Тема 1. Введение в C++

Контакты

Валединский Владимир Дмитриевич

сайт с материалами: v-dinsky.info

вопросы, комментарии и т.п. по почте

v-dinsky@yandex.ru

в теме письма пишите "лекции-2"
на это слово будет настроен фильтр,
чтобы письмо не затерялось в массе других

Административные вопросы

В 2021–2022 учебном году курс лекций на механико-математическом факультете МГУ проходит в дистанционном режиме. Здесь содержится конспект сказанного на лекциях. Это позволит слушателям не тратить время на конспектирование, однако пояснения и обсуждения многих моментов проводится именно на лекции, и не всегда отражается в ее конспективной записи. Таким образом, для ознакомления с материалом стоит прослушать именно лекцию, а содержимое данного файла сохранит основные ключевыет моменты.

По учебному плану в этом семестре запланирован экзамен, а зачета нет. Поэтому традиционно уже несколько лет оценка экзамена зависит от практической работы на семинарах. Что получится в этом семестре сказать невозможно, так как очное проведение экзамена остается под большим вопросом, но все же мы попытаемся сохранить общие принципы выставления оценки. Эта оценка складывается их трех слагаемых: работы в семестре, решение практической задачи на экдамене, и ответ на “теоретические вопросы билета”. В какой пропорции эти слагаемые составят окончательную оценку пока тоже не очень понятно (это будет завичеть от того, что мы успеем начитать, как пойдет решение семестровых задач и т.д.), но примерные пропорции могутвыглядеть так: семестр - 2, задача - 2, теория - 1.5, т.е. человек мог получить 5 при небольших дефектах ответа на билет, решения задачи, или работы в семестре. Как это булет в этом семестре, мы сказать пока не в состоянии, но общий принцип сохранится, т.е. работа в семестре будет непосредственно влиять на итоговую оценку.

Программа лекций второго семестра состоит из двух больших разделов: схемы хранения данных и некоторые алгоритмы обработки данных. По сути это квалификационный минимум программиста — знание как представить данные в своих программах и как их обрабатывать в “стандартных” ситуациях.

Первая часть курса посвящена классическим схемам хранения данных таким как стек, очередь, дек, список, дерево, граф, множесво и т.п., а вторая часть знакомит с такими же алгоримами обработки информации (поиск в деревьях, алгоритмы на графов, сжатие данных и т.п.).

Базовым инструментом работы во 2 семесте будет язык C++, так как он дает

необходимые объектно-ориентрированные средства, которые оказываютсяв данном контексте очень удобными.

Мы не будем отводить отдельное время подробному изучению языка, а сразу будем строить требуемые реализации, разъясняя по мере неодходимости используемые концепции и конструкции. Преподагается что каждый может поискать в сети подробные описания необходимых конструкций.

Однако с самого начала все же придется ввести несколько ключевых понятий.

Объект — это некоторая конструкция, которая обладает внутренним состоянием (набором данных) и набором функций, позволяющих это состояние модифицировать. Функции из этого набора могут быть доступны внешнему использованию (public интерфейс), но также могут быть закрыты от внешнего воздействия и использоваться только как внутренние (private) алгоритмы объекта.

Объектно-ориентированная программа создает некоторое количество объектов, устанавливает им начальны состояния, а далее, при помощи взаимодействия объектов приводит их к тому состоянию, котороя является конечной целью. В этом плане, программирование является аналогом управленческой деятельности, когда разноообразные исполнители координируют свои действия для достижения желаемого результата.

В терминах языка C++ объект реализуется через базовое понятие класса.

Класс — это структура, которая содержит некоторый набор данных и также набор методов (функций), которые могут выполнять работу с данными этого класса, преобразуя эти данные и тем самым меняя состояние конкретного экземпляра класса.

Также вводятся понятия прав доступа к данным и методам. Одни операции могут выполняться кем угодно, другие операции могут быть разрешены только экземплярам данного класса или кому-то еще по определенным правилам. Разграничение прав доступа составляет одну из основных концепций языка C++.

Мы не будем здесь приводить исчерпывающее формальное изложение всех правил языка C++, а ограничимся лишь некоторыми примерами, помогающими понять суть. Для более подробной информации можно обратиться к формальным описаниям языка и справочным пособиям.

Простейший пример — Vector

Рассмотрим формализм введения классов в C++ на примере вектора.

Многие конструкции могут быть записаны в разных формах, поэтому надо иметь в виду, что дальнейшие примеры не являются единственно возможным вариантом, и об этом даны комментарии в устной лекции.

Пусть мы хотим записать решение некоторой геометрической задачи с использованием векторов a,b,c,d и некоторыми естественными операциями с ними. Например,

```
int main()
{
    double s, r;
    Vector a, b, c, d;

    a = b + c + 3*d;  // линейная комбинация векторов
    r = a*b;          // скалярное произведение
```

```
   return 0;
}
```

Т.е. нам надо определить класс Vector и задать ему операции между векторами, между векторами и числами и т.п.

Определение класса может использоваться в нескольких файлах, поэтому, обычно его помещают в заголовочный файл. В нашем случае назовем его Vector.h.

После некоторых размышлений и принятия решений по поводу функциональности, описание класса может принять примерно такой вид.

```
// файл Vector.h
#include <stdio.h>

class Vector
{
 private:
   double x,y,z; // координаты вектора
 public:
   // конструкторы
   Vector();
   Vector(double xx, double yy, double zz);
   Vector(const Vector &a);        // конструктор копирования
   ~Vector() = default;            // деструктор

   // операции, определяемые через дружественные функции
   friend double operator*(const Vector &a, const Vector &b);
   friend Vector operator+(const Vector &a, const Vector &b);
   friend Vector operator*(double c, const Vector &a);

   // операции, определяемые через методы класса
   const Vector & operator=(const Vector &b);

   // распечатка вектора
   void Print(FILE *f);
};
```

Конструкторы класса отвечают за то в какой состоянии будет создаваться класс при его объявлении в программе.

Деструктор отвечает за уничтожении класса. По сути деструктор должен освобождать ресурсы, захваченные классом во время его жизни. В тривиальныех случаях (как здесь) можно довериться деструктору по умолчанию.

конец лекции 1

В описании класс только объявлен, а реалиация его методов обычно размещается также в отдельном файле. Назовем этот файл Vector.cpp.

```
// файл Vector.cpp
#include "Vector.h"
```

```
// реализация класса Vector и его friend функций

Vector::Vector()
{
   x = y = z = 0;
}
Vector::Vector(double xx, double yy, double zz)
{
   x = xx; y = yy; z = zz;
}
Vector::Vector(const Vector &v)
{
   x = v.x; y = v.y; z = v.z;
}
const Vector & Vector::operator=(const Vector &b)
{
   x = b.x;  y = b.y; z = b.z;
   return *this;
}
void Vector::Print(FILE *f)
{
   fprintf(f, "%f  %f  %f\n", x, y, z);
}

double operator*(const Vector &a, const Vector &b)
{
    return a.x*b.x + a.y*b.y + a.z*b.z;
}
Vector operator+(const Vector &a, const Vector &b)
{
     return Vector(a.x+b.x, a.y+b.y, a.z+b.z);
}
Vector operator*(double c, const Vector &a)
{
    return Vector(c*a.x, c*a.y, c*a.z);
}
```

Теперь для проверки нашей реализации можно составить работоспособную программу.

```
// файл VecProg.cpp
#include "Vector.h"

// демонстрационный пример использования класса Vector

int main()
{
```

```
    double s, r;
    Vector a, b;
    Vector c(1.0, 2.0, 4.0), d(5, 3, -7);

    a = c + 3*d;
    a.Print(stdout);
    b = a + Vector(1,2,3);
    b.Print(stdout);

    r = a*b;
    s = operator*(a,b);
    fprintf(stdout,"scal prod %f %f\n", r, s);

    return 0;
}
```

Что осталось недосказанным, или краткий итог.

Определение класса может быть записано разными способами. Пока нет полного понимания, что должно присутствовать обязательно, а что можно переложить на плечи компилятора, условимся, что определение класса должно содержать:

— конструктор без параметров (можно по умолчанию =default)
— другие конструкторы с параметрами (если надо по смыслу задачи)
— конструктор копирования
— деструктор (можно по умолчанию =default)
— методы класса (по смыслу задачи)
— внутренние переменные класса нужно как правило размещать в секции private

Определение операций (перегрузка операций) может реализовываться либо дружественными функциями, либо в некоторых случаях как методы класса. В последнем случае первым аргументом операции является класс, от имени которого вызывается данный метод.

Операция может быть вызвана не только записью ее знака вместе с аргументами, но так же как и функция класса (например `a+b` или `a.operator+(b)`).

Аргументы и возвращаемые значения целесообразно задавать как (константные) ссылки с тем, чтобы лишний раз не задействовать конструктор копирования. При этом надо понимать, что возвращать ссылку можно только на объект, существующий вне контекста данной функции, поскольку локальные объекты (переменные) функции перестают существовать после завершения функции, и ссылка на них уже не будет иметь смысла.

Более подробную информацию о перегрузке операций и прочих рассмотренных вопросах можно получить из справочной литературы или поиском в сети.

Как упражнение по данной лекции предлагается доработать класс Vector, внеся в него множество других операций и методов, чтобы получить возможность работать с векторами “как с числами”.

конец лекции 2

Нововведения и отличия от C

Язык C++ развивается, есть несколько принципиальных версий. Одна из наиболее "революционных" — C++ 11, но нам надо до нее дорасти. Поэтому пока в рамках "классических" версий.

Перечислим основные отличия и нововведения в языке C++ по сравнению с C.

- понятие класса и все с этим связанное
  - private и public доступ к членам класса и методам
  - конструкторы и деструкторы
  - перегрузка операций
  - указатель this
  - статические члены класса
  - const методы
  - наследование и виртуальные функции [пока не рассматриваем]
- struct есть `class {public: ... };`
- объявление объектов в любом месте кода
- ссылочный тип данных — создание "синонима" имени объекта обращение по имени, но работа как по указателю
- полиморфизм функций — функции могут имет одинаковые имена, но должны иметь разные сигнатуры
- значения параметров по умолчанию `int fun(double a = 3.14, int b = 0);`
- пространства имен namespace
- управление памятью `new delete` ("вызовы" конструктора и деструктора)
- строгая типизация указателей, различные способы преобразования указателей static_cast, dynamic_cast, const_cast, reinterpret_cast
- исключения, try-catch блоки
- механизм шаблонов template
- потоковый ввод-вывод, прегрузка операторов ввода-вывода для классов

Некоторые понятия рассматривались на предыдущей лекции. Некоторые мы здесь кратко поясним, некоторые оставим то тех пор, пока они не будут реально нужны.

Статические члены класса. static
— устанавливает зону видимости в рамках файла static int w;
— сохранение и первичная инициализация в блоке static int v; int d; .....
— общий элемент для всех экземпляров класса

```
class A                              A  a, b, c;
{                                    a.x = 1;                a.fun();
 public:                             b.x = 2;                A::fun();
   int x;                            c.x = 3;
   static int y;                     A::y = 6;
   static void fun() { y = 5; }
};                                   b.y = 7;
                                     a.y = 7;
```

`int A::y;` - по сути обычная переменная, но только "привязянная" к классу

const методы и вообще о константных объектах.

```
double a;
int i, j;
char str[20];

#define  PI  3.1415926535
const double pi = 3.1415926535;
const double ccc = f(.......);
         a = pi;  // можно
         pi = 2;  // нельзя!
         a = PI;

const int *p;     // или    int const *p;
         p = &i;  // можно
         j = *p;  // можно
         *p = j;  // нельзя!

int * const p = &i;
         p = &j;  // нельзя!
         j = *p;  // можно
        *p = j;   //  можно

const char * s = "the text";
         printf("%s\n", s);  // можно
         s[4] = '_';            // нельзя!
         s = str;               // можно

const char * cosnt s = "the text";

struct A                                          void fun(const A & r)
{                                                 {
   int x;                                            int z = r.x;
   A(int xx) { x = xx; }                             //r.x = 3;
```

```
   int Value() const { return x; }                    z = r.Value();
   int & lValue() { return x; }                       z = r.lValue();
   const int & lValue() const { return x; }           r.lValue() = 25;
};                                                  }
```

Ссылочный тип данных.

```
int x;
int &rx = x;      rx - синоним для x

void fval(int a)        void fptr(int *p)     void fref(int &r)
{                       {                     {
   a = 1;                  *p = 2;               r = 3;
}                       }                     }


int s = 0;
fval(s);          // s есть 0
fptr(&s);         // s есть 2
fref(s);          // s есть 3

void fcref(const int &r)
{
   int b = r;  // можно
   r = 3;      // нельзя
}
```

Полиморфизм функций и параметры по умолчанию

```
                                                      fun(3.0);
   int fun();
   int fun(int x);
   int fun(double x);
   int fun(int x, double y);                       fun(1.0, 2);  ???
   int fun(double x, double y);

   double fun(int x);   !!!!!!!!!

   int fun(double x = 1, double y = 2);

       fun(x, y);
       fun(x);       fun(x, 2.0);
       fun();        fun(1.0, 2.0);

       void print(FILE * f = stdout);
```

```
print();
print(my_file_ptr);
```

Пространства имен namespace

```
namespace ABC
{
    int r;
    ... прочий код ...
}

double r;
r = 1;
ABC::r = 1;
```

```
ABC::r = 1;
```

```
using namespace ABC;
r = 1;
```

```
using namespace std;
```

Управление памятью new delete
("вызовы" конструктора и деструктора)

```
void * malloc(size_t num_of_bytes);
void   free(void * ptr);

new <конструктор>                                  Vector *p = new Vector(1,2,3);
new <конструктор без параметров> [количество]      Vector *q = new Vector [128];
delete указатель                                   delete p;
delete [] указатель                                delete [] q;

при отказе - исключение std::bad_alloc
```

конец лекции 3

Исключения, try-catch блоки.

Сигнал, "распространяющийся" вверх по последовательности вызовов функций и, возможно, несущий с собой некоторое значение (экземпляр класса).

Исключение вызывается (throw) явно в коде (своем или библиотечном) или операционной системой в процессе выполнения (run time exception).

Этот сигнал можно перехватить try-catch блоком.

```
if (возникла ошибка) throw  25;
if (другая ошибка) throw  26;
```

```
try {
  .....
} catch (int x) {
```

```
int main()
{
  try
```

```
   if (x==25) ....       {
   if (x==26) ....          ......
}                        } catch(...) {
                         {
                       }
```

```
class MyException {....};
if (....) throw new MyException();
```

```
try {
  .....
} catch (MyException * p) {
    cout << p->Code() << std::endl;
    cout <<  p->Message();
    delete p;
    exit(-1);
}
```

Механизм шаблонов template.

```
template<class T>
T abs(T x)
{
   return (x>=0) ? x : -x;

}
```

```
int x, ax;       ax = abs(x);
short y, ay;     ay = abs(y);
double z, az;    az = abs(z);
char * s, *as;  // as = abs(s);

          az = abs<double>(-3);
```

Точно также можно параметризовать описания и определения классов — об этом совсем скоро ...