1. ZADATAK

PRII – G1

Izvršiti definiciju funkcija na način koji odgovara opisu (komentarima) datom neposredno uz pozive ili nazive funkcija. Možete dati komentar na bilo koju liniju code-a koju smatrate da bi trebalo unaprijediti ili da će eventualno uzrokovati grešku prilikom kompajliranja. Također, možete dodati dodatne funkcije ili metode koje će vam olakšati implementaciju programa.

```
#include <iostream>
using namespace std;

char* AlocirajTekst(const char* tekst) {
    if (!tekst) return nullptr;
    size_t vel = strlen(tekst) + 1;
    char* temp = new char[vel];
    strcpy_s(temp, vel, tekst);
    return temp;
}
string crt = "\n-------------------------------------------------------
--------------------------\n";

string PORUKA_TELEFON = crt +
"TELEFONE ISKLJUCITE I ODLOZITE U TORBU, DZEP ILI DRUGU LOKACIJU VAN
DOHVATA.\n"
"CESTO SE NA TELEFONIMA (PRO)NALAZE PROGRAMSKI KODOVI KOJI MOGU BITI
ISKORISTENI ZA\n"
"RJESAVANJE ISPITNOG ZADATKA, STO CE, U SLUCAJU PRONALASKA, BITI
SANKCIONISANO." + crt;

string PORUKA_ISPIT = crt +
"0. PROVJERITE DA LI ZADACI PRIPADAJU VASOJ GRUPI (G1/G2)\n"
"1.  SVE  KLASE  SA  DINAMICKOM  ALOKACIJOM  MORAJU  IMATI  ISPRAVAN
DESTRUKTOR\n"
"2. IZOSTAVLJANJE DESTRUKTORA ILI NJEGOVIH DIJELOVA BIT CE OZNACENO
KAO TM\n"
"3. ATRIBUTI, METODE I PARAMETRI MORAJU BITI IDENTICNI ONIMA U TESTNOJ
MAIN FUNKCIJI,"
"   OSIM AKO POSTOJI JASAN RAZLOG ZA MODIFIKACIJU\n"
"4. IZUZETKE BACAJTE SAMO TAMO GDJE JE IZRICITO NAGLASENO\n"
"5. SVE METODE KOJE SE POZIVAJU U MAIN-U MORAJU POSTOJATI.\n"
"    AKO NEMATE ZELJENU IMPLEMENTACIJU, OSTAVITE PRAZNO TIJELO ILI
VRATITE NEKU DEFAULT VRIJEDNOST\n"
"6.  RJESENJE  KOPIRAJTE  U  .DOCX  DOKUMENT  (NAZIV  DOKUMENTA  =  BROJ
INDEKSA, npr. IB150051.docx)\n"
"7.  NA  FTP  SERVER  POSTAVITE  SAMO  .DOCX  DOKUMENT,  A  NE  CIJELI
PROJEKAT\n"
"8. SVE NEDOZVOLJENE RADNJE TOKOM ISPITA BIT CE SANKCIONISANE\n"
"9.  U  MAIN  FUNKCIJI  MOZETE  DODAVATI  TESTNE  PODATKE  I  POZIVE  PO
VLASTITOM IZBORU\n"
"10. KORISTITE VISUAL STUDIO 2022 (C++17) I TESTIRAJTE PROGRAM U OBA
MODA (F5 i Ctrl+F5)\n"
"11.  NA  KRAJU  PROVJERITE  DA  LI  STE  RJESENJE  ISPITA  POSTAVILI  U
ODGOVARAJUCI FOLDER NA FTP SERVERU" + crt;

enum Kategorija { ELEKTRONIKA, KNJIGE, MODA, PREHRANA };
const  char*  KategorijaNazivi[]  =  {  "ELEKTRONIKA",  "KNJIGE",  "MODA",
"PREHRANA" };
```

```
template<class T1, class T2, int max>
class KolekcijaParova {
    T1* _prvi[max] = { nullptr };
    T2* _drugi[max] = { nullptr };
    int _trenutno = 0;
public:
    KolekcijaParova() = default;
    ~KolekcijaParova() {
        for (int i = 0; i < _trenutno; i++) {
            delete _prvi[i]; _prvi[i] = nullptr;
            delete _drugi[i]; _drugi[i] = nullptr;
        }
    }
    int GetTrenutno() const { return _trenutno; }
    T1& GetPrvi(int indeks) { return *_prvi[indeks]; }
    T2& GetDrugi(int indeks) { return *_drugi[indeks]; }
    T1& operator[](int indeks) { return *_prvi[indeks]; }
    friend ostream& operator<<(ostream& COUT, KolekcijaParova& obj) {
        for (int i = 0; i < obj._trenutno; i++)
            COUT << obj.GetPrvi(i) << " " << obj.GetDrugi(i) << "\n";
        return COUT;
    }
};

class DatumVrijeme {
    int* _godina, * _mjesec, * _dan, * _sati, * _minute, * _sekunde;
public:
    DatumVrijeme(int dan = 1, int mjesec = 1, int godina = 2000, int
sati = 0, int minute = 0, int sekunde = 0) {
        _godina = new int(godina);
        _mjesec = new int(mjesec);
        _dan = new int(dan);
        _sati = new int(sati);
        _minute = new int(minute);
        _sekunde = new int(sekunde);
    }
    ~DatumVrijeme() {
        delete _godina; delete _mjesec; delete _dan;
        delete _sati; delete _minute; delete _sekunde;
    }
};

class Proizvod {
    char* _naziv;
    Kategorija _kategorija;
    int _cijena;
public:
    Proizvod(const char* naziv = "", Kategorija kategorija =
ELEKTRONIKA, int cijena = 0)
        : _kategorija(kategorija), _cijena(cijena) {
        _naziv = AlocirajTekst(naziv);
    }
    ~Proizvod() { delete[] _naziv; }

    const char* GetNaziv() const { return _naziv; }
    Kategorija GetKategorija() const { return _kategorija; }
    int GetCijena() const { return _cijena; }
    friend ostream& operator<<(ostream& COUT, const Proizvod& p) {
```

```
        COUT       <<       p._naziv       <<       "       "       <<
KategorijaNazivi[(int)p._kategorija] << " " << p._cijena;
        return COUT;
    }
};

class Transakcija {
protected:
    DatumVrijeme _vrijemeRealizacije;
    int _iznos;
public:
    Transakcija(DatumVrijeme vrijemeRealizacije, int iznos = 0)
        : _vrijemeRealizacije(vrijemeRealizacije), _iznos(iznos) {
    }
    virtual ~Transakcija() {}
    virtual string Info() const = 0;
    const   DatumVrijeme&   GetVrijemeRealizacije()   const   {   return
_vrijemeRealizacije; }
    int GetIznos() const { return _iznos; }
};

class Kupovina : public Transakcija {
    vector<Proizvod> _kupljeniProizvodi;
public:
    Kupovina(DatumVrijeme            vrijemeRealizacije)            :
Transakcija(vrijemeRealizacije, 0) {}
    const    vector<Proizvod>&    GetProizvodi()    const    {    return
_kupljeniProizvodi; }
};

class Povrat : public Transakcija {
    vector<Proizvod> _vraceniProizvodi;
public:
    Povrat(DatumVrijeme              vrijemeRealizacije)              :
Transakcija(vrijemeRealizacije, 0) {}
    const    vector<Proizvod>&    GetProizvodi()    const    {    return
_vraceniProizvodi; }
};

class Kupac {
    static int _id;
    char* _sifra;
    char* _imePrezime;
    vector<Transakcija*> _transakcije;
public:
    Kupac(const char* imePrezime) {
        _imePrezime = AlocirajTekst(imePrezime);
        _sifra        =        AlocirajTekst(GenerisiSifru(imePrezime,
_id).c_str());
        _id++;
    }
    ~Kupac() {
        delete[] _sifra;
        delete[] _imePrezime;
        for (auto* transakcija : _transakcije) delete transakcija;
        _transakcije.clear();
    }
    const char* GetSifra() const { return _sifra; }
```

```
    const char* GetImePrezime() const { return _imePrezime; }
    vector<Transakcija*>& GetTransakcije() { return _transakcije; }

    friend ostream& operator<<(ostream& COUT, Kupac& kupac) {
        COUT << crt << kupac._imePrezime << " [" << kupac._sifra <<
"]\n";
        for (int i = 0; i < kupac._transakcije.size(); i++)
            COUT << " - " << kupac._transakcije[i]->Info() << "\n";
        COUT << crt;
        return COUT;
    }
};

class Prodavnica {
    char* _naziv;
    vector<Kupac> _kupci;
public:
    Prodavnica(const char* naziv) { _naziv = AlocirajTekst(naziv); }
    ~Prodavnica() { delete[] _naziv; }
    Prodavnica(const     Prodavnica&     obj)     {     _naziv     =
AlocirajTekst(obj._naziv); _kupci = obj._kupci; }
    const char* GetNaziv() const { return _naziv; }
    vector<Kupac>& GetKupci() { return _kupci; }
};

const char* GetOdgovorNaPrvoPitanje() {
    cout << "Pitanje -> Pojasnite razliku izmedju virtualnih i cistih
virtualnih metoda, te korelaciju virtualnih metoda sa polimorfizmom
(navesti kratki primjer)?\n";
    return "Odgovor -> OVDJE UNESITE VAS ODGOVOR";
}

const char* GetOdgovorNaDrugoPitanje() {
    cout << "Pitanje -> Pojasniti razliku izmedju konstruktora kopije
i move konstruktora, razlike u implementaciji, te navesti primjere
implicitnog i eksplicitnog poziva?\n";
    return "Odgovor -> OVDJE UNESITE VAS ODGOVOR";
}

int main() {

    cout << PORUKA_TELEFON; cin.get(); system("cls");
    cout << PORUKA_ISPIT; cin.get(); system("cls");
    cout << GetOdgovorNaPrvoPitanje() << endl;
    cin.get();
    cout << GetOdgovorNaDrugoPitanje() << endl;
    cin.get();

    //funkcija za generisanje sifre kupca na osnovu imena i prezimena
i rednog broja.
    //sifra je u formatu INICIJALI:TRENUTNA_GODINA-ID_KUPCA, npr.
AB:2025-003.
    //koristiti trenutnu godinu, dobijenu iz sistema na kome se
program izvrsava
    //funkciju koristiti prilikom kreiranja objekta klase Kupac za
inicijalizaciju atributa _sifra
    cout << GenerisiSifru("Amina Buric", 3) << endl; // AB:2025-003
    cout << GenerisiSifru("Amar Macic", 15) << endl; // AM:2025-015
```

```
    cout << GenerisiSifru("Maid Ramic", 156) << endl; // MR:2025-156

    //za validaciju sifre koristiti funkciju ValidirajSifru koja
treba, koristeci regex, osigurati postivanje osnovnih pravila
    //vezanih za format koja su definisana u prethodnom dijelu
zadatka.
    if (ValidirajSifru("AB:2025-003"))
        cout << "SIFRA VALIDNA\n";
    if    (!ValidirajSifru("Ab:2025-003")    &&    !ValidirajSifru("AB-
2025/003") && !ValidirajSifru("AB-003:2025"))
        cout << "SIFRA NIJE VALIDNA\n";

    KolekcijaParova<int, string, 20> listaProizvoda;
    for (int i = 0; i < 10; i++)
        listaProizvoda.Dodaj(i, "Proizvod_" + to_string(i));
    cout << listaProizvoda << crt;

    //DodajNaPoziciju - dodaje par (99, Proizdovi_99) na lokaciju 1
tj. lokaciju definisanu vrijednoscu prvog parametra,
    // a vraca novo stanje kolekcije tj. kolekciju zajedno sa
novododatim elementom
    KolekcijaParova<int,     string,     20>     prosirenaLista     =
listaProizvoda.DodajNaPoziciju(1, 99, "Proizvod_99");
    cout << prosirenaLista << crt;

    // UkloniRaspon - od lokacije definisane prvim parametrom uklanja
broj elemenata definisanih drugi parametrom
    // (pocevsi od lokacije 2 ukloni 3 elementa), a vraca pokazivac na
kolekciju parova s uklonjenim elementima
    KolekcijaParova<int,    string,    20>*    uklonjeniProizvodi    =
prosirenaLista.UkloniRaspon(2, 3);
    cout << "Uklonjeni:\n" << *uklonjeniProizvodi << crt;
    /*
    Uklonjeni:
        1 Proizvod_1
        2 Proizvod_2
        3 Proizvod_3
    */
    cout << "Preostali:\n" << prosirenaLista << crt;
    /*
    Preostali:
        0 Proizvod_0
        99 Proizvod_99
        4 Proizvod_4
        5 Proizvod_5
        6 Proizvod_6
        7 Proizvod_7
        8 Proizvod_8
        9 Proizvod_9
   */
    *uklonjeniProizvodi = prosirenaLista;
    cout << "Proizvodi:\n" << *uklonjeniProizvodi << crt;

    try
    {
        //baciti izuzetak u slucaju nepostojeceg opsega
        listaProizvoda.UkloniRaspon(3, 10); // izuzetak - neispravan
opseg
```

```
    }
    catch (exception& e) {
        cout << "Exception: " << e.what() << crt;
    }

    DatumVrijeme vrijeme1(5, 10, 2025, 9, 30, 0), vrijeme2(5, 10, 2025, 10, 15, 0), vrijeme3(5, 10, 2025, 12, 36, 0);

    Proizvod telefon("Telefon FITPhone", ELEKTRONIKA, 1500), knjiga("Napredno C++ programiranje", KNJIGE, 55),
        slusalice("Slusalice FSX", ELEKTRONIKA, 129), laptop("Laptop FITLx", ELEKTRONIKA, 1499);

    Kupovina kupovina1(vrijeme1), kupovinaDuplikatVremena(vrijeme1), kupovina2(vrijeme2), kupovinaSlusalice(vrijeme2),
        kupovinaDuplikatProizvoda(vrijeme3);

    //dodaje proizvod u listu kupljenih proizvoda i azurira iznos kupovine
    kupovina1.DodajProizvod(telefon);
    kupovina2.DodajProizvod(knjiga);

    //format povratne vrijednosti info metode
    cout << kupovina1.Info() << endl; //05.10.2025 09:30:00 KUPLJENO 1 PROIZVODA U UKUPNOM IZNOSU OD 1500KM

    Kupac amina("Amina Buric"), goran("Goran Skondric"), berun("Berun Agic");

    // DodajTransakciju - oneomguciti dupliranje transakcija sa istim vremenom, kod kupovine onemoguciti
    // dupliranje proizvoda, a povrat omoguciti samo ako je proizvod kupljen.U zavisnosti od rezultata izvrsenja
    // metoda vraca true ili false
    amina.DodajTransakciju(kupovina1);
    amina.DodajTransakciju(kupovina2);

    kupovinaDuplikatProizvoda.DodajProizvod(knjiga);
    //amina je u kupovina2 vec kupila knjigu, duplikat proizvoda, onemoguciti dodavanje
    if (!amina.DodajTransakciju(kupovinaDuplikatProizvoda))
        cout << "Duplikat proizvoda\n";

    kupovinaDuplikatVremena.DodajProizvod(laptop);
    //amina je u kupovina1 vec imala transakciju u vrijeme1, duplikat vremena, onemoguciti dodavanje
    if (!amina.DodajTransakciju(kupovinaDuplikatVremena))
        cout << "Duplikat vremena\n";

    Povrat povratKnjige(vrijeme2);
    povratKnjige.DodajProizvod(knjiga);

    //format povratne vrijednosti Info metode
    cout << povratKnjige.Info() << endl; // 05.10.2025 10:15:00 VRACENO 1 PROIZVODA U UKUPNOM IZNOSU OD 55KM

    //povrat dozvoljen samo ako je proizvod ranije kupljen
    if (amina.DodajTransakciju(povratKnjige))
```

```
        cout << "Povrat uspjesno izvrsen\n";

    Prodavnica tehnika("Tehnika"), knjizara("Knjizara");
    tehnika.DodajKupca(amina);
    tehnika.DodajKupca(goran);
    knjizara.DodajKupca(berun);

    try {
        tehnika.DodajKupca(amina); // amina je vec dodata kao kupac
    }
    catch (exception& e) {
        cout << "Exception: " << e.what() << crt;
    }

    kupovinaSlusalice.DodajProizvod(slusalice);
    //registraciju transakcije, pored direktnog nacina - preko kupca,
    //je moguce realizovati i u okviru odredjene prodavnice koja ce
pronaci kupca
    //po sifri i dodati mu transakciju proslijedjenu kao parametar.u
zavisnosti od rezultata
    //izvrsenja metoda vraca true ili false (i dalje vaze pravila
vezana za dodavanje transakcije)
    if           (!tehnika.RegistrujTransakcijuKupcu(amina.GetSifra(),
kupovinaSlusalice))
        cout << "Transakcija registrovana\n";

    vector<Prodavnica> prodavnice;
    prodavnice.push_back(tehnika);
    prodavnice.push_back(knjizara);

    /*
    Funkcija UcitajPodatke ima zadatak ucitati podatke o prodavnicama
i njihovim kupcima iz fajla cije ime se proslijedjuje kao parametar
    (fajl mozete pronaci zajedno sa ispitnim zadatkom). Svaka linija u
fajlu treba biti u formatu "ime i prezime kupca|naziv prodavnice".
Funkcija za
    svaki red u fajlu:
        - unutar vector-a, po nazivu, pronadje ranije dodatu ili
kreira novu prodavnicu,
        - kreira novog kupca ukoliko vec nije registrovan u naznacenoj
prodavnici,
        - dodaje kupca u naznacenu prodavnicu (onemoguciti
dupliciranje kupaca u istoj prodavnici).
    Na kraju, sve prodavnice sa svojim kupcima se trebaju nalaziti u
proslijedjenom vektoru prodavnice.
    Funkcija vraca true ako je ucitavanje podataka bilo uspjesno (u
vector ucitan najmanje jedan podatak
    o prodavnici ili kupcu), a false ako se desilo nesto neocekivano
ili niti jedan podatak nije ucitan.

    Primjer sadrzaja fajla:

        Emina Junuz|Tehnika";
        Jasmin Azemovic|Tehnika";
        Zanin Vejzovic|Knjizara";
    */
    string nazivFajla = "podaci.txt";
```

```
    if (UcitajPodatke(nazivFajla, prodavnice)) cout << "Ucitavanje
uspjesno" << crt;
    for (auto& prodavnica : prodavnice)
        cout << prodavnica.GetNaziv() << " sa " <<
prodavnica.GetKupci().size() << " kupaca" << crt;

    //vraca listu svih kupaca iz prodavnice koji su imali najmanje
jednu transakciju u proslijedjenoj kategoriji,
    //te koliko su ukupno potrosili na proizvode iz te kategorije
(ukupna potrosnja za kategoriju se
    // izracunava: kupovina - povrat)
    KolekcijaParova<Kupac, int, 50> potrosnja =
tehnika.PotrosnjaPoKategoriji(ELEKTRONIKA);

    for (int i = 0; i < potrosnja.GetTrenutno(); i++)
        cout << potrosnja.GetPrvi(i).GetImePrezime() << " [" <<
potrosnja.GetPrvi(i).GetSifra() << "] "
        << " potrosio/la " << potrosnja.GetDrugi(i) << " KM na
proizvode iz kategorije ELEKTRONIKA" << crt;

    cin.get();
    return 0;
}
```