Объектно-ориентированное программирование и проектирование

© Хуторова Ольга Германовна

Лекция 6

Темы лекции

- Объектно-ориентированное программирование
  - Классы
    - Инкапсуляция
    - Наследование
    - Полиморфизм
  - Шаблоны классов
  - STL

C++
Базовые типы:

- целые типы (int, unsigned int, long int)
- типы с плавающей запятой (float, double, long double)
- символьный тип (char)
- логический тип (bool)
- тип void

C++ Производные типы

- перечислимые типы (enum)
- указатели
- ссылки
- массивы
- структуры и классы

Структуры в C++

```
struct Person
{ string FirstName;       // Имя
 string LastName;         // Фамилия
int BirthYear;           // Год рождения
 };

Person Vasya;
Vasya.FirstName=“Basil";
Vasya.LastName=“Pupkin";
Vasya.BirthYear=1992;
cout<<Vasya.FirstName<<" "<<Vasya.LastName<<
 " was born in "<<Vasya.BirthYear<<" year"<<endl;
```

Функции - часть типа данных

- основная идея объектно-ориентированного программирования

Классы, данные и методы класса

- Класс (class) - это тип, определяемый пользователем, включающий в себя данные и функции, называемые методами или функциями-членами класса.
  - Данные класса - то, что класс знает.
  - Функции-члены (методы) класса - то, что класс делает.

Объявление класса

- Объявление класса содержит объявление элементов данных и прототипы функций-элементов класса.
- Переменные могут быть любого типа, включая другие классы.
- Переменные объявленные внутри класса имеют область видимости класса, т.е. от точки объявления переменной до конца класса.

```
class class_name
{
public:
   int data_member;         // Элемент данных
   void show_member(int);   // Функция-элемент
};
```

Использование класса

- Объявление переменных типа этого класса (*объектов)*:

  class_name object_one, object_two;

- Работа с объектами – через методы класса

  object_one. show_member(int);

Управление доступом к классу

- **public** – доступ открыт всем, кто видит определение данного класса.
- **private** – доступ открыт самому классу (т.е. функциям-членам данного класса) и друзьям (friend) данного класса, как функциям, так и классам.
- **protected** – доступ открыт классам, производным от данного.

по умолчанию объявления в классе считаются частными.

Управление доступом к классу

```
class Example
{
 int x1;          // частные по умолчанию
 int f1(void);
protected:
      int x2;     // защищенные
       int f2(void);
private:
      int x3;     // опять частные
      int f3(void);
public:
       int x4;    // общедоступные
      ine f4(void);
 };
```

Использование класса - определение

```
class employee
{
    public:
       char name [64];
       long employee_id;
       float salary;
       void show_employee(void)
           {  cout << "Name: " << name << endl;
             cout << "ID: " << employee_id << endl;
              cout << "salary: " << salary << endl;
           };
        };
```

Использование класса

```
void main(void)
{
        employee worker, boss;

        strcpy(worker.name, "John Doe");
        worker.employee_id = 12345;
        worker.salary = 25000;

        strcpy(boss.name, "Happy Jamsa");
        boss.employee_id = 101;
        boss.salary = 101101.00;
        worker.show_employee();
        boss.show_employee();
    }
```

ОПРЕДЕЛЕНИЕ МЕТОДОВ КЛАССА ВНЕ КЛАССА

```
//Прототип функции
    void show_employee(void);
```

Ниже

…

```
    void employee:: show_employee (void)
```

Библиотека iostream

- #include <iostream>
- Операции ввода/вывода выполняются с помощью классов istream (потоковый ввод) и ostream (потоковый вывод). Третий класс, iostream, является производным от них и поддерживает двунаправленный ввод/вывод. Для удобства в библиотеке определены три стандартных объекта-потока:
- cin – объект класса istream, соответствующий стандартному вводу. В общем случае он позволяет читать данные с терминала пользователя;
- cout – объект класса ostream, соответствующий стандартному выводу. В общем случае он позволяет выводить данные на терминал пользователя;
- cerr – объект класса ostream, соответствующий стандартному выводу для ошибок. В этот поток мы направляем сообщения об ошибках программы.

Библиотека **string**

```
// Declare Preprocessor Directives
#include <string>
// Declare to use the namespace std.
using namespace std;
// Declare a string object.
string mystring = "Hello World";
// If you do not declare using namespace std.
// You must use the full name:
std::string mystring = "Hello World";
```

Пример

```
#include <iostream>
#include <string>
using namespace std;
int main() {
string in_string;
// вывести литерал на терминал пользователя
cout << “Input yours name: ";
// прочитать ответ пользователя в in_string
 cin >> in_string;
if ( in_string.empty() )
// вывести сообщение об ошибке на терминал пользователя
        cerr << “error, the string is empty !\n";
   else
        cout << “Hello!, " << in_string << "! \n";
return 0;
}
```

Функции конструктора и деструктора

- Не нужны, если создаются простые переменные
- Конструктор – выделение памяти и инициализация
- Деструктор – действие при освобождение памяти или закрытии потока
- Имя функции конструктора должно совпадать с именем класса, а после него должны следовать круглые скобки ().
- Функция конструктора должна иметь тип void, но не нужно это указывать.
- Функция конструктора должна располагаться под ключевым словом public.

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

class Myclass
    {
       public:
       Myclass( int x) //конструктор
         {data = x;}
       void show_member()
           {cout << data << endl;};
    //private:
     int data;
};
```

```
int main()
{
    Myclass A1(-111);
    A1.show_member();

    Myclass *A2 = new Myclass (-999);
    A2->show_member();
    delete A2;
    return 0;
}
```

Пример: Обьявление класса

```
class Point

    {
    public:
       Point();
       ~Point();
       void show_Point(void);
       void Distance(Point);
    private:
       float x,y;
    };
```

Пример1: Определение функций-членов класса

```
Point::Point()
    {
    x=100.*(float)rand()/RAND_MAX;
    y=100.*(float)rand()/RAND_MAX;
    }
//=====================================
  void Point::show_Point(void)
    {
    cout << "(" << x <<","<<y << ")"<<endl;
    }
//=====================================
    void Point::Distance(Point B)
    {
    cout << sqrt((x-B.x)*(x-B.x)+(y-B.y)*(y-B.y))<<endl;
    }
```

Пример1: Вызов метода класса

```
void main()

  {
     Point A, A1;
     A.show_Point();
     A1.show_Point();
     A.Distance(A1);
  }
```

Модульность

- описания классов включают в заголовочные файлы (*.H), а реализацию функций-членов классов - в файлы *.CPP.

три основных свойства ООП

- **Инкапсуляция** - сведение кода и данных воедино в одном объекте, реализация которого скрыта от пользователя

Инкапсуляция

```
class file
{
 public:
    void print ();
    void delete() ;
private:
    int readMessage();
    void nextMessage();
    string srcFile;
}
```

```
 void main()
{
file  my_file;
…
my_file.print() ;
my_file.delete();
}
```

три основных свойства ООП

- **Наследование** - наличие в языке ООП механизма, позволяющего объектам класса наследовать характеристики более простых и общих типов. Наследование обеспечивает как требуемый уровень общности, так и необходимую специализацию.

Наследование

- Объект производного класса является объектом базового класса.

  Наследование - это процесс добавления полей данных и методов-членов.

  Класс-наследник может добавлять свои поля и функции или переопределять функции базового класса.

НАСЛЕДОВАНИЕ. Пример 1

```
class employee
{
public:
 employee (char *, char *, float);
~ employee ();
 void show_employee(void);
    private:
       char name[64];
       char position[64];
       float salary;
    };
```

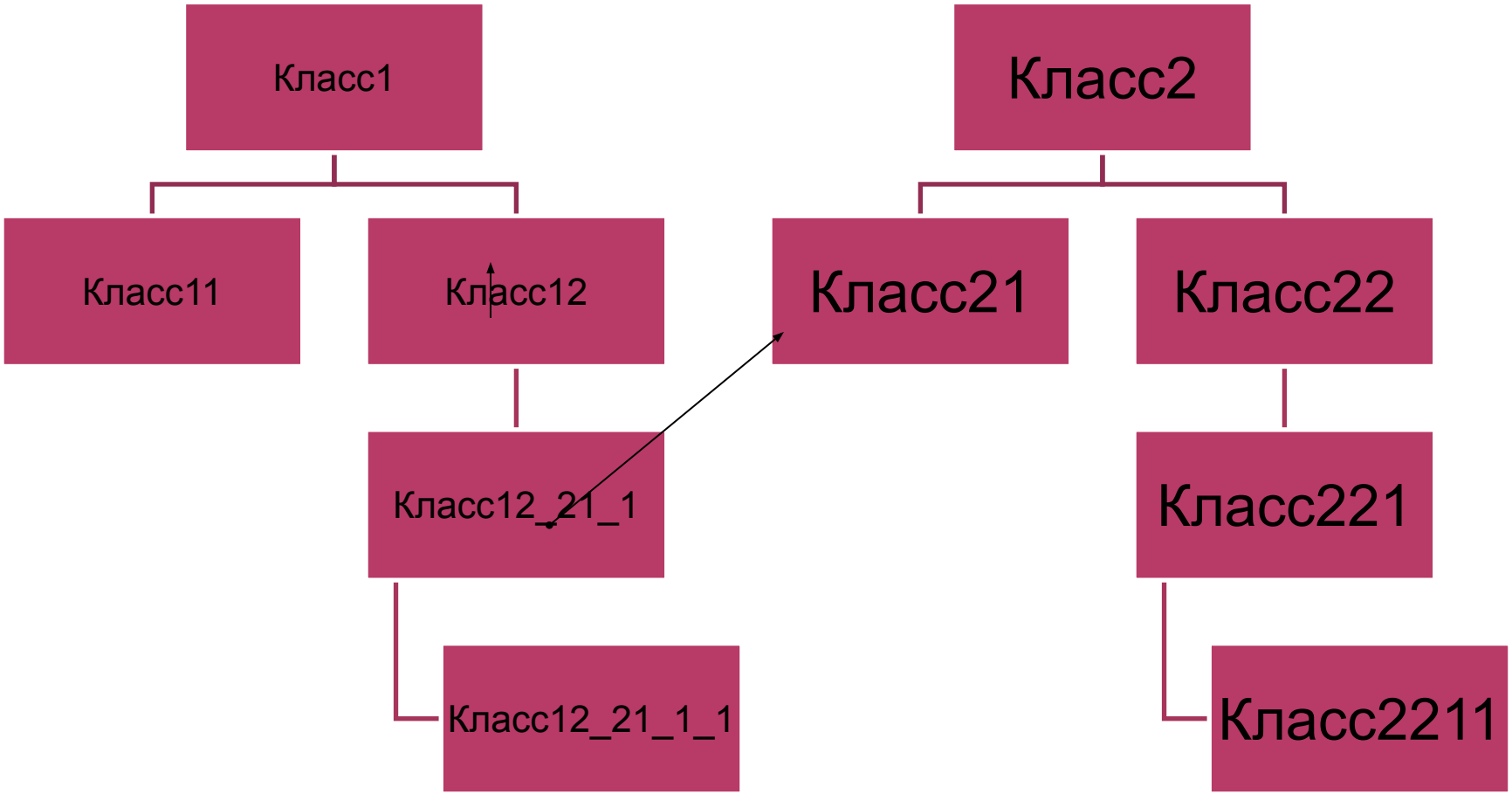
```
class manager : public employee
{
    public:
       manager (char *, char *, char
*, float, float, int);
    ~ manager ();
       void show_manager(void);

    private:
       float annual_bonus;
       char company_car[64];
       int stock_options;
    };
```

Множественное наследование

```
class MyClass1
{…};
class MyClass2
{…};
class MyClass12: public MyClass1, public MyClass2
{…}
```

ПОСТРОЕНИЕ ИЕРАРХИИ КЛАССОВ



три основных свойства ООП

- **Полиморфизм** - способность объекта изменять форму во время выполнения программы.
  - Можно иметь несколько версий одной и той же функции - переопределение наследником функций-членов базового класса

Статический полиморфизм (ранее связывание)

```
class Figure
 {... void Draw(); ... };

class Square : public Figure
{ ... void Draw(); ... };

class Circle : public Figure
{ ... void Draw(); ... };

Circle::Draw() {…}
Square::Draw() {…}
Figure::Draw() {…}
```

```
//Во время компиляции
определяется вызов  функции
```

```
Circle *c = new Circle(0,0,5);
Figure *f = c;
c->Draw();
f->Draw();
```

```
// Указатели друг другу равны, но
    для f будет вызвана другая
    функция, чем для c, поскольку
    f — указатель на объект класса
    Figure.
```

Динамический полиморфизм

- Вызываемая функция определяется во время выполнения.
- Функции-члены должны быть **виртуальными:** *virtual* перед именем .
- Любой производный от базового класс может использовать или перегружать виртуальные функции.
- Для создания полиморфного объекта следует использовать указатель на объект базового класса.

Динамический полиморфизм (позднее связывание)

```
class Figure
{ ...
 virtual void Draw();
 ... };

class Square : public Figure
{ ...
virtual void Draw() ;
 ... };

 class Circle : public Figure
{ ...
virtual void Draw();
 ... };
```

```
Figure * figures[10];

figures[0] = new Square(1, 2, 10);
figures[1] = new Circle(3, 5, 8);
 ...

for (int i = 0; i < 10; i++)
    figures[i]->Draw();
```

Шаблон класса

задает образец определений семейства классов. Тип данных, которым оперирует класс, указывается в качестве параметра при создании объекта, принадлежащего к этому классу.

Объявление параметризованного класса:

```
template <class тип_данных> class имя_класса
{ . . . };

template <class T1> class MyClass
{
 MyClass (int size);
 . . . };
```

Объявление объектов, основанных на шаблоне класса

template_class_name <typel, type2> object_name (parameter1, parameter2);

Пример:

template <class T1> class MyClass

{ MyClass (int size); . . . };

MyClass <char > Small_caps (100) ;

несколько параметров

- несколько параметров-типов

```
template <class T1, class T2, class T3>
        class Container;
```

- параметр-тип и параметр-константа

```
template <class Type, int size>
        class Buffer;
```

Основные свойства шаблонов классов

- Шаблоны могут быть производными (наследоваться) как от шаблонов, так и от обычных классов.
- Шаблоны могут использоваться в качестве «родителей» для других шаблонов или классов.

Пример 2

```
template <class Type> class Queue
{ public:
    Queue();
    ~Queue();
    Type& remove();
    void add( const Type & );
    bool is_empty();
    bool is_full();
private:
     ... };
Queue<int> qi;
Queue<string> qs;
```

Определение деструктора вне шаблона

```
template < class Type> class Queue
      { public:
      Queue()
      ~Queue();
      Type& remove();
      void add( const Type & );
      bool is_empty()
      …}

template < class Type> Queue< Type> ::~Queue()
{
while (! is_empty() ) remove();
}
```

Пример 3.1:

```
template<class T, class T1> class array
{
public:
   array(int size);
   T1 sum (void);
   T mean(void);
   void show_array(void);
   int add_value(T);
private:
   T *data;
   int size;
   int index;
};
```

Пример 3.2:

```
template<class T, class T1>
        T array<T, T1>::mean()
{
    T1 sum = 0;
    int i;
    for (i = 0; i < index; i++)
        sum += data[i] ;
    return (sum / index);
  }
```

Пример 3.3:

```
template<class T, class T1>
    T array<T, T1>::add_value(T value)
{
   if (index == size)
   return(-1); // Массив полон
   else
 {
      data[index] = value;
      index++;
      return(0); // Успешно
   }
}
```

Пример 3.4:

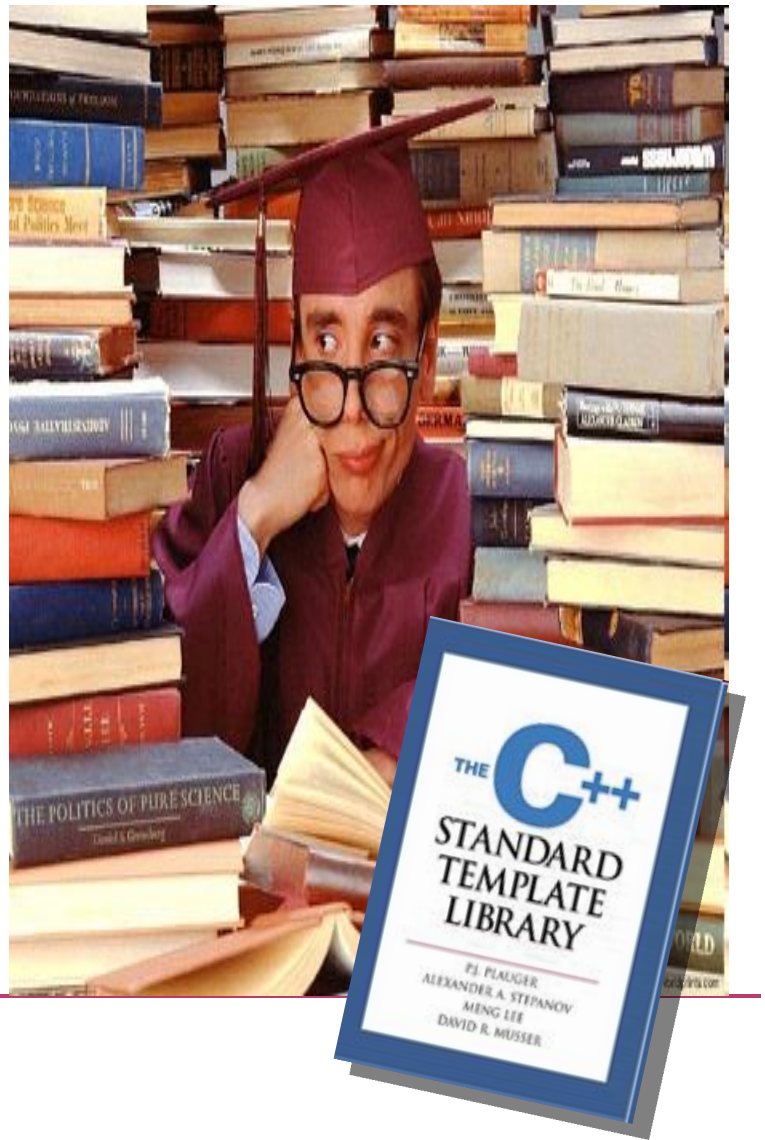
```
void main(void)  {
array<int, long> numbers(100); // Массив 1
array<float, float> values(200); // Массив 2
 int i;
 for (i = 0; i < 50; i++) numbers.add_value(i);
cout << «mean1" << numbers. mean () << endl;

for (i = 0; i < 100; i++) values.add_value(i *100);
 cout << “mean 2" << values. mean() << endl;
     }
```

БИБЛИОТЕКА STL

- STANDARD TEMPLATE LIBRARY
  стандартная библиотека шаблонов

- экономия времени на разработку и сопровождение программ, эффективность и более высокая надежность этих программ, более легкая стыковка кода



библиотека C++ STL (standard templates library)

- набор готовых шаблонов (для типов данных)
- контейнер (container): управляет набором объектов в памяти (вектора, списки, стеки, очереди - заготовки)
- итератор (iterator): обеспечивает для алгоритма средство доступа к содержимому контейнера
- функциональный объект (function object): инкапсулирует функцию в объекте (напр. сортировка)
- алгоритм (algorithm): определяет вычислительную процедуру
- адаптер (adaptor): шаблонные классы, которые обеспечивают отображения интерфейса

КОНТЕЙНЕРЫ

- **Последовательные**
- (***линейный список***)

- list
- vector
- deque
- …

Ассоциативные

(***ключ – значение***)

- map
- multimap
- set
- …

Последовательные контейнеры

- **vector -** массив с произвольным доступом,
- **list -** список
- **deque -** дек;
- **stack -** стек
- **queue -** очередь
- **priority queue -** то же, что и queue, но может сортировать данные по приоритету.

Контейнеры и итераторы

- Итератор – механизм доступа к определенному элементу коллекции (как указатель)
- Имея итератор p какого-то контейнера, можно перейти к следующему или предыдущему элементу (++p, --p), получить элемент (*p), сравнить с другим итератором того же контейнера (p==p1).
- Итераторы контейнеров реализованы в виде шаблонов, причем они являются частью определения класса самого контейнера

Итераторы

- константные и
- обычные,
- прямые,
- обратные,
- двунаправленные,
- с произвольным доступом.

VECTOR – динамический массив

- **size() – возвращает текущий размер вектора**
- **begin() – возвращает итератор, установленный на начало вектора**
- **end() - возвращает итератор, установленный на конец вектора**
- **push_back() – записывает значение в конец вектора**
- **insert() – записывает значение непосредственно перед элементом, на который ссылается итератор**
- **erase() – удаляет элемент из вектора**
- **…**

Пример использования вектора

```
#include <vector>
…
vector<char> v;                    //пустой вектор
int i;                             //счетчик
for (i=0; i<10; i++) v[i] = i + 'a'; //заполнение
//устанавливаем итератор в конец
vector<char>::iterator p=v.end();
v.insert (p,3,'X');    //вставляем три символа X в вектор
//выводим на экран содержимое вектора
for (i=0; i<v.size(); i++) cout<< v[i] << " ";
```

на экране:

a b c d e f g h i j X X X

пример, использующий итераторы для перебора элементов вектора

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

main()
 {
// создаем вектор
  vector<int> v(10);

  for (int i=0; i<10; i++)
        v[i] = i;
// Создаем копию
    vector<int> res = v;
 // итератор
vector<int>::iterator p;
    for
    (p=res.begin();p!=res.end();p++)
    *p *= 2;

    for
    (p=res.begin();p!=res.end();p++)  cout <<
    *p << endl;


        return 0;

        }
```

List – двусвязный список

front	предоставляет доступ к первому элементу
back	предоставляет доступ к последнему элементу
begin cbegin	возвращает итератор на первый элемент
end cend	возвращает итератор на элемент, следующий за последним
empty	проверяет отсутствие элементов в контейнере
size	возвращает количество элементов
max_size	возвращает максимально допустимое количество элементов
clear	удаляет содержимое
insert	вставка элементов

Example of List

```
#include <iostream>
#include <list>
…

main()
{
   list<int> L;
  L.push_back(0);              // Insert a new element at the end
  L.push_front(0);             // Insert a new element at the beginning
   L.insert(++L.begin(),2);    // Insert "2" before position of first argument
                               // (Place before second argument)
  L.push_back(5);
  L.push_back(6);
   list<int>::iterator i;
        for(i=L.begin(); i != L.end(); ++i)
              cout << *i << " ";
    cout << endl;
return 0;
}
```

На консоли: 0 2 0 5 6

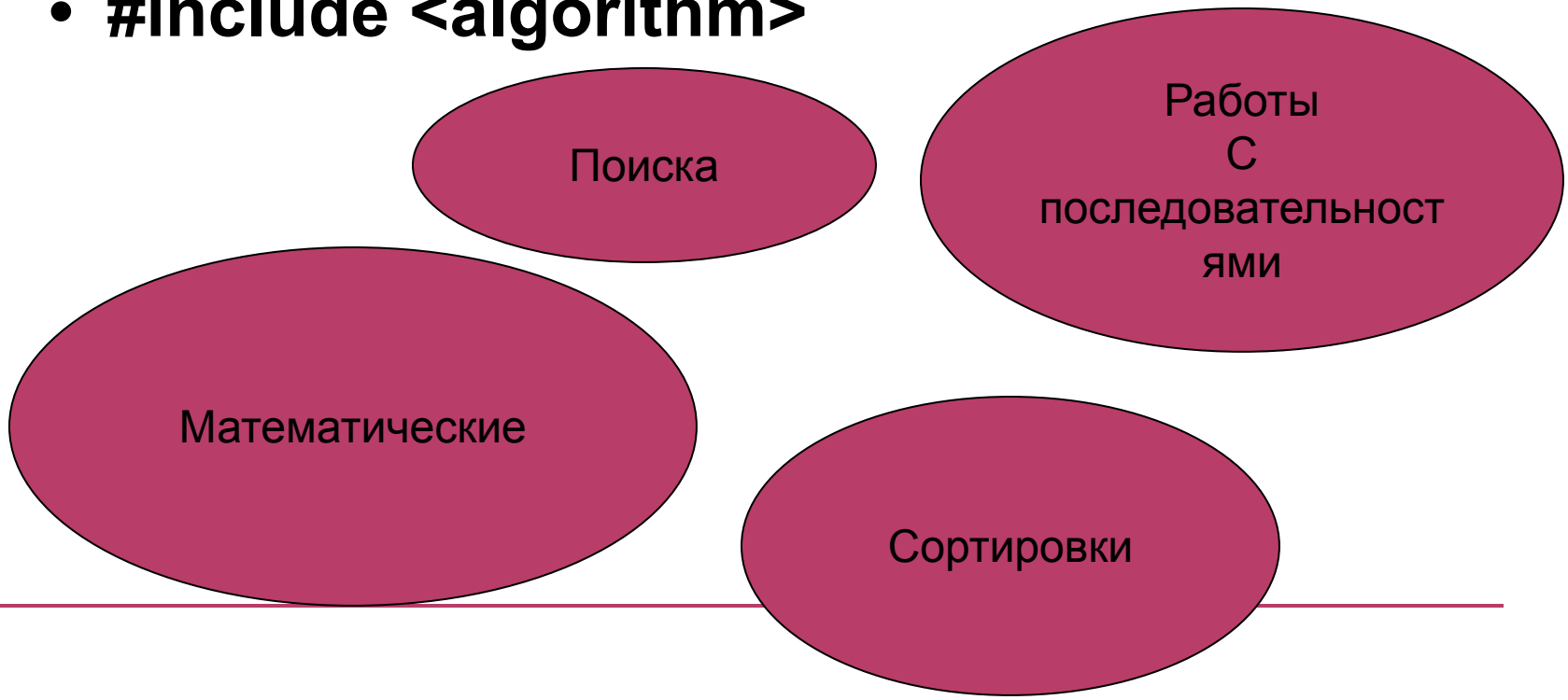
Ассоциативные контейнеры

- обеспечивают быстрый поиск данных, основанных на ключах. Библиотека предоставляет четыре основных вида ассоциативных контейнеров:
  - *set* - множество (уникальные ключи)
  - *multiset* - множество с дубликатами (поддерживает равные ключи)
  - *map* – словарь - ассоциативный контейнер, который поддерживает уникальные ключи (не содержит ключи с одинаковыми значениями) и обеспечивает быстрый поиск значений другого типа *T*, связанных с ключами
  - *multimap* (словарь с дубликатами)

Алгоритмы STL

- набор готовых функций, которые могут быть применены к STL коллекциям и могут быть подразделены на основных группы

  - **#include <algorithm>**



Линейный поиск FIND

- Пример:

- list<int> L;
- L.push_back(3);
- L.push_back(1);
- L.push_back(7);
- list<int>::iterator result = find(L.begin(), L.end(),7);

Быстрая сортировка *SORT*

- int A[ ] = {1, 4, 2, 8, 5, 7};
- const int N = sizeof(A) / sizeof(int);
- sort(A, A + N);
- copy (A, A + N, ostream_iterator<int> (cout, " "));

- // The output is " 1 2 4 5 7 8".

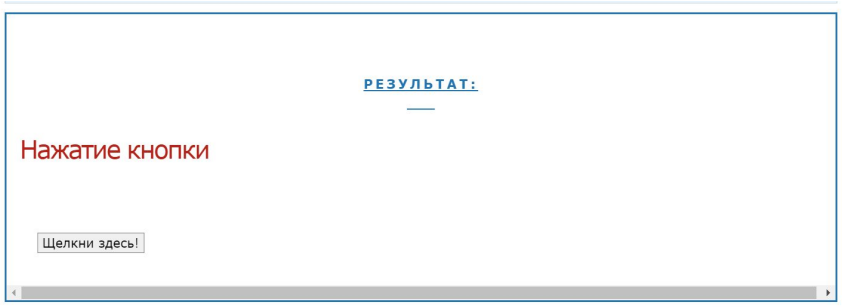
ООП Python. Пример

```
import pandas as pd
import numpy as np
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# reading as pandas frame
FileMeteo="..\\..\\1_Data\\Meteo\\Meteo_Kazan\\27595_2022.xls" D2 = pd.read_excel(FileMeteo,
skiprows=7, usecols=[0,1,3,5,23,24], header=None)
D2[0]=pd.to_datetime(D2[0], dayfirst=True)      # переводим в дату
D2=D2.sort_values(by=0)
# UTC время в GPS сутках с начала года
t3=D2[0].to_numpy().astype(float)/(3600.*24*1e9)-365.25*(2022-1970)-3./24.
fig = plt.figure(1)
plt.subplot(3,1,1)
plt.plot(t1,ZTD1)
plt.plot(t2,ZTD2)
plt.ylabel('ZTD')
plt.xlim(t_begin, t_end)
```

ООП Python. Пример

```
from netCDF4 import Dataset
import matplotlib.pyplot as plt
#_____________________________Reanaliz_____________________________
filename1 ='..\\0_Convection_IWV\\TP_CAPE_CIN\\TP_conv_2022.nc'
Klat=1
Klon=0
nc1 = Dataset(filename1, 'r', Format='NETCDF3_64BIT_OFFSET')
print (nc1.variables)
cape=nc1.variables['cape'][:,Klat,Klon,1]          #Convective available potential energy J kg**-1
#ewss=nc1.variables['ewss'][:,Klat,Klon]           #Eastward turbulent surface stress N m**-2 s
cin=-nc1.variables['cin'][:,Klat,Klon,0]           #Convective inhibition J kg**-1
Year= np.trunc(nc1.variables['time'][100]/24./365.25+1900)
# UTC время в сутках с начала года 01/01==1
t4 = nc1.variables['time'][:]/(24.)+np.trunc((1900-Year)*365.25)+1#+3./24.
#-----------------------------------------------------------------
```

ООП. javascript Пример




```
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html;charset=utf-8">
<script>
<!--
function btnClick()
{
var Txt1 = "";
var Txt2 = "";
Txt1 = document.Test.bt.value;
Txt2 = document.Test.bt.name;
document.getElementById('ex1').innerHTML="<HR>"+
"Вы нажали кнопку: " + Txt1.bold() +
" с именем: " + Txt2.bold() +"<HR>";
}
//-->
</script>
</head>
<body>
<H1>Нажатие кнопки</H1>
<div id="ex1"></div>
<FORM NAME="Test">
<INPUT TYPE="button" NAME="bt" VALUE="Щелкни здесь!"
onClick="btnClick();">
</FORM>
</body>
</html>
```

Литература к лекции

- В. Ольшевский C++
- Б. Страуструп C++
- А.Уваров Visual C++
- Х. Дейтел, П. Дейтел Как программировать на C++
- http://msdn.microsoft.com/ru-ru/library/ct1as7hw%28en-us%29.aspx
- cppreference.com