Министерство Образования и Науки Российской Федерации

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

№ ____

«Технология программирования»

Методические указания к лабораторным работам для студентов по направлению 230100 «Информатика и вычислительная техника» очной формы обучения

НОВОСИБИРСК
2017

УДК 004.45 (076.5)

Составители: ***Д.О. Романников*****, канд. техн. наук, доц. каф. Автоматики**

Рецензент ***А.В. Гунько*****, канд. техн. наук, доц. каф. Автоматики**

Работа подготовлена на кафедре автоматики

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА №1

Цель работы: изучить основы работы с классами в C++, а именно: структуру классов, спецификаторы области видимости, реализацию методов, в том числе, конструкторов и деструкторов.

Общие сведения

C++ — компилируемый статически типизированный язык программирования общего назначения. Является вторым по распространению языком в мире после Java. C++ используется не только в персональных и серверных компьютерах, но и широко распространен для программирования микроконтроллеров и DSP процессоров.

Hello world на C++

Самой распространенной программой для начала изучения языка программирования является программа “Hello world”. Рассмотрим программу, приведенную ниже:

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

void main()
{
  cout << "Hello world!" << endl;
  cout << "Let’s start learn the C++ language" << endl;
  return 0;
}
```

Данная программа выводит на консоль строки «Hello world» и «Let’s start learn the C++ language». Реализуется при помощи выражения «*cout* <<», которое используется для вывода на поток *stdout* выражения стоящего правее. Для работы с «*cout*» необходимо подключить библиотеку <*iostream*> и указать, что будет использоваться пространство имен (*namespace*) *std*.

Классы

В C++ для программирования в объектно-ориентированном стиле используются классы и объекты. Пример объявления класса приведен ниже:

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

class Line
{
private:
  double length;
public:
  void set_length(double);
  double get_length(void);
  line();
};
```

Необходимо отметить, что хорошей практикой является разделение объявления класса и его реализации по отдельным файлам. Например, вышеприведенный код необходимо разместить в файле с названием класса – «line» и расширением «h» или «hpp» (line.h/line.hpp). А реализацию методов класса разместить отдельно в файле «line.cpp». Рассмотрим эту реализацию:

```
#include “line.h”
using namespace std;
void Line::set_length( double len )
{
  length = len;
}

double Line::get_length( void )
{
  return length;
}
```

Методы класса имеют доступ к атрибутам класса: могут получать значение переменных и изменять его. Пример использования реализованного класса приведен ниже:

```
int main()
{
  Line new_line;
```

```
  cout << endl;

  return 0;
}
```

Классы в C++ могут иметь различную область видимости, а именно:

- *private* – атрибуты или методы доступны только «внутри» класса;
- *protected* – атрибуты или методы доступны «внутри» класса, а также классам потомкам;
- *public* – атрибуты или методы доступны как «внутри» и «снаружи» класса, так и для потомков.

Работа с памятью в C++

C++ как язык, не содержащий «сборщика мусора», предполагает ручное управление памятью. Для выделения памяти используется оператор *new*, а для ее освобождения оператор *delete*.

Синтаксис выделения и освобождения памяти с использованием операторов *new* и *delete* приведен ниже:

```
p_var = new typename;
delete p_var
```

Где p_var – это указатель типа typename. Рассмотрим пример использования операторов *new*/*delete* для классов:

```
#include <iostream>
#include “line.h”
using namespace std;

int main()
{
  Line *new_line = new Line;

  new_line->set_length(6.0);
  cout << "Length of line : " << new_line->get_length();
  cout << endl;

  delete new_line;
  return 0;
}
```

Конструкторы, деструкторы

При программировании в ООП стиле часто бывает необходимым выполнить начальную инициализацию объекта, например: выделить память под атрибуты класса, создать файл или выполнить запрос в базу данных. Для этих целей используются *конструкторы*. *Деструктор* имеет обратное назначение. Данные методы подчиняются следующим правилам:

- Конструктор имеет такое же имя, как и класс;
- Конструктор не имеет возвращаемого значения;
- Когда в программе создается экземпляр класса, компилятор вызывает конструктор, если конструктор существует, или конструктор по умолчанию, если конструктор не существует;
- Деструктор имеет такое же имя, как и класс, но содержит тильду «~» перед именем;
- Деструктор не имеет возвращаемого значения.

Рассмотрим пример реализации конструктора:

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

class Line
{
private:
  double *length;
public:
  Line(int);
  ~Line();
  void set_length(double);
  double get_length(void);
  Line();
};
Line::Line(void)
{
  length = new double;
}
Line::~Line(void)
{
  delete length;
}
```

```
{
  *length = len;
}
double Line::get_length( void )
{
  return *length;
}
int main()
{
  Line *new_line = new Line;

  new_line->set_length(6.0);
  cout << "Length of line : " << new_line->get_length();
  cout << endl;

  delete new_line;
  return 0;
}
```

Хорошей практикой программирования является покрытие кода *unit-тестами* – специальными тестами, которые позволяют проверить на корректность отдельные части кода (классы, методы). Задача любого теста сопоставить результат выполнения проверяемого кода с ожидаемым результатом. Например, проверяя если необходимо проверить функцию *length* класса *std::string* то один из вариантов проверки будет иметь следующий вид:

```
bool checkPositiveLength() {
  string s(“aaa”);
  return s.length() == 3;
}
bool checkEmptyString() {
  string s(“”);
  return s.length() == 0;
}
```

Порядок выполнения работы

1. Реализовать функции программы согласно варианту. Сделать атрибуты класса приватными. Добавить в класс методы для валидации входных данных в методах. Общий код вынести в отдельные методы;
2. Запретить всем методам (для которых это возможно по смыслу) менять состояние класса.

Методические указания

1. Выполните декомпозицию поставленной задачи. Покажите взаимосвязи между классами;
2. Задайте методам класса минимально допустимые области видимости;
3. Выполните проверку входных данных для публичных методов класса.

Варианты:

1. Разработать класс для работы с комплексными числами. Должны быть поддерживаться операции сложения, вычитания, умножения, деления с комплексными и вещественными числами, операции получения вещественной и мнимой частей;
2. Разработать класс для работы с рациональными числами. Должны быть поддерживаться операции сложения, вычитания, умножения, деления с рациональными числами, операции получения числителя и знаменателя;
3. Разработать класс для поиска заданного слова в файле;
4. Разработать класс для подсчета числа вхождений каждого слова в файле;
5. Разработать класс для преобразования числа из шестнадцати разрядного (текстового) представления в десятичное(численное). Без использования библиотечных функций;
6. Разработать класс для подсчета слов в файле по заданному разделителю;
7. Разработать класс для работы с матрицами. Должны быть поддерживаться операции сложения, вычитания и вывода результата в консоль;
8. Разработать класс для отображения календаря для заданного года с разбиением на месяцы;

9. Разработать класс «Калькулятор», в котором будет выполняться базовые арифметические (+, -, *, /) и тригонометрические (cos, sin, tg, ctg) действия;
10. Разработать класс для преобразования числа в двоичное представление. Без использования библиотечных функций;
11. Разработать класс для работы с матрицами. Должны быть поддерживаться операции сложения, вычитания матрицы с числом и вывода результата в консоль;
12. Разработать класс для работы с векторами. Должны быть поддерживаться операция умножения вектора на число, умножение двух вектором и вывода результата в консоль.

Контрольные вопросы

1. Для чего нужны классы?
2. Какими способами выполняется обращение к методам/атрибутам объекта?
3. Какие бывают спецификаторы областей видимости, чем они отличаются?
4. Как в C++ выполняется вывод на *stdout/stderr*?
5. В каких случаях класс может содержать несколько конструкторов? Как компилятор определяет какой конструктор необходимо вызвать?
6. При каких условиях вызывается деструктор класса? Примеры.
7. Когда нужно выделять память под переменные статически, когда динамически?

ЛАБОРАТОРНАЯ РАБОТА №2

Цель работы: изучить основные принципы наследования классов в C++.

Общие сведения

Наследование – механизм, позволяющий написать новый класс (дочерний, производный) на основе уже существующего класса (родительского, базового). При этом поведение дочернего класса определяется методами и атрибутами, которые получены от класса-родителя, а также новыми методами и атрибутами дочернего класса.

Рассмотрим пример наследования классов.

```
class Parent {};
class Child_1: public Parent {};
class Child_2: protected Parent {};
class Child_3: private Parent {};
```

В данном примере от базового класса *Parent* наследуются классы *Child_1*, *Child_2*, *Child_3*, причем все наследуемые классы имеют различные спецификаторы наследования: *public*, *protected*, *private*. *Public*-наследование означает, что те методы и атрибуты базового класса, которые были объявлены как *public* и *protected* остаются *public* и *protected* после наследования соответственно. *Protected*-наследование означает, что те методы и атрибуты базового класса, которые были объявлены как *public* и *protected* будут иметь спецификатор доступа *protected* после наследования. *Private*-наследование означает, что те методы и атрибуты базового класса, которые были объявлены как *public* и *protected* будут иметь спецификатор доступа *private* после наследования.

Стоит заметить, что не все методы и атрибуты класса наследуются. Наследованию не подлежат все конструкторы, деструкторы класса, методы и атрибуты, которые имеют спецификатор *private*.

Конструкторы

Конструкторы при наследовании классов не наследуются. Также объявление конструктора в классе-потомке выглядит иначе. Рассмотрим пример.

```
class Parent
```

```
  int x, y;
public:
  Parent(int a, int b);
};
Parent::Parent(int x, int y) {
  this->x = x;
  this->y = y;
}

class Child: public Parent
{
  int z;
public:
  Child(int x, int y, int z);
};
Child::Child(int x, int y, int z): Parent(x, y)
{
  this->z = z;
}
```

Обратите внимание на то, каким образом объявлен конструктор для класса *Child*: после обычного объявления конструктора следует вызов конструктора родительского класса с аргументами из конструктора класса-наследника.

Порядок вызова конструкторов/деструкторов

При наследовании важное значение имеет то, в каком порядке будут вызываться конструкторы/деструкторы родительского и дочернего классов. Порядок вызова конструкторов/деструкторов следующий: при создании объекта дочернего класса конструкторы вызываются в порядке от родительского класса к классу-наследнику. Деструкторы вызываются строго в противоположном порядке – от дочернего класса к родительскому.

Переопределение методов

Цель наследования классов – изменить поведение класса потомка на основании поведения класса-родителя. Для этого можно добавлять новые методы (чаще всего с использованием методов родительского класса) или переопределить поведение существующих.

Порядок выполнения работы

1. Реализовать функции программы согласно варианту. Покажите, где в программе используется отношение наследования. Объясните, в чем преимущество использования отношения наследования перед агрегацией в используемом примере;
2. При использовании наследования задайте методам минимально возможную по смыслу область видимости.

Варианты:

1. Разработать классы для работы с матрицами. В матрице могут содержаться вещественными и комплексные числа. Должны поддерживаться операции сложения, вычитания матриц;
2. Разработать класс для работы с матрицами, в которых могут находиться рациональные и вещественные числа. Должны поддерживаться операции сложения, вычитания матриц;
3. Разработать класс калькулятора, который может выполнять операции как с арабскими числами, так и с римскими (в рамках лабораторной работы можно ограничиться числами от 0 до 10);
4. Разработать иерархию классов для НГТУ, где должны быть отражены такие сущности как «институт», «факультет», «кафедра», «военная кафедра», «лаборатория», «хим. лаборатория», «компьютерный класс». Задать свойства классов и методы;
5. Разработать класс для поиска заданного слова в: 1) файле с расширением *txt*; 2) файле с расширением *«lab»*; 2) файле с расширением *«notes»*;
6. Разработать класс для работы с векторами. Должны быть поддерживаться операции умножения двух векторов и вывода результата в консоль. В векторе могут быть находиться как арабские, так и римские числа (в рамках лабораторной работы можно

7. Разработать иерархию классов, где должны быть отражены в виде классов такие сущности как «int», «long», «double», «boolean». Задать свойства классов и методы;
8. Разработать класс для работы с векторами. Должны быть поддерживаться операции умножения двух векторов и вывода результата в консоль. В векторе могут быть находиться как вещественные, так и рациональные числа;
9. Разработать класс для работы с векторами. Должны быть поддерживаться операции умножения вектора на число и вывода результата в консоль. В векторе могут быть находиться как целые числа в десятичном, так и/или шестнадцати разрядном (текстовом) представлении;
10. Разработать иерархию классов для новосибирского зоопарка, где должны быть отражены такие сущности как «лев», «тигр», «лигр», «белый медведь», «бурый медведь», «орел», «утка». Задать свойства классов и методы;
11. Разработать программу для поиска товара на складе. Склад задается двумерным массивом. Товар может занимать произвольное количество ячеек массива. Если ячейка занята одним товаром, то другой товар в нее положить невозможно. Реализовать классы склада и товаров, которые занимают следующий объем: 1) 1x1; 2) 1x2; 2) 2x2. Задать свойства классов и методы;
12. Разработать иерархию классов для магазина, где должны быть отражены различные товары, например: «торт», «лапша», «макароны», «пирожное», «молоко». Задать свойства классов и методы. На основании полученного набора классов пользователь должен быть в состоянии сформировать чек.

Контрольные вопросы

2. Как работает наследование в C++? Какие области видимости наследуются?
3. Почему в C++ принято определять виртуальный деструктор?
4. Наследуются ли конструкторы/деструкторы классов?
5. Каким образом можно из класса потомка обратиться к методам/атрибутам родительского класса?
6. Как происходит вызов конструкторов в цепочке классов наследников?

ЛАБОРАТОРНАЯ РАБОТА №3

Цель работы: изучить основные принципы работы с контейнерами *vector, list, deque* STL и их итераторами в C++.

Общие сведения

В стандартной библиотеке (STL) C++ для работы с распространенными структурами данных есть множество реализованных классов для упрощения работы с ними. Основными контейнерами для представления списков являются: *vector, list, deque*.

Для работы с массивами в C++ используется контейнер *vector*. Создание вектора выполняется следующим образом: *«vector<type> variable_name;»*, где *type* – любой валидный тип данных, а *variable_name* – имя контейнера. Основными методами для работы с векторами являются:

```
v.push_back(100); // Добавление нового значения (100) в конец массива
v.pop_back(); // Извлечение из массива последнего значения
v.at(10) = 5; //v[10] = 5; Обращение к 10-ому элементу массива
size_t size = v.size(); // Получение размера массива
bool v.empty(); // Проверка на то, что массив пуст
v.clear(); // Удаляет все элементы массива
```

Контейнер *list* также предназначен для работы с массивами, но в отличии от вектора, в котором элементы массива хранятся в непрерывной последовательной области памяти, элементы массива хранятся в

произвольных ячейках памяти, что приводит к более медленной сортировке и более быстрому добавлению/удалению в произвольном месте массива.

Создание списка выполняется также, как и для вектора, за исключением использования ключевого слова *list*: *«list<type> variable_name;»*.

Контейнер *deque* представляет собой двухстороннюю очередь, что позволяет эффективно выполнять операции вставки и удаления как в конец контейнера, так и в начало. Контейнеры *deque* и *vector* предоставляют похожую функциональность, но устроены по-разному. В отличие от вектора, который хранит данные в одном последовательном массиве, данные в *deque* разбиты на линейные массивы, которые могут быть не связаны между собой. Более сложное строение *deque* позволяет получить большую производительность по сравнению с *vector* при добавлении больших последовательностей данных неизвестной длины.

Итератор (iterator) – специальный тип данных, предназначенный для универсального перебора элементов контейнеров.

Во всех контейнерах C++ есть методы *begin()* и *end()*, возвращающие итераторы для соответствующих контейнеров. Перебор элементов с использованием итераторов будет выглядеть следующим образом:

```
vector<int> myvector;
for (vector<int>::iterator it = myvector.begin(); it != myvector.end(); ++it)
  cout << *it << endl;
```

Порядок выполнения работы

1. Реализовать функции программы согласно варианту. Объяснить использования выбранных структур данных;
2. Посчитать асимптотическую сложность получившегося решения.

Варианты:

1. Разработать класс динамического массива на основе статического

2. Разработать метод класса, в котором будет выполняться обращение связного списка (без использования алгоритмов STL);
3. Разработать метод класса, в котором будет выполняться обращение связного списка начиная от n-ого элемента и заканчивая элементом m (без использования алгоритмов STL). n <= m;
4. Разработать класс базы пользователей, в котором хранится список пользователей, а также выполняются операции поиска пользователя по заданному имени, поиск суммы счетов пользователей, поиск пользователя с максимальным, минимальным состоянием счета;
5. Разработать класс, в котором элементы в всегда хранятся в отсортированном порядке. Класс должен поддерживать операции добавления, удаления элементов и вывод содержимого на консоль;
6. Разработать класс *deque* с использованием двухсвязного списка и контейнера *vector*;
7. Выполнить исследование сравнения *list* и *vector*. Построить графики времени добавления/поиска 1к/10к/100к/1М элементов. Объяснить результаты;
8. Выполнить исследование сравнения *deque* и *vector*. Построить графики времени добавления/поиска 1к/10к/100к/1М элементов. Объяснить результаты;
9. Для программы из прошлой лабораторной работы реализовать функциональность хранения истории выполняемых операций и ее отображение;
10. Выполнить исследование сравнения *deque* и *vector*. Построить графики времени удаление элемента из контейнера из 1к/10к/100к/1М элементов. Объяснить результаты;
11. Выполнить исследование сравнения *list* и *vector*. Построить графики времени удаление элемента из контейнера из 1к/10к/100к/1М элементов. Объяснить результаты;

12. Выполнить исследование сравнения *list* и *deque*. Построить графики времени удаление элемента из контейнера из 1к/10к/100к/1М элементов. Объяснить результаты.

Методические указания

1. Для реализации сортировки используйте подходящий контейнер. Объясните выбор контейнера;
2. При использовании C++11 удобно создавать итераторы при помощи ключевого слова *auto*: *auto it = myvector.begin()*;

Контрольные вопросы

1. Какие есть контейнеры в C++?
2. Для чего нужны контейнеры?
3. В чем отличие массивов от *vector/list/deque*?
4. Что такое итераторы? Для чего они используются?
5. Как итерироваться по контейнерам?

ЛАБОРАТОРНАЯ РАБОТА №4

Цель работы: изучить основные принципы работы с контейнерами *stack, queue, heap, prioprity_queue* STL и их итераторами в C++.

Общие сведения

В C++ для работы с такими структурами данных как стеки и очереди реализованы контейнеры *stack* и *queue*. Стек реализует поведение *LIFO* (*last-in first-out*), при котором элементы добавляются/извлекаются в/из с одной стороны контейнера.

Таким образом, основными операциями для стека являются следующие операции:

```
stack<int> s;
```

```
int val = s.top (); // Получение значения вершины стека
s.pop() // Удаление вершины стека
s.empty() // проверка стека на наличие элементов
s.size() // получение количества элементов в стеке
```

В отличии от стека очереди реализуют поведение *FIFO* (*first-in first-out*), при котором элементы изначально попадают в конец очереди, а извлекаются из ее начала. Интерфейс *queue* похож на интерфейс *stack*:

```
queue<int> q;
q.push(1); // Добавление нового значения (1)
q.push(2); // Добавление нового значения (2)
int val = q.front(); // Получение первого элемента в очереди (1)
int b = q.back() // Получение последнего элемента в очереди (2)
q.pop() // Удаление первого элемента
q.empty() // Проверка очереди на наличие элементов
q.size() // Получение количества элементов в очереди
```

В отличии от всех других контейнеров для очереди и стека в C++ нет итераторов, при этом для «итерирования» по данным контейнерам можно выполнять следующим образом (для стека аналогичным образом):

```
while (!q.empty()) {
    cout << ' ' << q.front();
    q.pop();
}
```

Для представления очереди с приоритетом в STL C++ реализован класс *priority_queue*. В отличии от обычной очереди, очередь с приоритетом определяет порядок внутри очереди не только по очередности добавления, но и по передаваемому параметру – приоритету. В независимости от очередности добавления элемент с большим приоритетом будет извлечен из очереди первым. Методы работы с такой очередью приведены ниже:

```
std::priority_queue<int> pq;
pq.push(30);
pq.push(100);
pq.push(25);
pq.push(40);
while (!pq.empty()) {
  cout << ' ' << pq.top(); // Элементы будут извлечены в
  pq.pop();                // следующем порядке:100 40 30 25
}
```

Структура данных пирамида (*heap*) представляет собой бинарное дерево, в

значение узла должно быть больше, чем любое из значений потомков данного узла (для *min-heap* - меньше). В C++ для работы с *heap* в STL существуют следующие функции:

```
int myints[] = {10,20,30,5,15};
vector<int> v(myints,myints+5);
make_heap (v.begin(),v.end()); // создание heap на основе вектора v
v.front(); // получение максимального элемента в heap (30)
pop_heap(v.begin(),v.end()); // удаление максимального элемента
v.pop_back(); // выполняется в две операции
v.front(); // получение максимального элемента в heap (20)
v.push_back(40); // добавление нового элемента также выполняется
push_heap(v.begin(),v.end()); // в две операции
v.front(); // получение максимального элемента в heap (40)
```

Приведенный выше код демонстрирует работу *max-heap*, для получения структуры для *min-heap* необходимо использовать переопределенный объект для сравнения:

```
struct comparator {
  bool operator()(int i, int j) {
    return i > j;
  }
};
make_heap(v.begin(), v.end(), comparator());
```

Порядок выполнения работы

1. Реализовать функции программы согласно варианту. Объяснить использования выбранных структур данных;
2. Посчитать асимптотическую сложность получившегося решения.

Варианты:

1. Написать класс, который будет выполнять обращение строки с использованием стека;
2. Реализовать класс, который хранит список пар pair<string,int>, где первое число – имя пользователя, второе – сумма набранных балов. Реализовать метод, который будет выводить на консоль имена топ-3 пользователей с максимальными баллами;
3. Для списка пар pair<string,int>, где первое число – имя пользователя,

вывод на консоль всех имен пользователей в отсортированном по убыванию баллов порядке;

4. Реализовать класс стека с использованием связного списка;
5. Реализовать класс стека с использованием массива;
6. Реализовать класс очереди с использованием массива;
7. Реализовать класс очереди с использованием связного списка;
8. Реализовать класс, в котором будет проверяться правильность скобочной последовательности. К примеру «()()», «((фыв))» - правильные последовательности, «(()», «())(» - неправильные последовательности;
9. Реализовать класс очереди с приоритетом с использованием *heap*;
10. Реализовать класс, в котором выполняется вывод на консоль всех фалов в директории рекурсивно. В рамках лабораторной работы можно задать «псевдоструктуру» директорий в программе;
11. Реализовать класс, который эффективно выполняет операции стека и определения максимального значения в стеке;
12. Разработать класс «Редактор», в котором реализуется функциональность «Undo» для редактируемого текста.

Контрольные вопросы

1. Как работает контейнер *stack*?
2. Как работает контейнер *queue*?
3. Какие функции в C++ позволяют работать с *heap*?
4. Принцип работы контейнера *priority_queue*?

ЛАБОРАТОРНАЯ РАБОТА №5

Цель работы: изучить основные принципы работы с контейнерами *map, set, unordered_map, unordered_set* STL и их итераторами в C++.

Общие сведения

Важным контейнером для ассоциации по типу ключ-значение, является контейнер *map*. Для создания контейнера необходимо подключить заголовочный файл <*map*> и создать переменную *«map <key type,data type> variable _name;»,* где *key_type* и *data_type* тип данных ключа и значения соответственно. Основными методами для работы с контейнером являются:

```
map<string,int> m; // Создание словаря со строковым ключом
m.insert(pair<string,int>(“key”,1));
m[“key3”] = 2; // добавление новых элементов в словарь

map<string, int>::iterator it;
it = m.find(“some-key”);

if (it != m.end()) cout << “Ключ найден” << endl;
else cout << “Ключ не найден” << endl;

if (it != m.end()) {
  m.erase(it); // Удаляет элемент с заданным итератором
}
cout << “Значение ключа 1: ” << m.find(“1”)->second << endl;

m.size(); // Получение размера
m.clear(); // Удаляет все элементы массива
```

std::unordered_map<K,V> также представляет собой словарь и реализует такой же интерфейс, но в отличии от *map* операции в нем выполняются в среднем за $O_{ср}(1)$ (при этом перестроение хеш-таблицы занимает O(n)).

Контейнеры *std::set<T>* и *std::unordered_set<T>* представляют собой абстракцию над математическим понятием множества. Примеры использования такого словаря приведены ниже:

```
set<int> s;
for (int i=0; i<5; ++i) s.insert(i); // Добавление значений от 0 до 5
s.insert(3); // Элемент не будет добавлен. Такой уже есть
set<int>::iterator it;
it = s.find(4); // Операция поиска возвращает итератор
if (it != s.end()) {
  s.erase(it); // Удаление найденного элемента
}
```

Порядок выполнения работы

1. Реализовать функции программы согласно варианту. Объяснить использования выбранных структур данных;
2. Посчитать асимптотическую сложность получившегося решения.

Варианты:

1. Разработать класс для определение переданных в URL параметров;
2. Разработать класс для считывания значений конфигурационного файла в формате: «ключ» : «значение»;
3. Разработать класс для подсчета числа вхождений каждого слова в файле;
4. Разработать класс для подсчета числа вхождения уникальных слов в файле;
5. Разработать класс телефонной книги. Разработанный класс должен поддерживать операции добавления нового абонента, поиска по имени, поиска по телефону;
6. Разработать класс базы автомобилей. Разработанный класс должен поддерживать операции добавления нового автомобиля, поиска по номеру, поиска по марке;
7. Разработать класс для определения разницы между набором слов в двух переданных строках;
8. Разработать класс базы пользователей. Разработанный класс должен поддерживать операции добавления и удаления нового пользователя, поиска по имени, поиска по *id*;
9. Реализовать класс «Пользователь», в котором будут храниться идентификаторы друзей пользователя. Реализовать класс, в котором: 1) будет определяться являются ли друзьями два переданных пользователя; 2) является хотя бы один двух переданных пользователей другом второго;

10. Выполнить исследование сравнения *map* и *unordered_map*. Построить графики времени добавления/поиска 1к/10к/100к/1М элементов. Объяснить результаты;
11. Выполнить исследование сравнения *set* и *unordered_set*. Построить графики времени добавления/поиска 1к/10к/100к/1М элементов. Объяснить результаты;
12. Разработать класс базы расписания автобусов. Разработанный класс должен поддерживать операции поиска расписания заданного номера автобуса, определение номеров рейсов автобусов в указанное время.

Контрольные вопросы

1. В чем различие между *map* и *unordered_map*?
2. В чем различие между *set* и *unordered_set*?

ЛАБОРАТОРНАЯ РАБОТА №6

Цель работы: изучить работу с исключениями.

Общие сведения

Исключения – механизм языков программирования, предназначенный для описания реакции программы на различные ошибки и/или проблемы (исключения). Также исключения позволяют значительно упросить затраты на обработку ошибок. Рассмотрим пример обработки исключений.

```
try {
  throw 1;
} catch (int a) {
  cout << "Exception handler: " << a << endl;
  return;
}
cout << "No exception detected!" << endl;
```

В вышеприведенном примере используются следующие ключевые слова: *try, throw, catch*. Конструкция *throw* используется для «кидания» исключения, где объектом исключения является аргумент. Конструкция *catch (int a) {}*

«кидаются» в конструкции *try {}*. При перехвате исключения управление в программе передается в *catch* блок.

В стандартной библиотеке C++ используется иерархия классов исключений, где базовым классом является *exception*, которой содержит метод *virtual const char* what() const noexcept* для описания исключительных ситуаций.

Если в блоке *try* «кидается» несколько различных типов исключений, то можно использовать несколько *catch* конструкций, например:

```
try {

} catch (int a) {
  cout << "Exception handler: " << a << endl;
} catch (exception& e) {
  cout << "Exception handler: " << e.what() << endl;
} catch (…) {
  cout << "Catch all!" << endl;
}
```

В вышеприведенном примере *catch(...)* используется для перехвата всех исключений.

Порядок выполнения работы

1. Для программы из предыдущей лабораторной работы реализовать обработку ошибок в программе при помощи исключений.

Методические указания

1. В качестве объектов исключений используйте иерархию классов исключений;
2. Для различных по смыслу ошибок используйте различные классы исключений;
3. В базовом классе исключений реализуйте виртуальную функцию, в которой будет выводиться информация о классе объекта, файле и номера строки.

Контрольные вопросы

1. Для чего необходимы исключения в C++?
2. Каким образом перехватываются исключения в C++?

3. Иерархия стандартных исключений в STL.
4. Каким образом выполняется освобождение памяти при возникновении исключительных ситуаций?
5. Зачем нужны именованные исключения?

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

Методические указания к лабораторным работам

Редактор
Технический редактор

Лицензия № 021040 от 22.02.96. Подписано в печать ___.___.___.
Формат 60 x 84 1/16. Бумага оберточная. Тираж 50 экз.
Уч.-изд.л. 2,0. Печ.л. 2. Изд. № _____. Заказ № _____ Цена договорная

Отпечатано в типографии
Новосибирского государственного технического университета
630092, г. Новосибирск, пр. К. Маркса, 20