# Предобработка шаблонов

Первый этап работы системы заключается в анализе «сырых» шаблонов, которые хранятся в папке *elements*. Этот анализ выполняется с помощью скрипта *make\_configure.py*. Для его работы необходимо наличие папки *elements* и файла *slots.txt*.

### Файлы

В папке *elements* располагаются файлы двух типов – шаблоны (вернее их заготовки) и модули общего назначения. Имя каждого файла-шаблона имеет следующий вид: *<имя слота>\_<имя шаблона>.cpp*.Например, файл *algorithm\_ga.cpp* содержит информацию о шаблоне *ga* (генетический алгоритм), который может быть использован для подстановки в слот *algorithm*.

Имена файлов модулей общего назначения начинаются с символа подчеркивания. Например, файл *\_randbool.cpp* содержит шаблон кода функции, генерирующей случайное логическое значение (0 или 1).

Файл *slots.txt* содержит список всех слотов, доступных системе. Этот список упорядочен в смысле *топологической сортировки* – все слоты, которые могут появиться в результате замены данного слота на любой из шаблонов, описываются ниже этого слота. Каждая строка файла содержит описание одного слота, сначала указывается его имя, затем короткая характеристика, используемая впоследствии для организации диалога с пользователем.

### Организация файлов с заготовками шаблонов

Каждый файл с заготовкой шаблона в папке *elements* представляет собой набор секций, каждая секция начинается со строки-заголовка, имеющей вид *[заголовок]*. Имеется несколько типов секций.

* *[title]* – короткое (в одну строку) описание шаблона, используется при диалоге с пользователем.
* *[details]* – развернутое (возможно в несколько строк) описание шаблона, используется, например, в качестве автоматически вставляемого комментария в генерируемом коде программы.
* *[include]* – список заголовочных файлов, которые необходимо подключить к генерируемой программе, для корректной работы *данного* шаблона. Каждый файл надо указывать так же, как в инструкции *#include* в программе на С++, например, *“stdafx.h”* или *<iostream>*.
* *[vars]* – список переменных шаблонирования, которые будут использоваться для замены цепочек вида *$variable$* при генерации программы. Каждая переменная описывается в отдельной строке, сначала указывается ее имя, затем значение. Например, строка *$gtype$ int* определяет переменную с именем *gtype* и со значением *int*. При генерации кода каждая цепочка *$gtype$* будет заменена на цепочку *int*.
* *[params]* – список числовых параметров, определяющих работу кода, представленного данным шаблоном. Каждый параметр описывается в отдельной строке. Описание параметра состоит из 1) его типа; 2) имени; 3) значения по умолчанию; 4) комментария (к комментарию автоматически относится весь остаток строки). При генерации кода параметры всех используемых шаблонов автоматически упаковываются в специальную структуру (типа parameters), к которой возможен доступ из любого оператора алгоритма.
* *[depends]* – список явных зависимостей данного шаблона от других шаблонов. Каждая зависимость описывается одной строкой, сначала указывается имя слота, затем имя шаблона. Например, шаблон *crossover\_singlepoint* (одноточечное скрещивание) зависит от шаблонов *encoding\_bool*, *encoding\_int* и т.д. – это значит, что данный шаблон может быть подставлен вместо слота *crossover* только при условии, что выбран тип кодирования двоичными, целочисленными и т.п. последовательностями, а не, например, перестановками.
* *[code]* – собственно код данного шаблона. Может содержать в себе 1) переменные шаблонирования (см. секцию *vars* выше); 2) слоты, которые помечаются, как *//:SLOT <slotname>*; 3) инструкции шаблонирования, описание которых см. ниже.
* *[end]* – завершает описание данного файла.

Файлы, описывающие модули общего назначения, содержат только код соответствующего модуля, но с поддержкой переменных и инструкций шаблонирования.

### Скрипт предобработки

Данный скрипт запускается командой *python make\_configure.py*. Результатом работы скрипта являются 1) папка elements со всеми шаблонами (очищенными от метаинформации); 2) единый файл *system.js* конфигурации всей системы слотов и шаблонов, в этом файле оказывается собранной вся метаинформация от всех шаблонов. Информация в этом конфигурационном файле устроена следующим образом.

Весь файл является *списком слотов*. Каждый слот описывается списком, в котором сначала указываются название слота (например, *algorithm*), затем краткое его описание (взятым из файла *slots.txt*). После этого перечисляются все шаблоны, привязанные к данному слоту.

Каждый шаблон представляется *словарем*, в котором указываются все параметры данного шаблона, в которые входят, в частности, информация обо всех секциях из заготовки шаблона. Кроме того, в список параметров входит параметр *id=<имя шаблона>* (берется из имени заготовки шаблона) и параметр *contains=<список слотов>*, включающий список всех слотов, обнаруженных в данном шаблоне.

# Диалог с пользователем

*В настоящее время* диалог с пользователем реализуется скриптом *configure.py* в формате помощника (wizard) в обычном текстовом режиме. Работа скрипта заключается в постепенной детализации начального слота *program*. Просматриваются все слоты в файле *system.js*. Если текущий слот подлежит детализации, то составляется список шаблонов доступных для этого слота и формируется запрос пользователю. Выбор пользователя запоминается. Список слотов, требующих детализации, расширяется слотами, содержащимися в выбранном пользователем шаблоне. После этого пользователем определяются значения параметров, привязанных к данному шаблону. Вся собранная информация сохраняется в файле *user.js*.

Файл user.js представляет собой список из двух словарей. Первый словарь содержит все слоты, которые были детализированы пользователем. Каждая запись в этом словаре имеет вид <слот> = <шаблон> и соответствует сделанному пользователем выбору. Второй словарь содержит все параметры, настроенные пользователем. Каждая запись в этом словаре имеет вид <параметр> = <значение>. Вся остальная информация о данном параметре (прежде всего его тип) доступна из файла *system.js* по имени этого параметра.

# Генерация кода

Генерация кода выбранного и настроенного пользователем алгоритма выполняется в два этапа. На первом подготавливаются данные для генерации, используя файлы *system.js* и *user.js*, а на втором этапе производится собственно генерация кода. Сначала рассмотрим, как выполняется второй этап, который реализуется скриптом *apply.py*. По сути, этот скрипт выполняет универсальную операцию по генерации текста, заполняя шаблон специально подготовленными данными.

### Генерация текста программы

Генерация текста выполняется рекурсивным образом функцией *apply*. Параметрами функции служат:

* текст шаблона *text*, представляющий собой список строк (заканчивающихся символом новой строки);
* текущая конфигурация *data*, (словарь, формат его описан ниже);
* текущий список (словарь) переменных шаблонирования *vars*;
* текущий список выполненных подстановок файлов *includes*.

Результатом работы функции является готовый текст *S*, представленный также списком строк, заканчивающихся символом новой строки.

Текст шаблона анализируется построчно. Каждая строка является либо простой строкой, либо командой шаблонирования. В случае простой строки, производится подстановка в нее значений тех переменных, которые имеются в этой строке. Каждая переменная в строке представлена своим именем, заключенным в знаки доллара, например, *$gtype$*. Если в списке *vars* имеется переменная с именем *gtype* = *int*, то цепочка *$gtype$* будет заменена на *int*. Если такой переменной нет, то эта цепочка просто удаляется из текста. Модифицированная таким образом строка добавляется в готовый текст *S*.

Каждая команда шаблонирования представляет собой строку, которая начинается с комбинации символов «*//:*». Поддерживаются два вида команд – команда вставки файла и команда вставки *секции*. Команда вставки файла имеет вид: *//:LOAD <имя файла>*. Если этот файл еще не использовался для вставки (т.е. его имени нет в списке *includes*), то выполняется вставка. Файл с указанным именем загружается из рабочей папки (в нашем случае из папки *elements2*), опять же, как список строк. Этот список включается в текст шаблона *text*, вместо выполняемой команды шаблонирования, и текущей строкой для обработки оказывается первая строка загруженного файла. Наконец, имя загруженного файла добавляется в список *includes*, что блокирует дальнейшие вставки этого файла в текст шаблона.

Команда вставки секции имеет вид *//:<имя секции>+*. При выполнении этой команды далее в тексте шаблона ищется команда закрытия секции *//:<имя секции>-*. Весь текст между этими двумя командами выделяется и записывается в переменную *text2*. В словаре *data* в поле *BODY* ищутся все записи с ключом *KEY*, значение которых равно имени текущей секции. Перебираем все эти записи, каждая из которых имеет тот же формат, что и словарь *data*. В каждой записи *r* выбираются все ключи не равные *KEY* или *BODY* и соответствующие пары ключ-значение добавляются в словарь *vars* – это переменные шаблонирования, связанные с данной секцией. Далее производится рекурсивный вызов функции *apply* для текста *text2*, конфигурации *r*, обновленного списка переменных *vars* и списка включенных файлов *includes*. Полученный в результате этого вызова текст добавляется к *S*.

Описанный прием позволяет выполнять итерации при генерации текста – из одной секции делается несколько копий, для генерации которых используются свои собственные данные. Кроме того, возможно использовать условную вставку секции – если в *data* отсутствует соответствующая запись, то текст секции не будет сгенерирован.

Рассмотрим пример работы функции *apply*. Пусть список переменных *vars = { }*, т.е. является пустым. Пустым является также список *includes = [ ]*. Текущей секция является *GLOBAL*, ее конфигурация *data* показана на следующем рисунке.



Тогда текст шаблона *text* (слева) будет преобразован в готовый текст *S* (справа) следующим образом.

|  |  |  |
| --- | --- | --- |
| // $details$  //:INCLUDE+  #include $header$  //:INCLUDE-  //:NAMESPACESTD+  using namespace std;  //:NAMESPACESTD-  struct parameters  {  //:PARAMETER+  $type$ $name$; // $details$  //:PARAMETER-  parameters()  {  //PARAMETER+  $name$ = $value$;  //PARAMETER-  }  }; | → | // optimizing algorithm...  #include <iostream>  using namespace std;  struct parameters  {  int pop\_size; // population size  int task\_dim; // task dimension  parameters()  {  pop\_size = 100;  task\_dim = 20;  }  }; |

### Подготовка данных для генерации кода

Данные для генерации кода готовятся с помощью скрипта *gen.py*. На его вход подаются конфигурационный файл всей системы *system.js* и выбранная пользователем конфигурация *user.js*. Выходом является файл (например, *config.js*), содержащий все необходимые данные для генерации кода (которая будет выполнена с помощью скрипта *apply.js*).

# Общая архитектура системы

