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;

const char* PORUKA = "\n-----------------------------------------------
---------------------------------\n"
"0. PROVJERITE DA LI PREUZETI ZADACI PRIPADAJU VASOJ GRUPI (G1/G2)\n"
"1. SVE KLASE TREBAJU POSJEDOVATI ADEKVATAN DESTRUKTOR\n"
"2.  NAMJERNO  IZOSTAVLJANJE  KOMPLETNIH  I/ILI  POJEDINIH  DIJELOVA
DESTRUKTORA CE BITI OZNACENO KAO TM\n"
"3.  SPASAVAJTE  PROJEKAT  KAKO  BI  SE  SPRIJECILO  GUBLJENJE  URADJENOG
ZADATKA\n"
"4. ATRIBUTI, NAZIVI METODA (SVE ISTO VAZI I ZA FUNKCIJE), TE BROJ I
TIP PARAMETARA MORAJU BITI IDENTICNI "
"ONIMA KOJI SU KORISTENI U TESTNOM CODE - U, OSIM U SLUCAJU DA POSTOJI
ADEKVATAN RAZLOG ZA NJIHOVU MODIFIKACIJU. "
"OSTALE POMOCNE METODE MOZETE IMENOVATI I DODAVATI PO ZELJI.\n"
"5. IZUZETAK BACITE SAMO U METODAMA U KOJIMA JE TO NAZNACENO.\n"
"6.  SVE  METODE  POZVANE  U  MAIN-U  ZADATKA  TREBAJU  POSTOJATI.  UKOLIKO
NISTE ZADOVOLJNI IMPLEMENTACIJOM "
"POTREBNO JE DA IMPLEMENTIRATE BAREM TIJELO TIH METODA (METODA MOZE
BITI PRAZNA),"
"A AKO METODA TREBA VRATITI NEKI PODATAK ONDA MOZETE VRATITI BILO KOJU
TJ. ZELJENU VRIJEDNOST ZAHTIJEVANOG TIPA.!\n"
"7.  NA  KRAJU  ISPITA  SVOJE  RJESENJE  KOPIRAJTE  U  .DOCX  FAJL  (IMENOVAN
BROJEM INDEKSA npr. IB150051.docx)!\n"
"8. RJESENJA ZADATKA POSTAVITE NA FTP SERVER U ODGOVARAJUCI FOLDER!\n"
"9. NEMOJTE POSTAVLJATI VISUAL STUDIO PROJEKTE, VEC SAMO .DOCX FAJL SA
VASIM RJESENJEM!\n"
"10.SVE NEDOZVOLJENE RADNJE TOKOM ISPITA CE BITI SANKCIONISANE!\n"
"11.ZA  POTREBE  TESTIRANJA,  U  MAIN-U,  BUDITE  SLOBODNI  DODATI  TESTNIH
PODATAKA (POZIVA METODA) KOLIKO GOD SMATRATE DA JE POTREBNO!\n"
"12.ZA  IZRADU  ISPITNOG  RJESENJA  KORISTITI  VISUAL  STUDIO  2022  I
RJESENJE TESTIRAJTE U OBA MODA (F5 i Ctrl+F5)!\n"
"13.NA KRAJU ISPITA PROVJERITE DA LI STE RJEŠENJE KOPIRALI U ADEKVATAN
FOLDER NA FTP SERVERU\n"
"-----------------------------------------------------------------------
----------\n";

enum  Specializacija  {  KARDIOLOGIJA,  ORTOPEDIJA,  DERMATOLOGIJA,
PEDIJATRIJA, OPSTA_MEDICINA };
const char* SpecijalizacijaNazivi[] = {
    "KARDIOLOGIJA",  "ORTOPEDIJA",  "DERMATOLOGIJA",  "PEDIJATRIJA",
"OPSTA MEDICINA"
};
const char* crt = "\n-----------------------------------------\n";

char* AlocirajNiz(const char* sadrzaj, bool dealociraj = false) {
    if (sadrzaj == nullptr)return nullptr;
    int vel = strlen(sadrzaj) + 1;
    char* temp = new char[vel];
```

```
    strcpy_s(temp, vel, sadrzaj);
    if (dealociraj)
        delete[]sadrzaj;
    return temp;
}

template<class T1, class T2, int max>
class KolekcijaG1 {
    T1* _elementi1[max];
    T2* _elementi2[max];
    int _trenutno;
public:
    T1& getElement1(int lokacija) { return *_elementi1[lokacija]; }
    T2& getElement2(int lokacija) { return *_elementi2[lokacija]; }
    int getTrenutno() const { return _trenutno; }
    friend ostream& operator<< (ostream& COUT, KolekcijaG1& obj) {
        for (size_t i = 0; i < obj._trenutno; i++)
            COUT << obj.getElement1(i) << " " << obj.getElement2(i) <<
endl;
        return COUT;
    }
};
class Termin {
    int* _sati; int* _minute; int* _sekunde;
public:
    Termin(int s = 0, int m = 0, int k = 0) {
        _sati = new int(s);
        _minute = new int(m);
        _sekunde = new int(k);
    }
    ~Termin() {
        delete _sati; delete _minute; delete _sekunde;
    }
    int getSati() const { return *_sati; }
    int getMinute() const { return *_minute; }
    int getSekunde() const { return *_sekunde; }
    friend ostream& operator<<(ostream& COUT, const Termin& t) {
        COUT << *t._sati << ":" << *t._minute << ":" << *t._sekunde;
        return COUT;
    }
};

class Dogadjaj {
protected:
    Termin _termin;
public:
    Dogadjaj(Termin termin) : _termin(termin) {}
    virtual ~Dogadjaj() {}
    virtual string Info() const = 0;
    const Termin& getTermin() const { return _termin; }
};

class Predavanje{
    char* _tema;
    Specializacija _specijalizacija;
public:
    ~Predavanje() {
        delete[] _tema;
```

```
    }
    const char* GetTema() const { return _tema; }
    Specializacija     GetSpecijalizacija()     const     {     return
_specijalizacija; }
};

class Ucesnik {
     static int _id; //iskoristiti za praćenje rednog broja ucesnika
i generisanje jedinstvene sifre
     char* _sifra; //sifra u formatu GG-IN-BBB, pojasnjena u main
funkciji, generisati prilikom kreiranja objekta
    char* _imePrezime;
    vector<Dogadjaj*> _prijavljeni;
public:
    Ucesnik(const char* imePrezime) {
        _imePrezime = AlocirajNiz(imePrezime);
    }
    ~Ucesnik() {
        delete[] _imePrezime;
        delete[] _sifra;
        for (auto prijava : _prijavljeni)
            delete prijava;
        _prijavljeni.clear();
    }
    const char* getSifra() const { return _sifra; }
    const char* getImePrezime() const { return _imePrezime; }
    vector<Dogadjaj*>& getDogadjaji() { return _prijavljeni; }
};

class Tim {
    char* _naziv;
    vector<Ucesnik> _clanovi;
public:
    Tim(const char* naziv) {
        _naziv = AlocirajNiz(naziv);
    }
    ~Tim() {
        delete[] _naziv;
    }
};

/*
Klasa Konferencija omogucava organizaciju i pracenje koji timovi i
ucesnici prisustvuju kojim predavanjima,
ali sama ne sadrzi direktno predavanja, vec ih povezuje preko ucesnika
i timova.
*/
class Konferencija {
    char* _naziv;
    KolekcijaG1<Tim*, Tim*, 20> _timovi;
public:
    Konferencija(const char* naziv)
    {
        _naziv = AlocirajNiz(naziv);
    }
    ~Konferencija() {
        delete[] _naziv; _naziv = nullptr;
    }
```

```
    char* getNaziv() const { return _naziv; }
    KolekcijaG1<Tim*, Tim*, 20>& getTimovi() { return _timovi; }
};

const char* GetOdgovorNaPrvoPitanje() {
    cout << "Pitanje -> Pojasnite razliku između 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;
    cin.get();
    cout << GetOdgovorNaPrvoPitanje() << endl;
    cin.get();
    cout << GetOdgovorNaDrugoPitanje() << endl;
    cin.get();

    /* sifra korisnika treba biti u formatu GG-IN-BBB pri cemu su:

       GG - posljednje dvije cifre trenutne godine (npr.za 2025 ->
25), preuzeti vrijednost iz sistema
       IN - inicijali ucesnika, velika slova(prvo slovo imena i
prezimena)
       BBB → trocifreni redni broj ucesnika kreiran na osnovu _id-a
(npr. 001, 023, 105)

       validnom sifrom treba smatrati i onu koja umjesto znaka crtica
'-' ima znak razmak npr: 25 DM 003 ili 25 DM-003
   */
    cout << GenerisiSifru("Denis Music", 3) << endl;//treba vratiti
25-DM-003
    cout << GenerisiSifru("Jasmin Azemovic", 14) << endl;//treba
vratiti 25-JA-014
    cout << GenerisiSifru("Goran skondric", 156) << endl;//treba
vratiti 25-GS-156
    cout << GenerisiSifru("emina junuz", 798) << endl;//treba vratiti
25-EJ-798

    //Za validaciju sifre koristiti funkciju ValidirajSifru koja
treba, koristeci regex, osigurati postivanje osnovnih pravila
    //vezanih za format koja su definisana u prethodnom dijelu
zadatka. Pored navedenih,
    if (ValidirajSifru("25-DM-003"))
        cout << "SIFRA VALIDNA" << endl;
    if (ValidirajSifru("25-JA-014") && ValidirajSifru("25-JA 014"))
        cout << "SIFRA VALIDNA" << endl;
    if (!ValidirajSifru("25-GS-15") || !ValidirajSifru("25-Gs-135") ||
!ValidirajSifru("25-GS-153G"))
```

```
        cout << "SIFRA NIJE VALIDNA" << endl;

    int kolekcijaTestSize = 9;
    KolekcijaG1<int, string, 10> kolekcija1;
    for (int i = 0; i < kolekcijaTestSize; i++)
        kolekcija1.AddElement(i, "Vrijednost -> " + to_string(i));
    cout << kolekcija1 << crt;

    /* metoda InsertAt treba da doda vrijednosti drugog i treceg
parametra na lokaciju koja je definisana prvim parametrom. Povratna
vrijednost metode
     je objekat (pozivaoc metode, u konkretnom slucaju objekat
kolekcija1) u okviru koga su, na definisanu lokaciju, dodati
proslijedjeni parametri.
     Nakon izvrsenja metode InsertAt, oba objekta, parovi1 i parovi2,
bi trebali posjedovati sljedeci sadrzaj:
     0  Vrijednost -> 0
     10 Vrijednost -> 10
     1  Vrijednost -> 1
     2  Vrijednost -> 2
    * ....
    */
    KolekcijaG1<int, string, 10> kolekcija2 = kolekcija1.InsertAt(1,
10, "Vrijednost -> 10");
    cout << kolekcija2 << crt;

    /*Metoda RemoveRange kao prvi parametar prihvata pocetnu lokaciju
a kao drugi parametar broj elemenata koje, od pocetne lokacije uklanja
iz kolekcije koja je pozvala
    tu metodu. U slucaju da zahtijevani broj elemenata ne postoji u
kolekciji  metoda treba baciti izuzetak.
    Na kraju, metoda treba da vrati pokazivac na novi objekat tipa
Kolekcija koji sadrzi samo uklonjene elemente*/

    KolekcijaG1<int,       string,       10>*       kolekcija3      =
kolekcija1.RemoveRange(1, 3);//uklanja 3 elementa pocevsi od lokacije
1

    cout << "Uklonjeni:\n" << *kolekcija3;
    cout << "Preostali:\n" << kolekcija1;

    try {
        kolekcija3->RemoveRange(2,   3);   //pokusavaju   se   ukloniti
nepostojeći elementi
    }
    catch (exception& e) {
        cout << "Exception: " << e.what() << crt;
    }

    delete kolekcija3;
    kolekcija1 = kolekcija2;
    cout << kolekcija1 << crt;


    Termin termin1(19, 02, 30), termin2(10, 30, 40), termin3(14, 15,
20), termin4(16, 45, 20);
```

```
    Predavanje oboljenja_srca(termin1, "Oboljenja srca",
KARDIOLOGIJA);
    Predavanje uv_zracenja(termin2, "Uloga UV zracenja u koznim
oboljenjima", DERMATOLOGIJA);
    Predavanje anemije(termin3, "Anemije u svakodnevnoj praksi",
OPSTA_MEDICINA);

    Ucesnik emina("Emina Junuz"), goran("Goran Skondric"), azra("Azra
Maric"), tajib("Tajib Hero");
     //metoda PrijaviSe dodaje prijavu na predavanje ucesniku,
ukoliko je prijava uspjesna, vraca true, a u suprotnom false.
     //onemoguciti dupliranje prijava na isto predavanje
    emina.PrijaviSe(oboljenja_srca);
    emina.PrijaviSe(anemije);
    goran.PrijaviSe(oboljenja_srca);
    goran.PrijaviSe(uv_zracenja);
    tajib.PrijaviSe(uv_zracenja);

    //info metoda vraca sve detalje o dogadjaju u string formatu npr.:
19:02:30 Oboljenja srca KARDIOLOGIJA
    if (!emina.PrijaviSe(oboljenja_srca)) {
        cout << "Ucesnik " << emina.getImePrezime() << " vec
prijavljen na predavanje " << oboljenja_srca.Info() << crt;
    }

    Tim timAlpha("Tim Alpha"), timBeta("Tim Beta");
     //metoda DodajUcesnika treba da doda ucesnika u tim, ukoliko
ucesnik vec nije clan tima, u suprotnom treba baciti izuzetak.
    timAlpha.DodajUcesnika(emina); timAlpha.DodajUcesnika(goran);
    timBeta.DodajUcesnika(azra); timBeta.DodajUcesnika(tajib);

    try {
        timAlpha.DodajUcesnika(emina);//emina je vec clanica tima
Alpha
    }
    catch (exception& e) {
        cout << "Exception: " << e.what() << "\n";
    }

    Konferencija savremena_medicina("Umjetna inteligencija u
dijagnostici i lijecenju – novo lice medicine");
    savremena_medicina.DodajTimove(timAlpha, timBeta);

     //ispisuje naziv konferencije, nazive timova i podatke o svim
clanovima tima
    cout << savremena_medicina;

    //metoda PrijaviDogadjaj treba omoguciti prijavu
dogadjaja/predavanja ucesniku/clanu proslijedjenog tima. na osnovu
poruka
    //koje se ispisuju u nastavku, implementirati metodu
PrijaviDogadjaj tako da se prijave vrse samo na osnovu ispravnih
podataka.
    if (savremena_medicina.PrijaviDogadjaj(timAlpha.getNaziv(),
emina.getSifra(), uv_zracenja))
        cout << "Prijava uspjesna" << crt;
```

```
    if        (!savremena_medicina.PrijaviDogadjaj("Tim        Gamma",
emina.getSifra(), anemije))
        cout << "Pokusaj prijave dogadjaja za nepostojeci tim" << crt;

    if       (!savremena_medicina.PrijaviDogadjaj(timAlpha.getNaziv(),
emina.getSifra(), uv_zracenja))
        cout << "Pokusaj dupliranja prijave predavanja" << crt;

    if       (!savremena_medicina.PrijaviDogadjaj(timAlpha.getNaziv(),
azra.getSifra(), uv_zracenja))
        cout << "Ucesnik nije clan proslijedjenog tima" << crt;

    if  (!savremena_medicina.PrijaviDogadjaj(timAlpha.getNaziv(),  "24-
GX-002", anemije))
        cout << "Prijava sa nepostojecom sifrom nije uspjela.";


    //metoda vraca sve ucesnike koji su se na odredjenoj konferenciji
prijavili na minimalno proslijedjeni broj predavanja
    vector<Ucesnik*> vrijedniUcesnici = savremena_medicina(2);
    for (auto ucesnik : vrijedniUcesnici)
        cout << ucesnik->getImePrezime() << "\n";

    /*
    Funkcija UcitajUcesnike ima zadatak ucitati podatke o ucesnicima i
njihovim timovima 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|naziv tima". Funkcija za
    svaki red u fajlu:
        - unutar vector-a, po nazivu, pronadje ranije dodati ili
kreira novi tim,
        - kreira novog ucesnika ukoliko vec nije dio tog tima,
        - dodaje ucesnika u odgovarajuci tim (onemoguciti dupliciranje
korisnika u istom timu).
    Na kraju, svi timovi sa svojim clanovima se nalaze u
proslijedjenom vektoru timovi.
    Funkcija vraca true ako je ucitavanje podataka bilo ouspjesno, a
false ako se desilo nesto neocekivano.

    Primjer sadrzaja fajla:
    Goran Skondric|Tim Alpha
    Emina Junuz|Tim Alpha
    Azra Maric|Tim Beta
    Tajib Hero|Tim Beta
    */

    vector<Tim> timoviIzFajla;
    if (UcitajUcesnike("ucesnici.txt", timoviIzFajla))
        cout << "Ucitavanje podataka USPJESNO.\n";
    else
        cout << "Ucitavanje podataka NEUSPJESNO.\n";

    for (auto& tim : timoviIzFajla)
        cout << tim << crt;

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