Билет №3.5

Метапрограммирование. Шаблоны и Generics. Частичная специализация шаблонов.

1 Метапрограммирование

Метапрограммирование — создание программ, которые создают другие программы как результат своей работы (либо — частный случай — изменяющие или дополняющие себя во время выполнения).

Метапрограммирование можно разделить на 2 направления: на *стадии компиляции (генерация кода)* и на *стадии выполнения (самомодифицирующийся код)*.

Первое направление позволяет получить программу при меньших затратах времени и усилий, чем если бы программист писал её вручную. Второе — расширяет возможности программиста.

1.1 Генерация кода

При этом подходе код программы не пишется вручную, а создается автоматически программой-генератором на основе другой, более простой программы.

Такой подход приобретает смысл, если при программировании вырабатываются различные дополнительные правила (более высокоуровневые парадигмы, выполнение требований внешних библиотек, стереотипные методы реализации определенных функций и пр.). При этом часть кода теряет содержательный смысл и становится лишь механическим выполнением правил. Когда эта часть становится значительной, возникает мысль задавать вручную лишь содержательную часть, а остальное добавлять автоматически. Это и проделывает генератор.

Реализуется 2 основными методами:

1. Шаблоны (наиболее известные случаи применения — препроцессор `C` и шаблоны в `C++`). Решают задачу, если соблюдение «правил» сводится к вставке в программу повторяющихся (или почти повторяющихся) кусков кода. Помимо этого, обладают еще рядом достоинств: например, помогают повторному использованию.

2. Внешнеязыковые средства (пример: генераторы синтаксических и лексических анализаторов `lex`, `yacc`, `bison`). Применяются в случаях, если простых средств вроде шаблонов недостаточно. Язык генератора составляется так, чтобы автоматически или с минимальными усилиями со стороны программиста реализовывать правила парадигмы или необходимые специальные функции. Фактически, это — более высокоуровневый язык программирования, а генератор — не что иное, как транслятор. Генераторы пишутся, как правило, для создания специализированных программ, в которых очень значительная часть стереотипна, либо для реализации сложных парадигм (таких, как паттерны проектирования).

1.2 Самомодифицирующийся код

Возможность изменять или дополнять себя во время выполнения превращает программу в виртуальную машину. Хотя такая возможность существовала уже давно на уровне машинных кодов (и активно использовалась, например, при создании полиморфных вирусов), с метапрограммированием обычно связывают перенос подобных технологий в высокоуровневые языки.

Основные методы реализации:

1. Интроспекция — представление внутренних структур языка в виде переменных встроенных типов с возможностью доступа к ним из программы. Позволяет во время выполнения смотреть, создавать и изменять определения типов, стек вызовов, обращаться к переменной по имени, получаемому динамически и пр.

2. Пространство имён `System.Reflection` и тип `System.Type` в `.NET`; классы `Class`, `Method`, `Field` в `Java`; представление пространств имен и определений типов через встроенные типы данных в `Python`; стандартные встроенные возможности в `Forth` по доступу к ресурсам виртуальной машины.

3. Интерпретация произвольного кода, представленного в виде строки.

   - Существует естественным образом во множестве интерпретируемых языков, например `eval()` в `PHP`.
   - Для `C++` есть библиотека, позволяющая «на лету» компилировать и генерировать исполняемый код (используется урезанный компилятор `gcc`).

- Для Forth использования процедуры интерпретации из строки EVALUATE.

Принципиальный недостаток технологий этого направления — неприменимость к компилируемым языкам. Можно ввести в такой язык интерпретатор, как в вышеуказанной библиотеке для C++, но это практически сведет на нет главное преимущество данных языков — производительность. Хороший задел, компиляции программы при загрузке, сравнимый с C демонстрируют удачные реализации Forth языка.

2 Шаблоны и Generics

Обобщённое программирование — это парадигма программирования, заключающаяся в написании алгоритмов, которые можно применять к различным типам данных. В том или ином виде поддерживается разными языками программирования. Возможности обобщённого программирования впервые появились в 70-х годах в языках CLU и Ada, а затем во многих объектно-ориентированных языках, таких как C++, Java, D и языках для платформы .NET.

2.1 Шаблоны

В языке C++ обобщённое программирование основывается на понятии «шаблон», обозначаемом ключевым словом template.

```
// returns maximum of two elements
template <typename T> T max(T x, T y)
{
    if (x < y)
        return y;
    else
        return x;
}
```

Интересное применение нашли шаблоны в языке C++. Оказалось, что шаблоны в этом языке являются тьюринг-полным функциональном языком. Другими словами на шаблонах C++ можно написать программу, реализующую произвольный алгоритм, и эта программа выполнится в момент компиляции. К примеру, можно посчитать 50-е число Фибоначчи. Тогда во время выполнения программы не придется тратить время на его вычисления. Одной интересной особенностью такого программирования на шаблонах, является встроенный механизм *мемоизации* (сохранения результата вычисления функции). Это значит, что рекурсивный алгоритм для вычисления k-го числа Фибоначчи работающий «в лоб» сделает порядка k операций (вместо ожидаемых 2^k).

```
template <int i> struct fib
{
   static const int val = fib<i - 1>::val + fib<i - 2>::val;
};

template <> struct fib<1>
{
   static const int val = 1;
};

template <> struct fib<2>
{
   static const int val = 1;
};

int main()
{
   std::cout << fib<20>::val;
}
```

Но это не самое интересное: программирование на шаблонах C++ позволяет общаться с типом как с обычным объектом. К примеру, можно составить список типов, удалить из него все встроенные типы, а из оставшегося списка создать объект, который будет унаследован от всех типов из данного списка. Для такого метапрограммирования была написана специальная библиотека MPL *(MetaProgramming Library)*.

2.2 Generics

Язык Java предоставляет средства обобщённого программирования, синтаксически основанные на C++. В Java **generics** (параметризованные типы или родовые типы) имеют мнимое сходство с шаблонами C++ как

по синтаксису, так и по ожидаемому месту их применения (например, в качестве контейнерных классов). Но это сходство только поверхностное — родовые типы в языке программирования Java почти полностью реализуются в компиляторе, который выполняет проверку типов и выявление типа *(type inference)* и, затем, генерирует обычные не параметризованные байткоды. Такая техника реализации, называемая *стиранием* (когда компилятор использует информацию о родовом типе для контроля типов и удаляет ее перед генерированием байткода), имеет неожиданные, а иногда и непонятные последствия. В то время как родовые типы являются большим шагом на пути к безопасности Java-классов, изучение их использования почти наверняка будет вызывать некоторую озадаченность (а иногда и мучения).

3 Частичная специализация шаблонов

Если у шаблона класса есть несколько параметров, то можно специализировать его только для одного или нескольких аргументов, оставляя другие неспециализированными. Иными словами, допустимо написать шаблон, соответствующий общему во всем, кроме тех параметров, вместо которых подставлены фактические типы или значения. Такой механизм носит название частичной специализации шаблона класса. Она может понадобиться при определении реализации, более подходящей для конкретного набора аргументов.

Рассмотрим шаблон класса Screen. Частичная специализации Screen<hi,80> дает более эффективную реализацию для экранов с 80 столбцами:

```
template <int hi, int wid>
class Screen {
};

template <int hi>
class Screen<hi, 80> {
public:
    Screen();
    // ...
private:
    string             screen;
    string::size_type  cursor;
    short              height;
};
```

Частичная специализация шаблона класса — это шаблон, и ее определение похоже на определение шаблона. Список параметров здесь отличается от соответствующего списка параметров общего шаблона. Для частичной специализации шаблона Screen есть только один параметр-константа hi, поскольку значение второго аргумента равно 80, т. е. в данном списке представлены только те параметры, для которых фактические аргументы еще неизвестны. Имя частичной специализации совпадает с именем того общего шаблона, которому она соответствует, в нашем случае Screen. Однако за ее именем всегда следует список аргументов.