanie nazw” przez kompilator („name mangling”).

- W oknie ***Solution Explorer*** zaznacz nazwę projektu. W menu ***Project*** powinno pojawić się na samym dole podmenu ***Properties***.

- W drzewku okna **Property** wybierz folder **linker/debug**. Teraz w polu **Generate Map File** wybierz **Yes (/MAP)**, a poniżej w polu **Map File Name** podaj nazwę pliku **\*\*\*\*.map, np.: \$(OutDir)/Test.map\*\*\*\***.
- Przebuduj program i w katalogu **debug** projektu odszukaj plik **Test.map**.
- Przypatrz się nazwom funkcji wygenerowanym przez kompilator. Czy nawet bez dokładnej wiedzy na temat sposobu generowania tych nazw możesz określić które funkcje należą do których klas? Czy można umieszczać w dwóch różnych klasach funkcje o tych samych nazwach i takich samych parametrach? (Inaczej mówiąc czy linkerowi się one nie pomylą?) Spróbuj odnaleźć konstruktory i destruktory klas **Wektor2D** i **Pojemnik**. W jaki sposób linker odróżni konstruktor domyślny (bezparametrowy) od destruktora?
- Na końcu dekorowanych nazw umieszczona jest informacja o parametrach funkcji. Czy na początku nazw metod klasy umieszczona jest informacja o typach zwracanych przez te metody (np. dla funkcji **Pojemnik::Max()**)? Czym muszą różnić się funkcje przeciążone: czyli funkcje o tych samych nazwach i należących do tej samej klasy?
- Jak myślisz, czy wśród nazw znajdujących się w pliku **\*\*\*\*.map\*\*\*\*** mogą wystąpić dwie identyczne nazwy funkcji?

## Zadanie 6

Dodaj do klasy **Pojemnik** funkcję **Max()** zwracającą największy wektor. O tym, który wektor jest większy, a który mniejszy, powinna decydować klasa **Wektor2D** a nie klasa **Pojemnik**. Zatem klasa **Pojemnik** powinna posługiwać się odpowiednimi operatorami zdefiniowanymi w klasie **Wektor2D**. Zdefiniuj również te operatory. Funkcja **Max()** powinna zadziałać w następujący sposób:

```
cout << poj.Max();
```

## Zadanie 7

Zmodyfikuj program tak aby każda klasa była umieszczona w oddzielnym pliku **.h** i **.cpp**.

## Dodatek (klasa **Wektor2D** z poprzednich ćwiczeń)

```
// -----
// deklaracja klasy
class Wektor2D
{
public:
    Wektor2D();
    Wektor2D( const double& xx, const double& yy );
    void operator+=( Wektor2D& v );
    void Drukuj();

private:
    double x;
    double y;
};
// -----
// konstruktor domyślny
Wektor2D::Wektor2D():x(0), y(0)
{
}
// -----
// konstruktor parametrowy
Wektor2D::Wektor2D( const double& xx, const double& yy ):x(xx), y(yy)
{
}
// -----
// operator +=
void Wektor2D::operator+=( Wektor2D& v )
{
    x += v.x;
    y += v.y;
}
// -----
// drukowanie wartości składowych klasy
void Wektor2D::Drukuj()
{
    cout.setf( ios::fixed );
    cout << "wektor: [";
```

```
    cout.precision(3);  
    // . . .  
  
    // . . .  
    cout.width(6);  
    cout << x << ",";  
    cout.precision(4);  
    cout.width(7);  
    cout << y << "]" << endl;  
}
```