

---

## Zadania z podstaw programowania obiektowego

---

1. Napisać klasę `Lista`, której zadaniem będzie przechowywanie listy liczb całkowitych. Klasa ta ma mieć następujące pola prywatne:

- `int * liczby;` – tablica, w której przechowywane będą liczby,
- `int pojemnosc;` – maksymalna liczba elementów, możliwych do przechowywania,
- `int rozmiar;` – aktualna liczba przechowywanych elementów.

Klasa `Lista` powinna mieć również następujące metody:

- konstruktor z parametrem określającym pojemność, który przydziela pamięć dla tablicy liczb oraz ustala wartości pozostałych pól klasy;
- destruktor zwalniający pamięć przydzieloną dla tablicy liczb;
- metodę `dodajElement`, która przyjmuje dokładnie jeden element – liczbę całkowitą, która dodawana jest do listy; w przypadku, gdy lista jest pełna powinien zostać wyświetlony komunikat o błędzie;
- metodę `znajdz`, której jedynym parametrem powinna być szukana liczba, natomiast wynikiem pozycja podanej liczby w liście (licząc od 0) lub -1, gdy liczby nie ma na liście;
- bezparametrową metodę `pisz`, która wypisuje informacje o liście, w tym jej rozmiar, pojemność oraz listę przechowywanych elementów;
- metodę `usunPierwszy`, która usuwa pierwsze wystąpienie podanej jako parametr liczby, jeżeli znajduje się ona na liście, tzn. jeżeli podana liczba występuje więcej niż jeden raz, to usuwane jest jedynie pierwsze jej wystąpienie;
- metodę `usunPowtorzenia`, która usuwa wszystkie powtórzenia elementów na liście, tzn. po jej wykonaniu na liście nie powinno być żadnych powtórzonych liczb;
- metodę `odwroc`, która odwraca kolejność elementów przechowywanych na liście;
- metodę `zapiszDoPliku`, która zapisuje zawartość listy do pliku tekstowego, którego nazwa podana powinna być jako pierwszy parametr;

Przykładowo, po wykonaniu poniższego fragmentu:

```

int main()
{
    const int N = 10;
    Lista * l = new Lista(N);
    for (int i = 0; i < N/2; ++i) {
        l->dodajElement( (1 << i) );
    }
    l->dodajElement(2);
    l->dodajElement(8);
    l->pisz();
    l->usunPierwszy(2);
    l->pisz();
    for (int i = 0; i < N/2; ++i) {
        l->dodajElement( (1 << i) );
    }
    l->pisz();
    cout << "Po usunięciu powtórzeń:" << endl;
    l->usunPowtorzenia();
    l->pisz();
    cin.get();
    return 0;
}

```

Na ekranie powinno zostać wyświetlone:

```

Lista:
    Pojemność: 10
    Rozmiar: 7
    Elementy: 1 2 4 8 16 2 8
Lista:
    Pojemność: 10
    Rozmiar: 6
    Elementy: 1 4 8 16 2 8
Nie można dodać więcej elementów, lista pełna!
Lista:
    Pojemność: 10
    Rozmiar: 10
    Elementy: 1 4 8 16 2 8 1 2 4 8
Po usunięciu powtórzeń:
Lista:
    Pojemność: 10
    Rozmiar: 5
    Elementy: 16 1 2 4 8

```

2. Napisać klasę Wektor reprezentującą wektor na płaszczyźnie kartezjańskiej. Klasa ta powinna mieć dwie składowe prywatne – współrzędne (x, y) oraz następujące metody publiczne:

- konstruktor bezparametrowy tworzący wektor zerowy,
- konstruktor z parametrami umożliwiającymi ustalenie wartości współrzędnych wektora,
- metodę `dlugosc`, której wynikiem ma być długość wektora,
- metodę `Wektor pomnoz(double a)`, której wynikiem ma być nowy wektor będący wynikiem przemnożenia bieżącego wektora przez podany skalar,
- metodę `Wektor dodaj(const Wektor &w)`, której wynikiem ma być nowy wektor będący sumą bieżącego i podanego wektora,
- metodę `Wektor odejmij(const Wektor &w)`, której wynikiem ma być nowy wektor będący różnicą bieżącego i podanego wektora,
- metodę `Wektor normalizuj()`, której wynikiem powinien być nowy wektor równy znormalizowanemu bieżącemu wektorowi,
- metodę `string toString()`, której wynikiem powinien być łańcuch opisujący wektor, np. "[1.2, -3]"

Przykład zastosowania klasy Wektor:

```
int main(int argc, const char *argv[])
{
    Wektor w1(2, 4),
           w2(1, 0);
    cout << "w1 = " << w1.toString() << " w2 = " << w2.toString() << endl;
    cout << "dł. w1 = " << w1.dlugosc() << " dł. w2 = " << w2.dlugosc() << endl;
    cout << "w1 + w2 = " << w1.dodaj(w2).toString() << endl;
    cout << "w1 - w2 = " << w1.odejmij(w2).toString() << endl;
    cout << "w1 * -2 = " << w1.pomnoz(-2).toString() << endl;
    cout << "w1 po normalizacji = " << w1.normalizuj().toString() << endl;
    cout << "w2 po normalizacji = " << w2.normalizuj().toString() << endl;
    cout << "w1 * 2 - w2 = " << w1.pomnoz(2).odejmij(w2).toString() << endl;
    return 0;
}
```

Wynik działania przykładowego programu:

```
w1 = [2.00, 4.00] w2 = [1.00, 0.00]
dł. w1 = 4.47214 dł. w2 = 1
w1 + w2 = [3.00, 4.00]
w1 - w2 = [1.00, 4.00]
w1 * -2 = [-4.00, -8.00]
w1 po normalizacji = [0.45, 0.89]
w2 po normalizacji = [1.00, 0.00]
w1 * 2 - w2 = [3.00, 8.00]
```

3. Napisać program do obsługi zamówień. W tym celu należy zaimplementować klasy `ElementZamowienia` oraz `Zamowienie`.

Klasa `ElementZamowienia` reprezentuje pojedynczą pozycję zamówienia i zawiera prywatne pola:

- `string nazwa_`
- `double cena_`
- `int liczbaSztuk_`

Dodatkowo klasa ta powinna mieć następujące metody publiczne:

- konstruktor bezparametrowy,
- konstruktor z parametrami umożliwiającymi przypisanie wartości początkowych pólom klasy,
- `string toString()` zwracającą łańcuch znaków opisujący element zamówienia, np. "Chleb 4.00 zł, 2 szt., łącznie 8.00 zł"
- `double obliczKoszt()` zwracającą łączny koszt danej pozycji zamówienia z uwzględnieniem rabatu,
- `double obliczRabat()` wyznaczającą rabat dla klienta, jeżeli klient zamówił co najmniej 5 sztuk tego towaru to rabat wynosi 10%, w przeciwnym razie 0%

Klasa `Zamowienie` zawiera listę zamówień i zawiera następujące pola prywatne:

- `ElementZamowienia *elementy_` – dynamiczna tablica z elementami zamówienia
- `int rozmiar_` – aktualna liczba elementów w zamówieniu
- `int maksRozmiar_` – maks. liczba elementów w zamówieniu

Dodatkowo klasa ta powinna mieć następujące metody publiczne:

- konstruktor z jednym parametrem określającym maks. liczbę elementów zamówienia; w konstruktorze powinna zostać przydzielona pamięć dla tablicy `elementy_`
- destruktor zwalniający pamięć przydzieloną w konstruktorze
- `bool dodaj(const ElementZamowienia &p)` – dodaje podany element do zamówienia (dopisuje do tab. `elementy_`)
- `double obliczKoszt()` – oblicza całkowity koszt zamówienia
- `void pisz()` – wypisuje na ekranie informacje o zamówieniu tj. listę pozycji, łączny koszt oraz naliczony rabat

Przykład zastosowania klas:

```

int main(int argc, const char *argv[])
{
    Zamowienie z;
    z.dodaj(ElementZamowienia("Chleb", 4.0, 2));
    z.dodaj(ElementZamowienia("Mleko", 2.5, 1));
    z.dodaj(ElementZamowienia("Cukier", 4.0, 5));
    z.dodaj(ElementZamowienia("Papierosy", 9.0, 1));
    z.pisz();
    return 0;
}

```

Przykładowy wydruk:

Zamowienie:

```

1. Chleb 4.00 zł, 2 szt., łącznie 8.00 zł
2. Mleko 2.50 zł, 1 szt., łącznie 2.50 zł
3. Cukier 4.00 zł, 5 szt., łącznie 18.00 zł
4. Papierosy 9.00 zł, 1 szt., łącznie 9.00 zł
Koszt całkowity: 37.5 zł
Naliczony rabat: 2 zł

```

4. Napisać klasę Czas służącą do zapamiętania okresu czasu tj. liczby godzin i minut. Klasa ta powinna mieć dwa pola prywatne:

- `int godz_`
- `int minuty_`

oraz metody publiczne:

- konstruktor z parametrami będącymi liczbą godzin i minut,
- konstruktor przyjmujący jako parametr łańcuch znaków na podstawie którego można ustalić wartość godzin i minut np. "12 h 58 min"
- `string toString() const` której wynikiem jest łańcuch znaków opisujący dany okres czasu, np. "29 h 19 min"
- `Czas dodaj(const Czas &t) const` której wynikiem jest nowy obiekt klasy Czas będący sumą bieżącego i podanego jako parametr obiektu
- `Czas odejmij(const Czas &t) const` analogicznie jak dodaj, tyle że odejmowanie,
- `Czas pomnoz(int ile) const` wynikiem ma być okres czasu pomnożony podaną liczbę razy,
- `static Czas sumuj(Czas *tab[], int n)` statyczna metoda klasy służąca do sumowania wszystkich okresów czasu podanych w tablicy będącej pierwszym parametrem

Przykładowy program:

```
int main(int argc, const char *argv[])
{
    Czas t1(10, 56);
    Czas t2(0, 123);
    cout << "t1 = " << t1.toString() << endl;
    cout << "t2 = " << t2.toString() << endl;
    cout << "t1 + t2 = " << t1.dodaj(t2).toString() << endl;
    cout << "t1 - t2 = " << t1.odejmij(t2).toString() << endl;

    Czas *tab[] = { &t1, &t2, &t2 };
    cout << "Czas::sumuj dla t1 + t2 + t2 = "
         << Czas::sumuj(tab, 3).toString() << endl;

    cout << "t1 * 2 = " << t1.pomnoz(2).toString() << endl;

    Czas t3("3 h 17 min");
    cout << "Konstruktor z łańcuchem: " << t3.toString() << endl;
    return 0;
}
```

Wydruk dla przykładowego programu:

```
t1 = 10 h 56 min
t2 = 2 h 3 min
t1 + t2 = 12 h 59 min
t1 - t2 = 8 h 53 min
Czas::sumuj dla t1 + t2 + t2 = 15 h 2 min
t1 * 2 = 21 h 52 min
Konstruktor z łańcuchem: 3 h 17 min
```