Контрольные вопросы

16 ноября 2022 г.

1 Приведите примеры ситуаций, в которых удобно использовать вариативные шаблоны.

Возможность сопоставлять параметрам шаблонов переменное количество аргументов шаблона позволяет использовать шаблоны в местах, где требуется передать произвольное количество аргументов произвольных типов. Вариативные шаблоны играют важную роль при реализации обобщённых библиотек, в частности, стандартной библиотеки C++.

Примеры ситуаций, в которых удобно использовать вариативные шаблоны:

- Передача произвольного количества аргументов различных типов:
  - Передача аргументов конструктору нового объекта в динамической памяти, которым владеет интеллектуальный указатель (move only object, так сказать). Иными словами, вариативный шаблон используется как обёртка вокруг другого объекта неизвестного типа.
  - Передача аргументов в поток выполнения, запускаемый библиотекой (<thread>).
  - Передача аргументов в конструктор нового элемента, добавляемого к вектору. (При работе с шаблонизированными методами вектора).

    Часто эти операции следуют move-semantic (т.е. аргументы передаются с использованием move-semantic).
- При создании вариативных шаблонов классов:
  - Создание класса, в котором произвольное количество параметров шаблона определяет типы соответствующих членов (т.е. реализация гетерогенных контейнеров, причём compile-time). Например, реализация std::tuple.
  - Создание класса, членами которого могут лишь указанные типы объектов.
  - Реализация класса, представляющего собой список индексов. (Данный класс может быть использован для реализации функций, работающих с некоторыми массивами).
- Создание обобщенного кода для обработки любого количества параметров произвольных типов. Например, реализация print(). Кроме того, при создании обобщённых функций используются свёртка и вариативные выражения, которые расширяют возможности и позволяют удобно и быстро(?) вычислять результаты.

2 Как можно обработать по очереди все аргументы из пакета аргументов функции?

Поочерёдная обработка всех аргументов из пакета аргументов функции может быть реализована с помощью вариативного шаблона функции, указывающего отдельно первый элемент, и её нешаблонной перегрузки, фактически обеспечивающей остановку рекурсии (обычно вызывается при пустом пакете параметров). А именно, если передаются один или несколько аргументов, используется шаблон функции, который позволяет обрабатывать первый аргумент (указанный отдельно), перед тем как рекурсивно та же функция применится далее к последующим аргументам. Наконец, нешаблонная перегрузка рассматриваемой функции, которая вызывается при пустом пакете параметров, остановит рекурсию.

Кроме того, в качестве замыкающей функции может быть использован и некоторый шаблон той же функции от какого-либо числа аргументов (шаблонная перегрузка). При этом рекурсия остановится, поскольку если два шаблона функций отличаются только завершающим пакетом параметров, то шаблон функции без такого пакета является предпочтительным.

Возможна обработка всех аргументов из пакета аргументов функции с помощью оператора sizeoff... для вариативных шаблонов (см. вопрос 3) с использованием инструкции if времени компиляции. А именно проверяется количество оставшихся аргументов, а затем происходит обработка, если есть, что обрабатывать. Однако этот подход не сработает при использовании обычной инструкции if, потому что в общем случае инстанцируются обе ветви инструкции if. Но будет ли инстанцированный код нужен – выясняется во время выполнения, в то время как инстанцирование вызова происходит во время компиляции. По этой причине, если вызывается шаблон для одного (последнего) аргумента, то инструкция с вызовом вариативно шаблонизированной функции тоже инстанцируется – на этот раз без аргументов, и если такой функции нет соответствующей перегрузки без аргументов, то будет получена ошибка.

Наконец, в некоторых случаях обработка всех аргументов из пакета аргументов функции может быть реализована с помощью выражений свёртки. Они дают возможность вычисления результата с применением бинарного оператора ко всем аргументам пакета параметров. Поэтому, если обработка подразумевает последовательное использование одного и того же оператора, можно использовать данный способ. При этом может потребоваться дополнительный класс или дополнительная функция для достижения необходимых целей реализации.

3 Как вычислить количество параметров в пакете параметров вариативного шаблона?

В C++11 введена новая версия оператора sizeof для вариативных шаблонов: sizeof... . Оператор может быть вызван как для пакета параметров шаблона, так и для пакета параметров функции. В любом случае результат его применения – количество обрабатываемых им типов (в случае вызова от пакета параметров шаблона) или аргументов (в случае вызова от пакета параметров функции).

4 Какие существуют разновидности выражений свёртки и когда они применяются?

С помощью свёртки (fold) сокращается (сворачивается) пакет параметров над бинарным оператором, т.е. вычисляется результат с применением бинарного оператора ко всем аргументам пакета параметров. Инстанцированные выражения свёртки вычисляются одним из способов (приведены разновидности выражений и соответствующие вычисления для пакета параметров из N элементов):

- unary left fold (... op pack) becomes (((packl op pack2) op pack3) ... op packN);
- unary right fold (pack op ...) becomes (packl op (... (packN-1 op packN)));
- binary left fold (init op ... op pack) becomes ((init op packl) op pack2) ... op packN);
- binary right fold (pack op ... op init) becomes (packl op (... (packN op init))).

Применение того или иного выражения свёртки определяется тем, как оно вычисляется. Например, бинарная левая свёртка может быть использована при обходе дерева или при выводе значений на консоль. Унарные свёртки могут использоваться, когда для некоторых объектов вызываются бинарные операторы, рассчитывающие некоторые значения, например, сумму или когда бинарный оператор служит разделителем между пакетом и некоторым объектом, к которому применяется функция (часто используется при наличии оператора (,)).

Когда унарная свёртка используется с пакетом параметров нулевой длины, разрешено использование лишь трёх операторов:

- &&. Значение пустого пакета оценивается как true.
- ||. Значение пустого пакета оценивается как false.
- (,). Значение пустого пакета оценивается как void().

5 В чём заключается разница между динамическим и статическим полиморфизмом?

Динамический полиморфизм в C++ реализуется посредством наследования классов и механизма виртуальных функций (наглядный пример – рассмотренная ранее иерархия геометрических фигур). Статический же полиморфизм реализуется шаблонами.

Разница между ними заключается в том, что:

- Динамический полиморфизм в C++ является ограниченным, т.е. интерфейсы типов, участвующих в процессе полиморфизма, предопределены дизайном общего базового класса.

  Статический же полиморфизм неограниченный, т.е. интерфейсы типов, участвующих в процессе полиморфизма, не предопределены заранее.

- При динамическом полиморфизме связывание интерфейсов наследующих классов и базового происходит в процессе выполнения программы (посредством т.н. виртуальной таблицы).

  При статическом – связывание интерфейсов происходит в процессе компиляции (ведь все типы уже должны быть известны).

Кроме того, для сравнения можно выделить преимущества и недостатки (а также некоторые дополнительные особенности/различия) динамического и статического полиморфизмов. Итак, если речь идёт о динамическом полиморфизме:

- Возможна удобная обработка разнородных коллекций.
- Размер исполняемого кода потенциально меньше (поскольку в данном случае нужна только одна полиморфная функция, тогда как для шаблонов с разными параметрами типов должны быть сгенерированы *отдельные экземпляры*).
- Код полностью компилируем.

Если речь идёт о статическом полиморфизме:

- Легко реализуются коллекции встроенных типов. Общность интерфейса не обязательно должна выражаться через общий базовый класс.
- Сгенерированный код потенциально выполняется быстрее (поскольку отсутствует необходимость в косвенном обращении через указатели постредством виртуальной таблицы, а невиртуальные функции могут быть встраиваемыми намного чаще).
- Могут использоваться конкретные типы, в которых имеются только частичные интерфейсы.

Помимо того, часто статический полиморфизм расценивается как более надёжный в плане безопасности типов, чем динамический, поскольку все связывания выполняются в процессе компиляции. На практике инстанцирование шаблонов может вызвать определенные неприятности в том случае, когда за идентично выглядящими интерфейсами скрываются разные семантические допущения (речь, например, о наличии ассоциативного оператора+ у типа, который таким оператором не обладает). Обычно этот вид семантического несоответствия встречается гораздо реже в иерархиях, основанных на наследовании; вероятно, это связано с более явным и точным определением интерфейса.

Литература

[1] Вандевурд, Дэвид, Джосаттис, Николаи М., Грегор, Дуглас. В17 Шаблоны C++. Справочник разработчика, 2-е изд.: Пер. с англ. — СпБ.: ООО "Альфа-книга 2018. – 848 с.: ил. – Парал. гит. англ.

[2] https://en.cppreference.com/w/cpp/language/fold