Akademska godina 2020/2021. 14. travnja 2021.

OBJEKTNO ORIJENTIRANO PROGRAMIRANJE – 1. KOLOKVIJ

Upute:

Kolokvij se piše 120minuta. Na kolokviju je dopušteno korištenje materijala s predavanja/vježbi i C++ dokumentacija. Upotreba nedozvoljenih materijala poništava ostvarene bodove na kolokviju. Svako prepisivanje je strogo zabranjeno. Rješenja zadataka potrebno je predati preko MS Teamsa.

Main dio zadatka služi kako bi provjerili ispravnost implementacije. Unutar zadatka dopušteno je dodavati metode/funkcije ako smatrate da Vam to može olakšati rješavanje zadatka.

Zadatak 1. – Kompleksni brojevi – trigonometrijski zapis (35 bodova)

Klasa Complex predstavlja kompleksne brojeve prikazane u trigonometrijskom zapisu. Kao takva enkapsulira r (float) modul kompleksnog broja i fi (float) argument kompleksnog broja.

Unutar klase potrebno je implementirati konstruktore *(4 bod)*, *gettere* i *settere (4 bod)* preopteretiti operatore ==, !=, *=, */ ,= , * i / *(svaki po 3 bod)* kao što je predviđeno deklaracijom, implementirati metodu `potencija` *(3 bod)* te je potrebno preopteretiti operator ispisa *(3 bod)*.

Prilikom računanja potrebno je paziti na to da vrijednost argumenta fi mora biti iz intervala [0,2π). Formule za računanje s kompleksnim brojevima su sljedeće:

$$z_1 = r_1(\cos\varphi_1 + i\, sin\varphi_1)$$

$$z_2 = r_2(\cos\varphi_2 + i\, sin\varphi_2)$$

$$z_1 \cdot z_2 = r_1 \cdot r_2(\cos(\varphi_1 + \varphi_2) + i\sin(\varphi_1 + \varphi_2))$$

$$\frac{z_1}{z_2} = \frac{r_1}{r_2}(\cos(\varphi_1 - \varphi_2) + i\sin(\varphi_1 - \varphi_2))$$

$$z_1^n = r_1^n(\cos n\varphi_1 + i\sin n\,\varphi_1)$$

Za π koristiti M_PI, potrebno dodati:

```
#define _USE_MATH_DEFINES
#include <math.h>
```

Deklaracija i main funkcija:

```
class Complex {
protected:
    float r, fi;
public:
    Complex();
    Complex(float, float);
    Complex(const Complex&);

    float getModul() const;
    float getArgument() const;

    void setModul(float);
    void setArgument(float);

    Complex potencija(int);

    Complex& operator=(const Complex&);
    Complex& operator*=(const Complex&);
    Complex& operator/=(const Complex&);

    Complex operator*(const Complex&) const;
    Complex operator/(const Complex&) const;

    bool operator==(const Complex&) const;
    bool operator!=(const Complex&) const;

};


ostream& operator<<(ostream& os, const Complex& c){
};


int main() {

    Complex c1{2,M_PI};
    cout << c1;

    Complex c2{1,1.5*M_PI};

    cout << c2;

    c2*=c1;

    cout << c2;

    Complex c3 = c2.potencija(-3);
    cout << c3;

    c3/=c1;
    cout << c3;

    cout << (c1 == c2) << endl;
    cout << (c1 != c2) << endl;
```

```
    Complex c4 = c2/c3;
    cout << c4;

    Complex c5 = c3 *c4;
    cout << c5;

    return 0;
}
```

Output nakon pokretanja:

```
2(cos 3.14159 + i sin 3.14159)
1(cos 4.71239 + i sin 4.71239)
2(cos 1.5708 + i sin 1.5708)
0.125(cos 1.5708 + i sin 1.5708)
0.0625(cos 4.71239 + i sin 4.71239)
0
1
32(cos 3.14159 + i sin 3.14159)
2(cos 1.5708 + i sin 1.5708)
```

Zadatak 2. Jednostruko povezana lista (35 bodova)

Implementaciju jednostruko povezane liste nadopunite sljedećim metodama:

- `void addKth(int k,int x)` *(13 bod)* – dodavanje elementa na k-to mjesto u listi (ako je k veći od trenutne duljine liste element dodati na kraj liste)
- `void eraseAll(int x)` *(8 bod)* – izbrisati sve elemente s danom vrijednošću unutar liste
- `void eraseDuplicates()` *(14 bod)* – izbrisati sve duplikate unutar liste, nakon pozivanje ove metode nad listom svaka vrijednost se u njoj može pojaviti najviše jednom (kod brisanja duplikata zadržavaju se prve pojave vrijednosti unutar liste)

Deklaracija i main funkcija:

```
class Node
{
public:
    int n;
    Node* next;
    Node(int n, Node* N): n(n), next(N){}
};

class List
{
```

```
protected:
    Node* head{}; // immediately set pointer to nullptr
public:
    List(){}
    void push_front(int);
    void display();

    bool erase(int);

    void eraseAll(int);
    void addKth(int,int);
    void eraseDuplicates();

    ~List();
};


int main()
{
    List L;
    L.push_front(2);
    L.push_front(-1);
    L.push_front(7);
    L.push_front(4);
    L.push_front(-1);
    L.push_front(4);
    L.push_front(7);
    L.push_front(4);

    L.display();

    L.erase(4);
    L.display();

    L.addKth(2,8);
    L.addKth(0,-5);
    L.addKth(20,-7);
    L.display();

    L.eraseAll(4);
    L.display();

    L.eraseDuplicates();
    L.display();

    return 0;
}
```

Output nakon pokretanja (do na memorijsku adresu):

```
4|0059D4E8 -- > 7|0059D7C0 -- > 4|0059D718 -- > -1|0059D788 -- > 4|0059D6E0 -
- > 7|0059D600 -- > -1|0059D558 -- > 2|0000
0000 -- >  null
7|0059D7C0 -- > 4|0059D718 -- > -1|0059D788 -- > 4|0059D6E0 -- > 7|0059D600 -
- > -1|0059D558 -- > 2|00000000 -- >  null
```

```
-5|0059D4E8 -- > 7|0059D7C0 -- > 4|0059D718 -- > -1|0059D590 -- > 8|0059D788
-- > 4|0059D6E0 -- > 7|0059D600 -- > -1|005
9D558 -- > 2|0059D8A0 -- > -7|00000000 -- >  null
-5|0059D4E8 -- > 7|0059D718 -- > -1|0059D590 -- > 8|0059D6E0 -- > 7|0059D600
-- > -1|0059D558 -- > 2|0059D8A0 -- > -7|00
000000 -- >  null
-5|0059D4E8 -- > 7|0059D718 -- > -1|0059D590 -- > 8|0059D558 -- > 2|0059D8A0
-- > -7|00000000 -- >  null
```

Zadatak 3. – Dostavna služba (30 bodova)

U vrijeme pandemije važnu ulogu igra dostavna služba. Kod dostave naglasak je na dvije stvari – paketu koji je potrebno dostaviti i dostavljaču koji ga dostavlja. Kako bi olakšali planiranje dostave potrebno je implementirati dvije klase – Paket i Dostavljac.

U klasi Paket enkapsuliraju se podaci o osobi kojoj se dostavlja paket (prezime - string) i adresi koja je dana u obliku 2d točke (klasa Point implementirana na nastavi i dana u zadatku). Unutar klase potrebno je implementirati konstruktore *(2 bod)* te odgovarajuće *settere* i *gettere (4 bod)* kako je predviđeno danom deklaracijom.

Klasa Dostavljac enkapsulira vektor (STL) objekata tipa Paket, konstruktore *(2 bod)* i metode getPaketi *(2 bod)*, dodajPaket *(4 bod)*, makniPaket *(4 bod)* i napraviRutu *(6 bod)*. Nabrojene metode trebaju raditi sljedeće:

- getPaketi() – vratiti vector paketa (*getter*)
- dodajPaket(Paket&)– ako paket s tom adresom ne postoji potrebno ga je dodati u vector paketa i vratiti true, inače nije potrebno ništa napraviti nego samo vratiti false
- makniPaket(Point&)– ako postoji paket s danom adresom potrebno ga je izbrisati iz vektora paketi i vratiti true inače vratiti false i ništa ne napraviti
- napraviRutu() – mijenja redoslijed paketa unutar vektora paketa kako bi se djelomično optimizirao postupak dostave, redoslijed se pravi na slijedeći način: kao prvi paket u ruti se odabire onaj s najmanjom vrijednošću koordinate x u adresi (ako je takvih više odabire se onaj koji se nalazi prije unutar vektora paketa), a zatim se odabire onaj paket čija euklidska udaljenost adrese je najmanja u odnosu na zadnji paket dodan u rutu, te se postupak ponavlja sve dok svi paketi ne budu dodani u rutu

Dodatno, potrebno je preopteretiti odgovarajuće operatore ispisa za klasu Point i Paket *(4 bod)* kako je predviđeno u deklaraciji te implementirati funkciju ispisPaketa *(2 bod)* koji koristeći for range loop ispisuje proslijeđeni vektor Paketa.

Potrebno uključiti i slijedeće:

```
#include <cmath>
#include <vector>
```

Deklaracija i main funkcija:

```
class Point {
    double x, y;
public:
    Point(): x{}, y{} {};
    Point(double x, double y): x{x}, y{y} {};
    Point(const Point& p): x{p.x}, y{p.y} {};

    void setx(double a) {x = a;}
    void sety(double b) {y = b;};

    double getx() const {return x;};
    double gety() const {return y;};

    double euclideanDistance(const Point& p) const;
};

ostream& operator<<(ostream& out, const Point& p){/* Implementirati */   }

double Point::euclideanDistance(const Point& p) const {
    return sqrt(pow(x - p.getx(), 2.0) + pow(y - p.gety(), 2.0));
}

/* Implementirati */
class Paket{
private:
    string prezime;
    Point adresa;
public:
    Paket(string&, Point&);
    Paket(const Paket&);

    void setPrezime(string&);
    void setAdresa(Point&);

    string getPrezime() const;
    Point getAdresa() const;

};

class Dostavljac{
private:
    vector<Paket> paketi;
public:
    Dostavljac() {};
    Dostavljac(vector<Paket>&);

    bool dodajPaket(Paket&);
    bool makniPaket(Point&);
    void napraviRutu();

    const vector<Paket>& getPaketi() const;
};

ostream& operator<<(ostream& out, const Paket& p){
}
```

```
void ispisPaketa(vector<Paket>& paketi){
}

int main() {

    string prezime = "Horvat";
    Point adresa{2,5};
    Paket p1{prezime, adresa};

    string prezime2 = "Ivanov";
    Point adresa2{-1,2};
    Paket p2{prezime2, adresa2};

    string prezime3 = "Lovric";
    Point adresa3{1,3};
    Paket p3{prezime3, adresa3};

    string prezime4 = "Horvatic";
    Point adresa4{2,5};
    Paket p4{prezime4, adresa4};

    string prezime5 = "Bilic";
    Point adresa5{3,1};
    Paket p5{prezime5, adresa5};

    string prezime6 = "Nakic";
    Point adresa6{13,2};
    Paket p6{prezime6, adresa6};

    Dostavljac d{};
    cout << d.dodajPaket(p1) << endl;
    cout << d.dodajPaket(p2) << endl;
    cout << d.dodajPaket(p3) << endl;
    cout << d.dodajPaket(p4) << endl;
    cout << d.dodajPaket(p5) << endl;
    cout << d.dodajPaket(p6) << endl;

    ispisPaketa(d.getPaketi());

    Point adresa_makni1{-2,3};
    Point adresa_makni2{3,1};

    cout << d.makniPaket(adresa_makni1) << endl;
    cout << d.makniPaket(adresa_makni2) << endl;

    cout << "-------------------------"<< endl;

    ispisPaketa(d.getPaketi());

    cout << "-------------------------"<< endl;
    d.napraviRutu();

    ispisPaketa(d.getPaketi());

    return 0;
}
```

Output nakon pokretanja:

```
1
1
1
0
1
1
Horvat: (2, 5)
Ivanov: (-1, 2)
Lovric: (1, 3)
Bilic: (3, 1)
Nakic: (13, 2)

0
1
-------------------------
Horvat: (2, 5)
Ivanov: (-1, 2)
Lovric: (1, 3)
Nakic: (13, 2)
-------------------------
Ivanov: (-1, 2)
Lovric: (1, 3)
Horvat: (2, 5)
Nakic: (13, 2)
```