Предисловие

Мир построен на C++ (и C — подмножество его).
— Герб Саттер

Инфраструктуры Google, Amazon и Facebook в значительной степени построены на C++. Кроме того, на C++ реализована значительная часть лежащих в их основе технологий. В области телекоммуникаций почти все подключения стационарных и сотовых телефонов управляются с помощью программного обеспечения, написанного на C++. Что еще более важно, все основные узлы передачи в Германии также обрабатываются с помощью C++, а это означает, что мир в семье автора безоговорочно полагается на программное обеспечение, написанное на C++.

Даже программы, написанные на других языках программирования, зависят от C++, поскольку именно на C++ реализованы самые популярные компиляторы — Visual Studio, `clang`, новейшие части Gnu и компилятор Intel. Тем более это верно для программного обеспечения, работающего в Windows, которая также реализована на C++ (как и пакет Office). Это вездесущий язык; даже ваш мобильный телефон и ваш автомобиль обязательно содержат компоненты, управляемые C++. Его изобретатель, Бьярне Страуструп, создал веб-страницу, с которой и взято большинство приведенных здесь примеров.

В области науки и техники многие пакеты высококачественного программного обеспечения реализованы на C++. Сила этого языка проявляется в особенности тогда, когда размеры проектов превышают некоторый определенный размер и когда используемые структуры данных становятся достаточно сложными. Не удивительно, что сегодня многие — если не все — моделирующие программы в области науки и техники реализуются на C++. Abaqus, deal.II, FEniCS, OpenFOAM — только некоторые из известных названий; то же самое можно сказать и о таком ведущем программном обеспечении в области CAD, как CATIA. Благодаря более мощным процессорам и улучшенным компиляторам (в которых могут использоваться не все современные возможности и библиотеки), на C++ все чаще реализуются даже встраиваемые системы. Наконец, мы не знаем, какое количество проектов было бы реализовано на C++, а не на C, начнись они немного позже. Например, хороший друг автора Мэтт Книпли (Matt Knepley), являющийся соавтором весьма успешной научной библиотеки PETSc, как-то раз признался, что если бы это было возможно, сегодня он переписал бы свою библиотеку на C++.

Причины для изучения C++

Как никакой иной язык, C++ охватывает весь спектр задач от программирования на уровне, достаточно близком к аппаратному обеспечению, на одном конце и до абстрактного программирования высокого уровня на другом.

Программирование низкого уровня — типа пользовательского управления памятью — позволяет разобраться, что в действительности происходит во время выполнения, а это, в свою очередь, помогает понять поведение программ на других языках. В C++ с минимальными усилиями можно написать чрезвычайно эффективные программы, производительность которых лишь незначительно ниже производительности выполнимого кода, написанного на машинном языке. Однако не стоит торопиться, пытаясь достичь максимальной производительности; сначала лучше сосредоточиться на написании ясных и выразительных программ.

Здесь в игру вступают высокоуровневые возможности C++. Язык непосредственно поддерживает широкий спектр программных парадигм, и среди прочего — объектно-ориентированное программирование (глава 6, "Объектно-ориентированное программирование"), обобщенное программирование (глава 3, "Обобщенное программирование"), метапрограммирование (глава 5, "Метапрограммирование"), параллельное программирование (раздел 4.6) и процедурное программирование (раздел 1.5).

Ряд методов и идиом программирования — таких как RAII (раздел 2.4.2.1) или шаблоны (раздел 5.3) — были изобретены в C++ и для применения C++. Язык программирования C++ настолько выразителен, что зачастую можно разработать подобные новые методы без внесения изменений в сам язык. И как знать, может быть, в один прекрасный день именно вы изобретете новые методы программирования?

Причины для чтения данной книги

Материал этой книги проверен на реальных людях. Автор три года (три раза по два семестра) читал своим студентам курс "C++ для ученых". В конце такого курса студенты (главным образом студенты математического факультета, но присутствовали и студенты физического и некоторых технических факультетов), часто до прохождения курса не имевшие о C++ никакого понятия, в конце курса вполне могли использовать такие "продвинутые" средства программирования, как шаблоны. Вы можете читать эту книгу в собственном темпе и направлении, следуя по основному пути от главы к главе или уделяя большее внимание дополнительным примерам и справочной информации из приложения A, "Скучные детали".

Красавица и чудовище

Программы на C++ могут быть написаны многими способами. В этой книге мы постепенно познакомим вас со все более и более сложными стилями программирования. Это требует использования все более и более сложных возможностей языка программирования, которые, на первый взгляд, могут показаться пугающими, но как только вы к ним привыкнете, станут простыми и понятными. Фактически программирование на высоком уровне не только применимо для широкого диапазона задач, но и обычно не менее, если не более, эффективно и удобочитаемо.

Чтобы получить первое впечатление, мы проиллюстрируем сказанное на простом примере — методе градиентного спуска с постоянным шагом. Принцип очень прост: вычисляем наиболее крутой спуск $f(x)$ с использованием градиента, скажем, $g(x)$ и следуем в этом направлении с шагами фиксированного размера до следующего локального минимума. Алгоритмической псевдокод так же прост, как и это описание.

Алгоритм 1. Алгоритм градиентного спуска

Вход: начальное значение x, размер шага s, критерий остановки ε, функция f, градиент g
Выход: локальный минимум x

1 **do**
2 | $x = x - s \cdot g(x)$
3 **while** $|\Delta f(x)| \geq \varepsilon$;

Мы напишем две простые реализации для этого простого алгоритма. Взгляните на них, не пытаясь пока что вникать в технические детали.

```
void gradient_descent ( double * x,
   double * y, double s, double eps,
   double  (*f)(double, double),
   double (*gx)(double, double),
   double (*gy)(double, double))
{
   double val = f(*x, *y), delta;
   do {
      *x -= s * gx(*x, *y);
      *y -= s * gy(*x, *y);
      double new_val = f(*x, *y);
      delta = abs(new_val - val);
      val= new_val;
   } while(delta > eps);
}
```

```
template <typename Value, typename P1,
        typename P2, typename F,
        typename G>
Value gradient_descent(Value x, P1 s,
                  P2 eps, F f, G g)
{
   auto val = f(x), delta = val;
   do {
      x -= s * g(x);
      auto new_val = f(x);
      delta = abs(new_val - val);
      val = new_val;
   } while(delta > eps);
   return x;
}
```

На первый взгляд, они очень похожи, и мы расскажем, какой нам нравится больше. Первая версия в принципе представляет собой чистый C, т.е. этот код компилируется и с помощью компилятора C. Преимущество заключается в непосредственной видимости оптимизации. Перед нами двумерная функция со значениями типа `double`. Мы предпочитаем вторую версию как более широко применимую — для функций произвольной размерности с произвольными типами значений. Как ни удивительно, такая универсальная реализация оказывается не менее эффективной. Более того, функции `F` и `G` могут быть встраиваемыми (см. раздел 1.5.3), так что экономятся накладные расходы на их вызовы, в то время как явное использование (уродливых) указателей на функции в левой версии затрудняет применение этой оптимизации.

Более длинный пример сравнения старого и нового стилей терпеливый читатель может найти в приложении A, “Скучные детали” (раздел A.1). В нем применение современных методов программирования куда более очевидно, чем в этом игрушечном примере. Но не будем задерживать ваше знакомство с книгой такими предварительными примерами.

Языки в науке и технике

Было бы хорошо, если бы все численные программы могли быть написаны на C++ без потери эффективности, но если только вы не найдете чего-то, что позволяет достичь этой цели без ущерба для системы типов C++, то лучше уж воспользоваться языком Fortran, ассемблером или архитектурно-зависимыми расширениями.
— Бьярне Страуструп

Научные и инженерные программы пишутся на разных языках программирования, и какой именно из них является наиболее приемлемым, как и везде, зависит от целей и имеющихся ресурсов.

Математический инструментарий, такой как MATLAB, Mathematica или R, потрясающ, если можно использовать имеющиеся в них алгоритмы. Реализуя собственные алгоритмы с мелкими (например, скалярными) операциями, мы можем получить значительное снижение производительности. Это может не быть проблемой, например, для небольших задач (или бесконечно терпеливого пользователя); в противном случае необходимо рассмотреть альтернативные языки.

Python отлично подходит для быстрой разработки и уже содержит научные библиотеки, такие как “scipy” и “numpy”; и приложения, основанные на этих библиотеках (зачастую реализованных на C и C++), оказываются достаточно эффективными. Но вновь определяемые пользователем алгоритмы с “мелкозернистыми” операциями приводят к снижению производительности. Python является отличным средством эффективной реализации задач малого и среднего размеров. Когда проекты вырастают и становятся достаточно большими, все более важной становится строгость компилятора (например, запрещение присваивания при несовпадении аргументов).

Fortran также является отличным выбором, если мы можем опираться на существующие, хорошо отлаженные операции, например на операции с плотными матрицами. Он хорошо подходит для выполнения домашних заданий, которые задает старый профессор (потому что он задает только то, с чем легко справится с помощью Fortran). По опыту автора, добавление новой структуры данных оказывается весьма громоздким, а написание большой моделирующей программы на Fortran является довольно сложной задачей, на которую сегодня решается постоянно уменьшающееся меньшинство исследователей.

C обеспечивает хорошую производительность, и кроме того, на C написано большое количество программ. Язык этот относительно небольшой и прост в изучении. Проблема заключается в написании без ошибок больших программ, использующих простые и опасные возможности языка, в особенности указатели (раздел 1.8.2) и макросы (раздел 1.9.2.1).

Языки наподобие C#, Java и PHP, вероятно, являются хорошим выбором, если основная сфера применения приложения — веб или обеспечение графического интерфейса при не слишком большом количестве вычислений.

C++ выделяется среди других языков программирования в особенности тогда, когда мы разрабатываем большие, высококачественные программы с высокой производительностью. Тем не менее процесс разработки не обязан быть медленным и болезненным. При правильном абстрагировании программы на C++ можно писать довольно быстро. Более того, мы оптимистично полагаем, что в будущем в стандарт C++ будет включено больше научных библиотек.

Очевидно, что чем больше языков мы знаем, тем больший выбор у нас имеется. Кроме того, чем лучше мы знаем эти языки, тем более разумным будет наш выбор. Большие проекты часто содержат компоненты на различных языках, и при этом в большинстве случаев по крайней мере ядра, критичные к производительности, реализованы на C или C++. Все это говорит в пользу того, что изучение C++ — не пустая трата времени, и его глубокое понимание в любом случае будет не лишним, если вы хотите стать великим программистом.

Соглашения об оформлении

Новые термины и понятия выделены *курсивом*. Для исходных текстов на C++ использован `моноширинный шрифт`. Важные детали отмечены **полужирным шрифтом**. Классы, функции, переменные и константы даны в нижнем регистре и при необходимости содержат символы подчеркивания. Исключение составляют матрицы, которые обычно именованы с первой заглавной буквой. Параметры шаблонов и концепции начинаются с заглавной буквы и могут содержать дополнительные заглавные буквы (например, CamelCase). Команды и выход программы `выводятся в следующем виде`.

Программы, требующие возможностей C++03, C++11 или C++14, помечены соответствующей пиктограммой. Некоторые программы, использующие только небольшое количество возможностей C++11, которые легко заменить выражениями C++03, явно не помечаются.

⇨ `directory/source_code.cpp`

За исключением очень короткого иллюстрирующего кода, все примеры программ в этой книге были протестированы по крайней мере на одном компиляторе. Стрелкой указывается путь к полной программе, приводимый в начале абзаца или раздела, в котором обсуждаются содержащиеся в ней фрагменты кода.

Все программы доступны на GitHub в открытом репозитории `https://github.com/petergottschling/discovering_modern_cpp`, так что их можно получить с помощью команды

```
git clone https://github.com/petergottschling/discovering_modern_cpp.git
```

В Windows удобнее использовать TortoiseGit (см. `tortoisegit.org`).