Инструкция по созданию шаблонов

Предположим, что имеется некоторое задание и нужно сделать новый версию данного задания.

Изначальное задание имеет следующий вид :

> Написать программу для вычисления функции y = f(x)
>
> - y = cos(x) / (x + 10) при x < -3
> - y = e ^ (0.1x) при -3 <= x < 4
> - y = lg(x) при 4 <= x < 6
> - y = sin(x) при x >= 6

Новый вариант задания скорее всего будет иметь другие выражения и диапазоны в которых они вычисляются.

Шаблон такого задания может иметь следующий вид

> Написать программу для вычисления функции y = f(x)
>
> - y = (выражение 1) при x < (1-ый диапазон)
> - y = (выражение 2) при (1-ый диапазон) <= x < (2-ой диапазон)
> - y = (выражение 3) при (2-ой диапазон) <= x < (3-ий диапазон)
> - y = (выражение 4) при x <= (3-ий диапазон)

То есть очередной вариант задания - это случайно сгенерированные выражения и диапазоны в которых они вычисляются и подставленные в нужное место. Для хранения таких выражений и диапазонов удобно использовать переменные или `объекты-параметры`, так как выражения и диапазоны могут понадобиться при создании `эталонного решения`.

Для того чтобы по одному шаблону построить: `вариант задания`, `эталонное решение` и `тестовые данные` - потребуется специальный шаблон-файл.

На данном этапе мы подходим к формату шаблонов заданий. Шаблон-файл разделен на 5 секций.

Описание блоков

Секция № 1 - ХРАНИЛИЩЕ_ОБЪЕКТОВ

`ХРАНИЛИЩЕ_ОБЪЕКТОВ` - хранит объекты-параметры.

`Объект-параметр` - набор символов который можно подставить в необходимое место. Объект-параметр может содержать как уже готовую последовательность символов так и исполняемую функцию, которая сначала выполняется, а потом помещает в объект-параметр значение.

объект-параметр имеет следующий синтаксис :

```
объект_параметр = “имя объекта-параметра” “:” значение_объекта_параметра {“,” значение_объекта_параметра} “;”
значение_объекта_параметра = “набор символов юникода” | функция
функция = “#имя_функции({аргументы функции [|]})”
```

Если описывать проще то синтаксис можно представить следующим образом (пробелы не играют никакой роли):

```
имя параметра : содержимое объекта параметра;
выражение 1   : sin(x + 270);
```

Также объекты-параметры могут содержать функции:

- `#rnd ( нижняя_граница | верхняя_граница | тип {| количество})` - генерирует случайное число в границе [нижняя_граница, верхняя_граница) типа `тип (double или int)`. Если необходимо сгенерировать сразу несколько значений используется дополнительный параметр “количество”, в таком случае сгенерированные числа будут разделены запятыми;

  ```
   было = число : #rnd(1  | 10 | int);
  стало = число : 5;
  ```

- `#genAE(сложность {| граница | тип | количество переменных})` - генерирует выражение некоторой сложности (сложность указывает количество знаков и функций (например log, sin и т.д.)), граница указывает диапазон генерации констант [-граница, граница), тип указывает на тип генерируемых констант (`double` или `int`), количество переменных, указывает на зависимость выражения от количества переменных (от 1 до бесконечности);

  ```
   было = выр : #genAE(5);
  стало = выр : cos(-99.45 + log(51.01 - (18.82 * x1)));

   было = выр2 : #genAE(5 | 50.0 | double | 2);
  стало = выр2 : (x1 / ((x2 + tan(x1)) * 18.82)) + x1;
  ```

- `#getAEcode()` - возвращает код для вычисления последнего сгенерированного выражения (вместе с проверками на возможность вычисления выражения) на языке `C++`.

  ```
  было =
      выр : #genAE(5);
      код : #getAEcode();

  стало =
      выр : cos(-99.45 + log(51.01 - (18.82 * x1)));
      код : if (((51.01 - (18.82 * x1)) > 0 && (51.01 - (18.82 * x1)) != 1))
  ```

```
                y = cos(-99.45 + log(51.01 - (18.82 * x1)));
            else
                y = 0.0;
```

- `#lua(код_на_луа)` - данная функция является второстепенной и предназначена для выполнения кода на `Lua` (изначально предполагалось использовать `Lua` лишь для вычисления мат. выражений, но в дальнейшем данный язые будет использовать для расширения предоставляемого функционала генератора). Все системные функции (кроме получения времени и даты) на `Lua` которые обращаются к OS запрещены в целях безопасности.

```
было =
    число : #lua("
        math.randomseed(12345689);
        return math.random(100);
        ");
стало =
    число : 88;
```

На данном этапе стало чуть-чуть понятнее про объекты-параметры. Теперь нужно сказать о там как их использовать, все довольно просто, чтобы использовать параметр нужно прибегнуть к следующему синтаксису:

```
@имя_объекта_параметра@
```

Между знаками `@` и `имя_объекта_параметра` пробелов быть не должно.

Пример использования подстановок:

```
было =
    количество : 100;
    задание: запишите в файл "Input.txt" @количество@ знаков;

стало =
    количество : 100;
    задание: запишите в файл "Input.txt" 100 знаков;
```

Секция 2 - ШАБЛОННЫЙ_ВИД

Данная секция шаблон-файла содержит конкретный вид варианта задания. В данном блоке можно и нужно использовать ранее описанные объекты-параметры.

Например:

```
Создайте файл с именем @имя файла@ и запишите в него @количество@ случайных чисел
```

Если мы опишем объекты-параметры `@имя файла@` и `@количество@` в секции `ХРАНИЛИЩЕ_ОБЪЕКТОВ` мы можем получить следующий варианта задания

```
При следующем значении объектов-параметров =
    имя файла : input.txt;
    количество : 113;

Получим следующее задание =
    Создайте файл с именем input.txt и запишите в него 113 случайных чисел
```

Описание первых блоков шаблон-файла

Синтаксис всех блоков одинаковый :

```
блок = “----{-}” конец_строки название_блока конец_строки содержимое_блока
конец_строки = ‘\n’
название_блока = “ХРАНИЛИЩЕ_ОБЪЕКТОВ” | “ШАБЛОННЫЙ_ВИД” | “РЕШЕНИЕ” | “СЛУЖЕБНОЕ”
| “ТЕСТОВЫЕ_ДАННЫЕ”
содержимое_блока = “осмысленный набор символов”
```

Описанное выше является сырым/техническим описанием блоков, далее будет представлен наглядный пример описания первых блоков:

```
----
ХРАНИЛИЩЕ_ОБЪЕКТОВ
имя файла : input.txt;
количество : 113;

----
ШАБЛОННЫЙ_ВИД
Создайте файл с именем @имя файла@ и запишите в него @количество@ случайных чисел
```

Выше было описан простой пример, а если вернуться к первоначальному заданию (расположено на самом верху), формат первых двух секций выглядит следующим образом:

```
----
ХРАНИЛИЩЕ_ОБЪЕКТОВ
выражение 1 : #genAE(5);
выражение 2 : #genAE(5);
выражение 3 : #genAE(5);
выражение 4 : #genAE(5);
```

```
1-ый диапазон : #rnd( 0.0  | 5.0  | double);
2-ой диапазон : #rnd( 5.0  | 10.0 | double);
3-ий диапазон : #rnd( 10.0 | 15.0 | double);

----
ШАБЛОННЫЙ_ВИД
написать программу для вычисления выражения
y = @выражение 1@ где x1 < @1-ый диапазон@
y = @выражение 2@ где @1-ый диапазон@ <= x1 < @2-ой диапазон@
y = @выражение 3@ где @2-ой диапазон@ <= x1 < @3-ий диапазон@
y = @выражение 4@ где x1 >= @3-ий диапазон@
```

Секция 3 - РЕШЕНИЕ

`РЕШЕНИЕ` - хранит эталонное решения для задания. Эталонное решение компилируется и используется для тестирования других программ предназначенных для выполнения поставленных задач.

Данная секция является копией `ШАБЛОННОГО_ВИДА`, здесь также можно использовать объекты-параметры, с тем отличием что данные (если быть точнее код) из этого блока должны компилироваться или интерпретироваться.

Дополним пример выше секцией с эталонным решением:

```
----
ХРАНИЛИЩЕ_ОБЪЕКТОВ
выражение 1 : #genAE(5);
код 1 : #getAEcode();

выражение 2 : #genAE(5);
код 2 : #getAEcode();

выражение 3 : #genAE(5);
код 3 : #getAEcode();

выражение 4 : #genAE(5);
код 4 : #getAEcode();

1-ый диапазон : #rnd( 0.0  | 5.0  | double);
2-ой диапазон : #rnd( 5.0  | 10.0 | double);
3-ий диапазон : #rnd( 10.0 | 15.0 | double);

----
ШАБЛОННЫЙ_ВИД
написать программу для вычисления выражения
y = @выражение 1@ где x1 < @1-ый диапазон@
y = @выражение 2@ где @1-ый диапазон@ <= x1 < @2-ой диапазон@
y = @выражение 3@ где @2-ой диапазон@ <= x1 < @3-ий диапазон@
y = @выражение 4@ где x1 >= @3-ий диапазон@

----
РЕШЕНИЕ
```

```
#include <cmath>
#include <iostream>

using namespace std;

int main()
{
    double y = 0.0;
    double x1;
    cin >> x1;

    if ( x1 <  @1-ый диапазон@)
    {
        @код 1@
    }
    else if (x1 >=  @1-ый диапазон@ && x1 < @2-ой диапазон@)
    {
        @код 2@
    }
    else if (x1 >= @2-ой диапазон@ && x1 < @3-ий диапазон@)
    {
        @код 3@
    }
    else if (x1 >= @3-ий диапазон@)
    {
        @код 4@
    }
    cout << y << "\n";
    system("pause");
    return 0;
}
----
```

Секция 4 - СЛУЖЕБНОЕ

СЛУЖЕБНОЕ - секция отвечающая за настройку генератора. На данный момент есть лишь 2 параметра

- знаки_арифм – отвечает за символы, которые будут использоваться во время генерации арифметических выражений;
- функции_арифм – отвечает за функции, которые будут использоваться во время генерации арифметических выражений.

Стандратный вид данной секции :

```
----
СЛУЖЕБНОЕ
знаки_арифм : +, -, *, /, ^;
функции_арифм : sin, cos, sqrt, tan, log, log10;
```

Секция 5 - ТЕСТОВЫЕ_ДАННЫЕ

ТЕСТОВЫЕ_ДАННЫЕ - секция которая содержит данные для тестирования эталонного решения и решения, присланного обучаемым, на корректность;

Данный блок схож с ХРАНИЛИЩЕ_ОБЪЕКТОВ, здесь используются объекты-параметры и значения присваемые им. Можно указывать заготовленные данные (имеется ввиду любой тип данных) через запятую или же использовать какие-либо ффункции, например использовать функцию для генерации большого количества случайных чисел в диапазоне и т.д.

Стандратный вид данной секции :

```
----
ТЕСТОВЫЕ_ДАННЫЕ
Тест1 : #rnd(100 | 200 | int | 1000);
Тест2 : 1,2,3,4,5,6,7,8,9,10;
```

Давайте дополним шаблон для изначального задания до конца:

```
----
ХРАНИЛИЩЕ_ОБЪЕКТОВ
выражение 1 : #genAE(5);
код 1 : #getAEcode();

выражение 2 : #genAE(5);
код 2 : #getAEcode();

выражение 3 : #genAE(5);
код 3 : #getAEcode();

выражение 4 : #genAE(5);
код 4 : #getAEcode();

1-ый диапазон : #rnd( 0.0  | 5.0  | double);
2-ой диапазон : #rnd( 5.0  | 10.0 | double);
3-ий диапазон : #rnd( 10.0 | 15.0 | double);

----
ШАБЛОННЫЙ_ВИД
написать программу для вычисления выражения
y = @выражение 1@ где x1 < @1-ый диапазон@
y = @выражение 2@ где @1-ый диапазон@ <= x1 < @2-ой диапазон@
y = @выражение 3@ где @2-ой диапазон@ <= x1 < @3-ий диапазон@
y = @выражение 4@ где x1 >= @3-ий диапазон@

----
РЕШЕНИЕ
#include <cmath>
#include <iostream>

using namespace std;
```

```
int main()
{
    double y = 0.0;
    double x1;
    cin >> x1;

    if ( x1 <  @1-ый диапазон@)
    {
        @код 1@
    }
    else if (x1 >=  @1-ый диапазон@ && x1 < @2-ой диапазон@)
    {
        @код 2@
    }
    else if (x1 >= @2-ой диапазон@ && x1 < @3-ий диапазон@)
    {
        @код 3@
    }
    else if (x1 >= @3-ий диапазон@)
    {
        @код 4@
    }
    cout << y << "\n";
    system("pause");
    return 0;
}
----
СЛУЖЕБНОЕ
знаки_арифм : +, -, *, /, ^;
функции_арифм : sin, cos, sqrt, tan, log, log10;
----
ТЕСТОВЫЕ_ДАННЫЕ
Тест1 : #rnd(100 | 200 | int | 1000);
Тест2 : 1,2,3,4,5,6,7,8,9,10;
----
```