Решени задачи: STL

Предавање 1

1.      Напиши function template за сортирање и програма во која се врши внесување, сортирање и печатење на поле од int елементи и поле од double елементи. Може да се користи било кој алгоритам за сортирање.

template<class T>

vector<T> bubbleSort(vector<T>& vector) {

for (int i = 0; i < vector.size() - 1; i++) {

for (int j = 0; j < vector.size() - i - 1; j++) {

if (vector[j] > vector[j + 1])

swap(vector[j], vector[j + 1]);

}

}

return vector;

}

int main() {

vector<int> intVector;

intVector.push\_back(5);

intVector.push\_back(8);

intVector.push\_back(9);

intVector.push\_back(54);

intVector.push\_back(4);

intVector.push\_back(3);

vector<double> doubleVector;

doubleVector.push\_back(6.43);

doubleVector.push\_back(77.56);

doubleVector.push\_back(2.3);

doubleVector.push\_back(56.4);

doubleVector.push\_back(3.78);

intVector = bubbleSort(intVector);

for(const auto& vector : intVector) {

cout<<vector<<", ";

}

cout<<endl;

doubleVector = bubbleSort(doubleVector);

for(const auto& vector : doubleVector) {

cout<<vector<<", ";

}

return 0;

}

2.      Преклопи го **printArray** така што ќе добие уште два целобројни аргументи **lowSubscript** и **highSubscript**. Повик на оваа функција треба да ги отпечати елементите помеѓу **lowSubscript** и **highSubscript**. Ако нивните вредности излегуваат од димензиите на полето, **printArray** треба да врати 0, во спротивно бројот на отпечатени елементи. Во main да се тестира функционирањето на двете верзии на **printArray** врз полињата a, b и c.

template< class T > //\*

void printArray( const T \*array, const int count, T lowSubscript, T highSubscript)//\*\*

{

int br = 0;

for ( int i = 0; i < count; i++ ) {

if(array[i] >= lowSubscript && array[i] <= highSubscript) {

cout << array[ i ] << " "<<endl;

br++;

}

}

if(br == count)

cout<<"0"<<endl;

}

3.      Напиши програма која содржи class template **Array** со која може да се инстанцира поле од било кој тип. Класата треба да содржи функции членки за сместување и преземање на вредност на елемент од полето. Да се демонстрира создавање на објекти за **int** и **double** тип.

template<class T>

class Array {

T\* data;

int size;

public:

explicit Array(int size) : size(size) {

data = new T[size];

}

~Array() {

delete[] data;

}

void setElement(int index, const T& value) {

if (index <= size) {

data[index] = value;

} else {

cout << "Index out of bounds." << endl;

}

}

T getElement(int index) const {

if (index < size) {

return data[index];

} else {

cout << "Index out of bounds." << endl;

return T();

}

}

};

int main() {

Array<int> int\_array(5);

int\_array.setElement(0,5);

int\_array.setElement(1,3);

int\_array.setElement(2,7);

int\_array.setElement(3,1);

int\_array.setElement(4,2);

cout << "Integer Array: ";

for (size\_t i = 0; i < 5; ++i) {

cout << int\_array.getElement(i) << " ";

}

cout << endl;

Array<double> doubleArray(3);

doubleArray.setElement(0,1.1);

doubleArray.setElement(1,2.2);

doubleArray.setElement(2,3.3);

cout << "Double Array: ";

for (size\_t i = 0; i < 3; ++i) {

cout << doubleArray.getElement(i) << " ";

}

cout << endl;

return 0;

}

Предавање 2

1. Напиши function template кој има еден параметризиран тип **(T)** и прифаќа четири аргументи: поле од тип **T**, почетен индекс, краен индекс и опциона, почетна вредност. Функцијата трба да ја врати сумата на елементите во полето зададени со почетниот и крајниот индекс. За почетната вредност да се употреби конструкторот на **T**.

template<typename T>

T sum(const vector<T>& arr, size\_t start, size\_t end, T initial\_value = T()) {

T total = initial\_value;

if(start > end || start >= arr.size() || end >= arr.size()) {

cout<<"Invalid indices"<<endl;

return total;

}

for (size\_t i = start; i <= end ; ++i) {

total += arr[i];

}

return total;

}

int main() {

vector<int> arr;

arr.push\_back(1);

arr.push\_back(2);

arr.push\_back(3);

arr.push\_back(4);

arr.push\_back(5);

int result = sum(arr,1,4);

cout<<result<<endl;

return 0;

}

**2.**Креирај класа **Kvadrat** со податоци **strana** и класа **Pravoagolnik** со податоци **sirina** и **dolzina**.  Напиши function template **sortirajPole** со кој се врши сортирање на поле и function template **pecatiPole** со кој се врши печатење на поле. Во **main** креирај две полиња од квадрати и праваголници, прикажи ги со **pecatiPole**, сортирај ги според плоштина со **sortirajPole** и повторно отпечати ги.

class Kvadrat {

public:

double strana;

Kvadrat(double s) : strana(s) {}

double plostina() const {

return strana \* strana;

}

};

class Pravoagolnik {

public:

double sirina;

double dolzina;

Pravoagolnik(double s, double d) : sirina(s), dolzina(d) {}

double plostina() const {

return sirina \* dolzina;

}

};

template<typename T>

void pecatiPole(const vector<T>& arr) {

for(const auto& element : arr)

cout<<element.plostina()<<", "<<endl;

}

template<typename T>

void sortirajPole(vector<T>& arr) {

bool swapped;

size\_t n = arr.size();

do {

swapped = false;

for (size\_t i = 0; i < n - 1; ++i) {

if (arr[i].plostina() > arr[i + 1].plostina()) {

swap(arr[i], arr[i + 1]);

swapped = true;

}

}

--n;

} while (swapped);

}

**3.**Една фирма води евиденција за лица кои можат да бидат вработени или клиенти. За секое лице се чуваат следните податоци:  **imeiprezime**, **mat\_br**, **vozrast**, **adresa**, **kod**. За вработените **kod** има целобројна вредност, а за клиентите е низа од знаци.

Креирај class template **Lica** во која ќе се сместуваат податоците за лица. Податоците кои се низи од знаци да се внесуваат преку динамичка алокација на меморија. Креирај две полиња од објекти, **Vraboteni** и **Klienti**. За печатење на двете полиња употреби ја **pecatiPole** од претходната задача.

template<typename T>

class Lica {

public:

string imeiprezime;

string mat\_broj;

int vozrast;

string adresa;

T kod;

Lica(string ip, string m\_b, int v, string a, T k)

: imeiprezime(ip), mat\_broj(m\_b), vozrast(v), adresa(a), kod(k) {}

void pecati() const {

cout << "Ime i prezime: " << imeiprezime << endl;

cout << "Maticen broj: " << mat\_broj << endl;

cout << "Vozrast: " << vozrast << endl;

cout << "Adresa: " << adresa << endl;

cout << "Kod: " << kod << endl;

}

};

template<typename T>

void pecatiPole(const vector<Lica<T> >& arr) {

for(const auto& element : arr)

element.pecati();

}

int main() {

vector<Lica<int> > Vraboteni;

vector<Lica<string> > Klienti;

Vraboteni.push\_back(Lica<int>("Petko","1231231231231",52,"Adresa 123",1324));

Klienti.push\_back(Lica<string>("Zoro","5345435435432",37,"Adresa 332","Kod123"));

pecatiPole(Vraboteni);

pecatiPole(Klienti);

return 0;

}

Предавање 3

**1.**Напиши class template **Zapis<T>** кој содржи три приватни членки: **x**, **y** и **z** од тип Т. Креирај function template **kontrolor** кој ќе проверува дали збирот на елементите**x, y** и **z** во **Zapis** надминува 10000. Напиши соодветна програма за демонстрација со инстанцирање на објекти од тип **Zapis<int>** и **Zapis<double>**.

template<typename T>

class Zapis {

private:

T x,y,z;

public:

Zapis() {}

Zapis(T x, T y, T z) : x(x), y(y), z(z) {

}

void kontrolor() {

if(x + y + z > 10000)

cout<<"Nadminuva"<<endl;

else

cout<<"Ne nadminuva"<<endl;

}

};

int main() {

Zapis<int> intZapis = Zapis<int>(55,432,503);

Zapis<double> doubleZapis = Zapis<double>(456.32,66.54,500.3);

intZapis.kontrolor();

doubleZapis.kontrolor();

return 0;

}

Предавање 4

**1.**Напиши класа **Imenik** којa содржи **ime** и **tel\_broj**. Креирај телефонски именик користејќи контејнер **vector**.   Прикажи го именикот во абецеден редослед.

class Imenik {

public:

string ime;

string broj;

Imenik(string ime, string broj): ime(ime), broj(broj) {}

friend ostream& operator<<(ostream& os, const Imenik& licnost) {

os<<licnost.ime<<", "<<licnost.broj<<endl;

return os;

}

bool operator<(const Imenik& licnost) const {

return ime[0] < licnost.ime[0];

}

};

int main() {

vector<Imenik\*> imenik;

Imenik ime1 = Imenik("Filip", "077767767");

Imenik ime2 = Imenik("Vladko", "078867453");

Imenik ime3 = Imenik("Goran", "072343563");

imenik.push\_back(&ime1);

imenik.push\_back(&ime3);

imenik.push\_back(&ime2);

sort(imenik.begin(), imenik.end());

for(const auto& element: imenik) {

cout<< \*element;

}

return 0;

}

**2.** За текст внесен од тастатура прикажи ги сите зборови од кои се состои текстот по еднаш, во абецеден редослед.

string text;

getline(cin, text);

vector<string> words;

stringstream ss(text);

string word;

while(ss >> word) {

words.push\_back(word);

}

sort(words.begin(), words.end());

auto it = unique(words.begin(), words.end());

words.erase(it, words.end());

for(const auto& word : words) {

cout<<word<<endl;

}

Предавање 5

**1.**Напиши function template **palindrom** кој има параметар **vector** и враќа **true** ако векторот е палиндром, а **false** во спротивно.

**template<typename T>**

**bool palindrom(const vector<T>& vec) {**

**int count = 1;**

**for(auto it = vec.begin(); it != vec.begin() + vec.size() / 2 ; it++) {**

**if(\*it != \*(vec.end() - count))**

**return false;**

**count++;**

**}**

**return true;**

**}**

**int main() {**

**vector<int> intVector;**

**intVector.push\_back(1);**

**intVector.push\_back(2);**

**intVector.push\_back(3);**

**intVector.push\_back(2);**

**intVector.push\_back(1);**

**vector<double> doubleVector;**

**doubleVector.push\_back(1.2);**

**doubleVector.push\_back(2.2);**

**doubleVector.push\_back(3.2);**

**doubleVector.push\_back(4.2);**

**doubleVector.push\_back(5.2);**

**vector<char> charVector;**

**charVector.push\_back('a');**

**charVector.push\_back('b');**

**charVector.push\_back('c');**

**charVector.push\_back('c');**

**charVector.push\_back('b');**

**charVector.push\_back('a');**

**if(palindrom(charVector))**

**cout<<"Palindrom"<<endl;**

**else**

**cout<<"Ne e palindrom"<<endl;**

**return 0;**

**2.      Дадени се три листи од e-mail адреси. Некои адреси се појавуваат само во една од листите, а некои се повторуваат и во другите листи. Напиши програма со која ќе се состави една листа во која ќе се содржат сите адреси по еднаш.**

**void insertUnique(list<string>& lst, const string& email) {**

**for (const auto& existingEmail : lst) {**

**if (existingEmail == email)**

**return;**

**}**

**lst.push\_back(email);**

**}**

**int main() {**

**list<string> emails1;**

**list<string> emails2;**

**list<string> emails3;**

**emails1.push\_back("email1@gmail.com");**

**emails1.push\_back("email2@gmail.com");**

**emails1.push\_back("email3@gmail.com");**

**emails1.push\_back("email4@gmail.com");**

**emails2.push\_back("email5@gmail.com");**

**emails2.push\_back("email6@gmail.com");**

**emails2.push\_back("email7@gmail.com");**

**emails2.push\_back("email4@gmail.com");**

**emails3.push\_back("email8@gmail.com");**

**emails3.push\_back("email1@gmail.com");**

**emails3.push\_back("email6@gmail.com");**

**list<string> emails;**

**for (const auto& email : emails1) {**

**insertUnique(emails, email);**

**}**

**for (const auto& email : emails2) {**

**insertUnique(emails, email);**

**}**

**for (const auto& email : emails3) {**

**insertUnique(emails, email);**

**}**

1. **Формирај deque d од цели броеви. Потоа формирај vector v од елементите на d, но во обратен редослед.**

**deque<int> d;**

**d.push\_front(5);**

**d.push\_front(4);**

**d.push\_front(3);**

**d.push\_front(2);**

**d.push\_front(1);**

**vector<int> v;**

**for(auto it = d.rbegin(); it != d.rend(); ++it) {**

**v.push\_back(\*it);**

**}**

Предавање 6

1. Во еден систем пристигнуваат процеси кои треба да бидат опслужени. Секој процес има идентификационен број и приоритет. Напиши прграма која за внесена листа на процеси и приоритети ќе прикаже по кој редослед тие ќе бидат опслужени.

map<int,list<Proces> > procesi;

Proces proces1 = Proces(456,1);

Proces proces2 = Proces(5334,4);

Proces proces3 = Proces(634,1);

Proces proces4 = Proces(123,7);

Proces proces5 = Proces(7756,3);

procesi[proces1.prioritet].push\_back(proces1);

procesi[proces2.prioritet].push\_back(proces2);

procesi[proces3.prioritet].push\_back(proces3);

procesi[proces4.prioritet].push\_back(proces4);

procesi[proces5.prioritet].push\_back(proces5);

for(const auto& proces : procesi) {

cout<<"Prioritet: "<<proces.first << endl;

for(const auto& lista : proces.second) {

cout<<"Ime na proces: "<<lista.broj<<endl;

}

}

Предавање 7

1.      Креирај листа од 10000 случајни цели броеви co вредности од 0 до 9999. Потоа прикажи ги следните вредности:

-         број на парни елементи

-         број на елементи помали од 2000

-         број на елементи во интервал [2000, 4000)

-         број на елементи во интервал [4000, 6000)

-         број на елементи во интервал [6000, 8000)

-         број на елементи во интервал [8000, 10000)

-         средна вредност на елементите

vector<int> numbers(10000);

generate(numbers.begin(), numbers.end(), []() {

return rand() % 9999;

});

int evenCount = count\_if(numbers.begin(), numbers.end(), [](int num) { return num % 2 == 0; });

int lessThan2000 = count\_if(numbers.begin(), numbers.end(), [](int num) { return num < 2000; });

int between2000and4000 = count\_if(numbers.begin(), numbers.end(), [](int num) { return num >= 2000 && num < 4000; });

int between4000and6000 = count\_if(numbers.begin(), numbers.end(), [](int num) { return num >= 4000 && num < 6000; });

int between6000and8000 = count\_if(numbers.begin(), numbers.end(), [](int num) { return num >= 6000 && num < 8000; });

int between8000and10000 = count\_if(numbers.begin(), numbers.end(), [](int num) { return num >= 8000 && num < 10000; });

double average = static\_cast<double>(accumulate(numbers.begin(), numbers.end(), 0)) / numbers.size();

Предавање 8

1.       Во една фирма платата на вработените се пресметува како производ од број на бодови и вредност на бод. Вредноста на бодот е иста за сите вработени.Имплементирај класа Vraboten која содржи податочни членови ime и bodovi. Напиши програма со која ќе се пресмета вкупната сума за да се исплати месечна плата за целата фирма. Програмата треба да ја прикаже вкупната сума и список на вработените сортиран според бројот на бодови. За пресметување на сумата да се употреби accumulate. За сортирање на листата да се употреби sort.

class Vraboten {

public:

string ime;

int bodovi;

static int vrednost\_na\_bod;

Vraboten(string ime, int bodovi) : ime(ime), bodovi(bodovi) {}

int vratiBodovi() {

return bodovi \* vrednost\_na\_bod;

}

void print() {

cout<<"Ime: "<<ime<<", Bodovi: "<<bodovi<<endl;

}

bool operator<(const Vraboten& other) const {

return bodovi < other.bodovi;

}

};

int Vraboten::vrednost\_na\_bod = 15;

int main() {

list<Vraboten> firma;

firma.push\_back(Vraboten("Mirce",250));

firma.push\_back(Vraboten("Svirce",540));

firma.push\_back(Vraboten("Kirce",350));

firma.push\_back(Vraboten("Pirce",450));

int totalSalary = accumulate(firma.begin(), firma.end(), 0, [](int sum, const Vraboten& v) {

return sum + v.vratiPlata();

});

firma.sort();

for(auto& covek : firma) {

covek.print();

}

return 0;

}

Предавање 9

**1.**Креирај два вектори со случаен број на елементи кој е во интервалот [1, 1000). Потоа ископирај го вториот вектор во првиот со copy. Претходно провери дали првиот вектор има големина поголема или еднаква на големината на првиот. Ако не е, да се изврши фрлање на исклучок пред да се изврши копирањето на векторот. Постапката да се повторува во циклус во кој е сместена **try-catch** секвенца.

srand(static\_cast<unsigned>(time(0)));

size\_t size1 = rand() % 1000 + 1;

size\_t size2 = rand() % 1000 + 1;

vector<int> v1(size1);

vector<int> v2(size2);

for (size\_t i = 0; i < v1.size(); ++i) {

v1[i] = rand() % 100;

}

for (size\_t i = 0; i < v2.size(); ++i) {

v2[i] = rand() % 100;

}

bool successfulCopy = false;

while (!successfulCopy) {

try {

if (v1.size() < v2.size()) {

throw runtime\_error("Големината на првиот вектор не е доволна за копирање на вториот вектор!");

}

copy(v2.begin(), v2.end(), v1.begin());

successfulCopy = true;

cout << "Копирањето беше успешно!" << endl;

} catch (const runtime\_error& e) {

cout << "Грешка: " << e.what() << endl;

size\_t newSize = rand() % 1000 + 1;

cout << "Генерирам нова големина за првиот вектор: " << newSize << endl;

v1.resize(newSize);

for (size\_t i = 0; i < v1.size(); ++i) {

v1[i] = rand() % 100;

}

}

}

Предавање 10

**1.**Напиши програма која демонстрира дека ако е фрлен покажувач на создаден објект, објектот нема да биде уништен.

class Objekt {

public:

int a;

Objekt(int a) {

this->a = a;

cout<<"Constructor"<<endl;

}

~Objekt() {

cout<<"Destructor"<<endl;

}

};

void throwObject() {

Objekt\* objekt = new Objekt(2);

throw objekt;

}

int main() {

try {

throwObject();

} catch(Objekt\* objekt) {

cout<<"Error: "<<objekt->a<<endl;

}

cout<<"Program continue"<<endl;

return 0;

}

**2.**Креирај класа **Garaza** во која има **Avtomobil** што може да има проблеми со својот **Motor**. Употреби **try** блок во конструкторот на **Garaza** за фаќање на исклучокот од **Motor** кога се иницијализира **Avtomobil**. Фрли друг исклучок од **catch** блокот во конструкторот на  **Garaza** и фати го во **main( )**.

class Motor {

public:

string problem;

Motor(string p) : problem(p) {}

};

class Avtomobil {

public:

string model;

bool defektNaMotor;

Avtomobil(string m, bool defekt) : model(m), defektNaMotor(defekt) {

if (defekt) {

throw Motor("Problem so motor kaj avtomobilot " + model);

}

}

};

// Класа за гаража

class Garaza {

public:

Avtomobil\* avtomobil;

Garaza(string model, bool defekt) {

try {

avtomobil = new Avtomobil(model, defekt);

} catch (Motor& motor) {

cout << "Greska vo Garaza Konstruktor: " << motor.problem << endl;

throw runtime\_error("Garazata ne moze da go skladira avtomobilot.");

}

}

~Garaza() {

delete avtomobil;

}

};

int main() {

try {

Garaza garaza("BMW", true);

cout << "Avtomobilot e uspesno skladiran vo garazata." << endl;

} catch (runtime\_error& e) {

cout << "Isklucok faten vo main: " << e.what() << endl;

}

try {

Garaza garaza("Toyota", false);

cout << "Avtomobilot e uspesno skladiran vo garazata." << endl;

} catch (runtime\_error& e) {

cout << "Isklucok faten vo main: " << e.what() << endl;

}

return 0;

}

Предавање 11

**1.**Напиши програма со која сите зборови од текстуална датотека со големина помала од 10 знаци ќе ги запише во друга текстуална датотека.

ifstream input("input.txt");

ofstream output("output.txt");

if(!input.is\_open()) {

cerr<<"Error: COuld not open input file: "<<input<<endl;

return 1;

}

if(!output.is\_open()) {

cerr<<"Error: COuld not open output file: "<<output<<endl;

return 1;

}

string word;

while(input >> word) {

if(word.length() < 10)

output << word <<endl;

}

input.close();

output.close();

Предавање 12

**1.**Креирај class temlate **Pole**што реализира поле од било кој тип. Имплементирај function template **max** за определување на најголем елемент во полето. Во **main** креирај полиња од **int,** **char** и **string**, пополни ги со по 5 произволни елементи и прикажи најголем елемент за секое поле.

template<typename T>

class Pole {

public:

list<T> lista;

Pole(list<T>& lista) {

this->lista = lista;

}

void max() {

T max = \*lista.begin();

for(const auto& element : lista) {

if(element > max)

max = element;

}

cout<<max<<endl;

}

};

**2.**Во еден магацин се води евиденција за артикли. Во текстуална датотека **inventar.txt** се чуваат следните податоци за секој артикл: инвентарен број (**int**), количество (**int**) и цена (**double**) за секој артикл.  Креирај класа**Inventar**која содржи**id**, **kolicestvo** и **cena**.  Напиши програма во која се создава **vector** од артикли, а потоа прикажува вкупна вредност на артиклите во магацинот и листа на сите артикли за кои количеството е помало од 10. Пресметувањето на вкупната вредност и прикажувањето на артиклите изврши го со STL алгоритми. Ако не постои влезна датотека, програмата да фрли исклучок.

class Inventar {

private:

int id;

int kolicestvo;

double cena;

public:

Inventar(int id, int kolicestvo, double cena) {

this->id = id;

this->kolicestvo = kolicestvo;

this->cena = cena;

}

double getCena() {

return kolicestvo \* cena;

}

int getKolicina() {

return kolicestvo;

}

friend ostream& operator<<(ostream& os, const Inventar& inventar) {

os <<id<<", "<<kolicina<<", "<<cena<<endl;

return os;

}

}

int main() {

fstream myFile;

vector<Inventar> magacin;

myFile.open("inventar.txt", ios::out);

if(!myFile.is\_open())

throw runtime\_error("The file is not open succesful!");

int id, kolicestvo;

double cena;

while(getline(myFile >> id >> kolicestvo >> cena)) {

magacin.push\_back(Inventar(id,kolicestvo,cena));

}

myFile.close();

double total = accumulate(magacin.begin(), magacin.end(), 0.0, [](double sum, const Inventar& inventar) {

return sum + inventar.getCena();

});

cout<<"Vkupnata cena e: "<<total<<endl;

for\_each(magacin.begin(), magacin.end(), [](const Inventar& ar) {

if(ar.getKolicina() < 10)

cout<<ar;

});

return 0;

}