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

По учебному плану в этом семестре запланирован экзамен, а зачета нет. Поэтому традиционно уже несколько лет оценка экзамена зависит от практической работы на семинарах. Общие принципы выставления оценки и процедура экзамена описаны на сайте на отдельной странице, поэтому здесь об этом подробно не пишем.

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, множество и т.п., а вторая часть знакомит с такими же алгоримами обработки информации (поиск в деревьях, алгоритмы на графах, сжатие данных и т.п.).

Базовым инструментом работы во 2 семесте будет язык C++, так как он дает необходимые объектно-ориентрированные средства, которые оказываются в данном контексте очень удобными. Также мы попытаемся познакомиться со "стандартными" библиотеками для хранения и манипуляций с данными, например, STL.

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

Язык C является типичным представителем процедурных языков программирования, т.е. реализует идеологию (парадигму) вычисления требуемого результата, исходя из заданного набора данных при помощи некоторого небора процедур.

данные — процедуры — результат

Достоинства — хорошо соотносится с традициоными математическими (инженерными) задачами, естественно реализуется в вычислительных системах.

Недостатки — неудобен, когда результат решения задечи не есть "отдельная сущность" (число, массив и т.п.), т.е. когда задача охватывает много различных "сущностей" — взаимосвязей между данными.

C++ поддерживает объектно-ориентированную идеологию, когда сущностью является "объект" с его внутренним "состоянием", а программа описывает взаимодействия объектов и изменения их состояний. Результатом является уже набор объектов, получивших требуемые состояния.

Язык C++ развивается, за время его жизни было выпушенно несколько стандартихованных версий. Одна из наиболее "революционных" версий — C++ 11, мы будем

в основном ориентироваться на нее, но до этого еще надо дорасти. Последующие версии также вводят в язык новые конструкции и возможности, но на данном этапе они просто не будут понятны.

Объекты в C++

С самого начала 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 начальные состояния, а далее, при помощи взаимодействия объектов приводит их к тому состоянию, которое является конечной целью. В этом плане, программирование является аналогом управленческой деятельности, когда разнообразные 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.

Для классов вводятся понятия прав доступа к данным и методам. Одни операции (public) могут выполняться кем угодно, другие операции (private) могут быть разрешены только экземплярам данного класса или кому-то еще в соответствии со специально определенными указаниями. Разграничение прав доступа составляет одну из основных концепций языка C++.

Мы не будем пока 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.

C++, новые понятия и отличия от C

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

- понятие класса и все с этим связанное
  - права доступа к членам класса и методам — private, public, protected.
  - конструкторы и деструкторы
  - указатель this
  - перегрузка операций
  - const методы
  - статические члены класса
  - дружественный friend доступ
  - наследование и виртуальные функции
- 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;
- многоликая и важная инструкция `const`;
- 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)
- исключения, try-catch блоки
- механизм шаблонов template
- строгая типизация указателей, различные способы преобразования указателей static_cast, dynamic_cast, const_cast, reinterpret_cast
- потоковый ввод-вывод, прегрузка операторов ввода-вывода для классов

Простейший пример — Number

Рассмотрим формализм введения классов в C++ на примере “усложненного” числа. Всем известна проблема вычислительной погрешности при выполнении операций с вещественными числами. Однако оценка погрешности результата, которую в итоге даст некоторый конкретный алгоритм, представляет собой иногда весьма нетривиальную задачу. Мы попробуем получить оценку этой погрешности автоматически, вычисляя ее “параллельно” с арифметическими операциями.

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

Итак пусть наше “число” одновременно хранит как значение value, так и его текущую погрешность error, т.е. каждое число можно рассматривать как $x + \varepsilon$, где x — точное значение, а ε — его погрешность. При выполнении арифметических операций, например $(x + \varepsilon_x) \pm (y + \varepsilon_y) = (x + y) \pm (\varepsilon_x + \varepsilon_y)$. погрешность результата может быть легко выражена через погрешности исходных аргументов. Так как погрешности могут иметь случайные и разные знаки, то для оценки максимальной возможной погрешности мы будем считать, что реализуется наихудщий вариантмаксимально вохможной погрешности, т.е. везде ставить модули значений. Так для сложения и вычитания погрешность результата в обоих случаях будет оцениваться через $|\varepsilon_x| + |\varepsilon_y|$ и аналогично для умножения и деления.

конец лекции 1

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

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

```
// файл Number.h
// #include ...

class NumberA {
private:
    double value, error;
public:
    // конструкторы и деструктор
    // присваивания
    // арифметические операции
    //   - как методы класса
    //   - как внешние (дружественные) функции
    // сравнения
    // приведения типов
    ...
    // элементарные функции
```

```
    ...
};
```

Конструкторы класса отвечают за то в каком 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 (=default).

В описании класс только объявляется, там перечисляются внутренние переменные (объекты) и заголовки функций (методов) с указанием прав доступа к ним. Сами реалиация методов обычно размещается также в отдельном файле. Если реализация метода очень простая, то для наглядности ее можно размещать непосредственно при описании класса.

В нашем случае назовем файл с описанием класса Number.h, а с реализацией методов — Number.cpp.

Работоспособный и почти полный пример приведен в архиве number.zip.

Далее разбор и обсуждение конструкций по тексту примера.

Несколько важных моментов.

Конструкторы можно задавать различными способами в зависимости от смысла класса. Вот несколько примеров. Здесь некоторые конструкторы переопределяют друг друга, поэтому в реальном проекте надо оставить только разные по сигнатуре записи. Но мы здесь приводим все записи в качестве иллюстрации.

```
NumberA::NumberA() = default;
NumberA::NumberA() { value = error = 0; }
NumberA::NumberA(double v, double e) { value = v; error = e; }
NumberA::NumberA(double v = 0, double e = 0) { value = v; error = e; }
NumberA::NumberA(double v = 0, double e = 0) : value(v), error(e) {}
```

Другим важным условием является наличие конструктора копирования (важно для передачи класса в функцию по зничению или возврата через return).

```
NumberA::NumberA(const Number &b) : value(b.value), error(b.error) {}
```

Деструктор в данном случае тривиальный, поэтому можем его вообще не объявлять, либо объявить как

```
NumberA::~NumberA() = default;
NumberA::~NumberA() {}
```

Переопределенный оператор может быть вызван двумя способами - как функция со своим специфичским именем, либо как выражение с кратким значком операции. Например, для присваивания имеем

```
const NumberA & NumberA::operator=(const NumberA &b);
вызовы
a.operator=(b);  или   a = b;
```

Заметим, что при отсутствии определений конструктора копирования, присваивания и деструктора компилятор создаст их по своему разумению, а именно как присваивание друг другу всех объявленных членов класса и деструкторов по умолчанию для объявленных элементов. Во многих случаях этого достаточно, и соответственно можно не определять эти функции явно. Однако на начальном этапе обучения настоятельно рекомендуется все же обяъвлять эти функции явно, чтобы не пропустить их, когда они будут необходимы для классов с более сложным внутренним устройством (когда за членами класса стоят дополнительно захвачнные ресурсы).

Существует так называемое **правило трех** — если программист явно определяет хотя бы одну из функций — конструктор копирования, присваивание, деструктор, то он должен определяить явно и остальные, поскольку если для одной из функций не зватает дефолтной функциональности, то скорее всего ее не хватает и для оставшихся.

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.

Аргументы и возвращаемые значения целесообразно задавать как (константные) ссылки с тем, чтобы лишний раз не задействовать конструктор копирования для всего класса (т.е. у нас передается по значению только ссылка — э это по сути адрес — 8 байт). 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.