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 прошлом учебном году курс лекций на механико-математическом факультете МГУ проходил в дистанционном режиме. В этом году он начинается очно. На это и будем пока ориентироваться. Здесь я буду выкладывать планы-конспекты лекций, стараясь делать это заранее (скажем, накануне). Это позволит слушателям заранее понять очем будет идти речь не также сократить время на конспектирование, однако пояснения и обсуждения многих моментов будет проводиться именно на лекции и не всегда может быть отражено 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. 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, 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. 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.

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 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.

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 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 — 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 погрешность исходных аргументов. Так как погрешности могут иметь случайные и разные знаки, то для оценки максимальной возможной погрешности мы будем считать, что реализуется наихудщий вариант, 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.

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, описание класса может принять примерно такой вид.

```
// файл Number.h
// #include ...

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

Права доступа — private, public, protected.

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.

В описании класс только объявлен, а реалиация его методов обычно размещается также в отдельном файле. Назовем этот файл Number.cpp. Если реализация метода очень простая, то можно ее размещять непосредственно при описании класса.

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

Далее разбор и обсуждение конструкций по тексту примера.

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