Федеральное государственное бюджетное образовательное учреждение высшего

образования

«Чувашский государственный университет им. И.Н. Ульянова»

Кафедра вычислительной техники

Лабораторная работа № 8

**Шаблоны Классов. Стандартные шаблоны классов (STL).**

Вариант 9

Выполнил: Иванов В.С.

студент группы ИВТ-41-22

Проверил: кандидат технических наук

Обломов Игорь Александрович

Чебоксары, 2024

Цель работы: изучение шаблонов классов, стандартных шаблонов классов, использование их на практике.

Выполнение работы:

***Задание №1.***

Создать шаблон класса, моделирующий линейный однонаправленный список. Предусмотреть операции добавлении, удаления и поиск заданного элемента.

template <class T> class List1 {

protected:

int top;

T st[100];

public:

List1() {

top = -1;

}

void Push(T);

T Pop();

void Out();

void Del();

void Search(T);

};

template <class T> void List1 <T>::Out() {

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

cout << "Element:" << st[i] << " " << endl;

}

}

template<typename T> void List1<T>::Push(T var) {

st[++top] = var;

}

template<typename T> T List1<T>::Pop() {

return st[top--];

}

template<typename T> void List1<T>::Del() {

if (top >= 0) {

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

st[i] = st[i + 1];

}

top--;

}

}

template<typename T> void List1<T>::Search(T var) {

bool found = false;

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

if (st[i] == var) {

found = true;

}

else {

continue;

}

}

if (found == true) {

cout << var << " Found" << endl;

}

else {

cout << "Not Found" << endl;

}

}

***Задание №2-4.***

Создайте шаблон базового класса « символ », позволяющий хранить одиночные символы типа char, int и производный от него шаблон класса « строка », позволяющего хранить смесь произвольных символов определенной длины. Дополните задание дружественными перегруженными операциями вывода в стандартный поток. Продемонстрируйте вывод одиночного символа и строки в поток. Создайте статическое поле и метод(ы) работы с ним в шаблоне базового класса, покажите возможность наследования статических компонент. В созданной иерархии замените ключевое слово class, стоящее перед именем базового и производного шаблона классов на слово struct или union.

template<typename T, typename n> struct Symbol {

protected:

T value;

n number;

public:

Symbol() {};

Symbol(char ch, int n) : value(ch), number(n) {};

friend void out(const Symbol <T, int> & sym);

};

template <typename T, int max\_lenght = 15> struct String : public Symbol <T,int> {

T data[max\_lenght];

int lenght;

public:

String() {};

String(T, int);

void out();

};

template <class T, int max\_lenght> String <T, max\_lenght>::String(T var, int max\_lenght) {

lenght = max\_lenght;

}

template <class T, int max\_lenght> void out(const Symbol <T, int> & sym) {

using Symbol <T, int>::out();

this->Symbol <T, int>::out();

cout << sym.data << endl;

}

***Задание №5-6.***

Продемонстрировать работу алгоритма find() для последовательных контейнеров списки (list), очереди с двусторонним доступом (deque) и стек (stack). В программе необходимо подключить соответствующие заголовочные файлы. Продемонстрировать работу алгоритма count() для последовательных контейнеров списки (list), очереди с двусторонним доступом (deque) и стек (stack).

list <int> v={ 10, 20,20, 30, 40 };

auto f\_list = find(v.begin(), v.end(), 20);

int f\_count = count(v.begin(), v.end(), 20);

deque <int> s = { 10, 20,20, 30, 40 };

int s\_count = count(s.begin(), s.end(), 20);

auto f\_deq = find(s.begin(), s.end(), 20);

stack <int> t;

t.push(10);

t.push(20);

t.push(20);

t.push(30);

t.push(40);

stack <int> u;

int u\_count = 0;

bool found = false;

while (!t.empty()) {

if (t.top() == 20) {

found = true;

u\_count++;

}

u.push(t.top());

t.pop();

}

if (found == true) {

cout << "Found" << endl;

}

else {

cout << "Not Found" << endl;

}

***Задание №7.\****

Объявить классы Person и Student, имеющие поля Name, Age, идентифицирующие имя и возраст человека и студента. В каждом из классов предусмотреть поля, характерные для его типов, например для класса Student таким полем может быть Ball, а для класса Person – поле Prof, идентифицирующее профессию человека. Объявить два контейнера типа vector, в одном из которых находятся объекты типа Person, а во втором – объекты типа Student. Используя алгоритм search(), определить последовательность значений, заданных контейнером Student, попадающих в контейнер Person. Сравнение проводить по значениям полей Name. Показать пример работы алгоритма count() на заданных контейнерах, в качестве критерия поиска можно использовать поля Ball типа Student, подсчитывающего число студентов, сдавших последнюю сессию на 4 и на 5. Для класса Person критерием выбора может быть величина оклада.

class Person {

public:

string Name;

int Age;

double Salary;

Person(string Name, int age, double salary) : Name(Name), Age(age), Salary(salary) {};

};

class Student {

public:

string Name;

int Age;

int Marks;

Student(string Name, int age, int marks) :Name(Name), Age(age), Marks(marks) {};

};

int main() {

vector <Person> person = { {"Vall", 20, 1000.0}, {"Val", 21, 2000.0}, {"Va", 22, 3000.0} };

vector <Student> student = { {"V234", 20, 4}, {"Vafwel", 21, 3}, {"Vall", 22, 5} };

auto f1 = search(person.begin(), person.end(),student.begin(), student.end(),

[](const Person& pe, const Student& st) {

return pe.Name == st.Name; });

int count\_4 = count\_if(person.begin(), person.end(), [](const Student& st) {

return st.Marks == 4; });

int count\_5 = count\_if(person.begin(), person.end(), [](const Student& st) {

return st.Marks == 5; });

}

***Задание №8.***

Используя алгоритм merge() объединить два вектора и списка в результирующие контейнеры.

vector <int> ex\_8\_1 = { 1,2,3,4,5,6 };

vector <int> ex\_8\_2 = { 7,8,9,13,14 };

vector <int> ex\_8(ex\_8\_1.size() + ex\_8\_2.size());

merge(ex\_8\_1.begin(), ex\_8\_1.end(), ex\_8\_2.begin(), ex\_8\_2.end(), ex\_8.begin());

for (int i = 0; i < ex\_8.size(); i++) {

cout << ex\_8[i] << " ";

}

***Задание №9.***

Используя собственные функции как функциональные объекты, показать пример их применения к спискам, очередям с двусторонним доступом.

list <int> ex\_9\_1 = { 1,2,3,4 };

list <int > ex\_9\_1\_1(ex\_9\_1.size());

function<int(int)> nn2 = numnum;

transform(ex\_9\_1.begin(), ex\_9\_1.end(), ex\_9\_1\_1.begin(), nn2);

for (int nnum : ex\_9\_1\_1) {

cout << nnum << " ";

}

cout << endl;

deque <int> ex\_9\_2 = { 5,6,7,8 };

deque <int> ex\_9\_2\_1(ex\_9\_2.size());

transform(ex\_9\_2.begin(), ex\_9\_2.end(), ex\_9\_2\_1.begin(), nn2);

for (int nnum : ex\_9\_2\_1) {

cout << nnum << " ";

}

***Задание №10.***

Самостоятельно изучить последовательные контейнеры: векторы, списки и очереди с двусторонним доступом, а также алгоритмы и методы для работы с ними. Показать примеры их использования. Данный пункт был выполнен в процессе выполнения остальных заданий лабораторной работы.

***Задание №11-12.***

Определить вектор, очередь содержащие элементы стандартного типа (целого, вещественного и т. д.), определить для них итераторы и опробовать алгоритмы и методы работы с ними. Определить класс, состоящий из небольшого числа компонент, сохранить объекты этого класса в очереди. Показать пример использования итераторов при работе с очередью объектов собственного типа. Определить список, содержащий элементы собственного типа (например, из предыдущего примера), объявить обратный итератор и обеспечить вывод списка в обратном порядке.

class Ex\_11 {

public:

int id;

string nam;

Ex\_11(int id, string nam) : id(id), nam(nam) {};

};

vector <int> ex\_11 = { 3,4,5,6,1,2,54 };

auto iter\_1{ ex\_11.begin() };

cout << \*iter\_1 << endl;

\*iter\_1 = 100;

cout << ex\_11[0] << endl;

queue <int> ex\_11\_2 ;

ex\_11\_2.push(10);

ex\_11\_2.push(20);

ex\_11\_2.push(30);

ex\_11\_2.push(40);

ex\_11\_2.push(50);

ex\_11\_2.push(60);

queue <int> it\_ex\_11\_2;

while (!it\_ex\_11\_2.empty()) {

cout << it\_ex\_11\_2.front() << " ";

it\_ex\_11\_2.pop();

}

vector <Ex\_11> objve;

Ex\_11 obj\_1(1, "Vall");

objve.push\_back(obj\_1);

queue <Ex\_11> objqu;

for (int i = 0; i < objve.size(); i++) {

objqu.push(objve[i]);

}

while (!objqu.empty()) {

Ex\_11 obj = objqu.front();

cout << obj.id << " " << obj.nam << endl;

objqu.pop();

}

list <Ex\_11> objli;

Ex\_11 obj\_2(2, "Val");

objli.push\_back(obj\_1);

objli.push\_back(obj\_2);

list<Ex\_11>::reverse\_iterator rit;

for (rit = objli.rbegin(); rit != objli.rend(); rit++) {

cout << rit->id << " " << rit->nam << endl;

}

Вывод: изучил шаблоны классов, стандартные шаблонов классов, использовал их на практике.