Контрольные вопросы. Задание 12.

29 ноября 2022 г.

1 Что означает полнота по Тьюрингу в теории вычислимости?

Полнота по Тьюрингу – характеристика исполнителя (множества вычисляющих элементов) в теории вычислимости, означающая возможность реализовать на нём любую вычислимую функцию (функцию вида $f{:}N \to N$, которая может быть реализована каким-либо Тьюринг-полным исполнителем). Другими словами, для каждой вычислимой функции существует вычисляющий её элемент или программа для исполнителя, а все функции, вычисляемые множеством вычислителей, являются вычислимыми функциями (возможно, при некотором кодировании входных и выходных данных).

Язык программирования C++ обладает как тьюринг-полнотой времени компиляции, так и тьюринг-полнотой времени исполнения.

2 Как можно использовать вычисления на этапе компиляции?

Вычисления на этапе компиляции могут быть использованы для метапрограммирования (т.е. для создания кода, который, будучи выполнен системой программирования, в свою очередь генерирует новый код, реализующий необходимую функциональность). Часто вычисления на этапе компиляции выполняются при использовании различных средств метапрограммирования. Более конкретно, вычисления на этапе компиляции могут быть использованы для:

- Метапрограммирования значений (т.е. для программирования вычислений значений во время компиляции).
- Метапрограммирования типов (т.е. для программирования вычислений типов во время компиляции).
- Гибридного метапрограммирования (см. в.5).

Метапрограммирование позволяет выполнять определенную часть необходимых пользователю вычислений на этапе трансляции программы. Его применение (а следовательно и использование вычислений на этапе компиляции) нередко объясняется повышением производительности (вычисление, которое выполняется во время трансляции, зачастую может быть полностью удалено из программы) или упрощением интерфейса (в общем случае метапрограмма короче, чем программа, в которую она раскрывается), а иногда и обеими этими причинами.

Ярким примером конструкций, использующих вычисления на этапе компиляции, являются constexpr и compile-time if (а также enable_if). Они позволяют достичь выполнения лишь первого этапа двухэтапной трансляции (время определения). Например, это позволяет реализовать один из вариантов определения числа параметров вариативного шаблона. Кроме того, constexpr и compile-time if позволяют уменьшить стоимость рекурсивного инстанцирования шаблонов.

3 Какие языковые механики работают на этапе компиляции?

На этапе компиляции работают языковые механики:

- Двухэтапная трансляция. При этом, используя enable_if или compile-time if, можно достичь выполнения лишь первого этапа трансляции.
- Отсутствие компиляции шаблонов в единую сущность, способную обработать любой тип данных. Вместо этого из шаблона генерируются различные объекты для каждого типа, для которого применяется шаблон. Поэтому при рекурсивном инстанцировании шаблонов необходим отдельный шаблон, обеспечивающий остановку рекурсии.
- Идиома SFINAE, обеспечивающая возможность выбора между различными реализациями шаблона функции для разных типов или разных ограничений. Это важно не только для рекурсивного инстанцирования шаблонов, но и для уменьшения стоимости этого инстанцирования (см. в.2).
- Возможность оценивания constexpr выражений (а также выражений в конструкции compile-time if или enable_if) во время компиляции.

4 Какие ограничения имеет метапрограммирование шаблонов?

Всеобъемлющее метапрограммное решение в C++ должно быть эффективным в трёх измерениях:

- вычисления;
- рефлексия – возможность функционально проверять возможности программы;
- генерация – возможность создания дополнительного кода программы.

Для достижения рефлексии удобно использовать вычисления, основанные на рекурсивном инстанцировании шаблонов. Но экземпляры шаблонов классов потребляют много памяти компилятора, которая не может быть освобождена до конца компиляции (в противном случае потребуется значительно больше времени на компиляцию). Даже сравнительно скромные шаблоны класса могут потребовать более килобайта памяти для каждого инстанцирования. Альтернативный вариант заключается в том, чтобы ввести новый стандартный тип для представления рефлексивной информации.

Кроме того, если шаблонный код включает условные конструкции, то, вообще говоря, инстанцируются шаблоны во всех ветвях (и при каждом инстанцировании могут инстанцироваться ещё и все члены внутри типа класса), если они есть. Итоговое количество инстанцирований может сильно превышать ожидаемое. К счастью, данную проблему можно решить, например, с помощью идиомы SFINAE и механизма std::enable_if.

Язык программирования C++ является полным по Тьюрингу, но вычисление всех функций, являющихся вычислимыми, с помощью шаблонов может быть не слишком удобным. Кроме того, поскольку экземпляр шаблона требует существенного количества ресурсов компилятора, обширные рекурсивные инстанцирования быстро замедлят работу компилятора. Как итог, Стандарт C++ рекомендует (но не предписывает) разрешить как минимум 1024 уровня рекурсивного инстанцирования, что достаточно для большинства (но, конечно, не для всех) задач шаблонного метапрограммирования.

5 Как устроено гибридное метапрограммирование шаблонов?

При гибридном метапрограммировании шаблонов во время компиляции можно программно собрать фрагменты кода с результатом, получаемым во время выполнения. Иными словами, при таком подходе смешиваются/сочетаются вычисления времени компиляции (которые могут быть достигнуты различными средствами метапрограммирования), определяющие общую структуру кода, с вычислениями времени выполнения (которые связаны с использованием конкретных функций), которые и определяют конкретный результат вычислений времени выполнения.

Гибридное метапрограммирование шаблонов используется, например, при реализации метода разворачивания цикла (loop unrolling) (вычисления времени компиляции осущетсвляются посредтством рекурсивного инстанцирования шаблона, а конкретный результат – с помощью конкретных функций); при реализации библиотек, которые в состоянии вычислять результаты значений типов различных единиц (вычисление значений происходит во время выполнения, а результирующие единицы (например, соответствующие типы) вычисляются во время компиляции).

Литература.

[1] Вандевуд, 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://ru.wikipedia.org/wiki/