

УДК 681.3.06

## ОРГАНИЗАЦИЯ РАБОТЫ С МНОЖЕСТВАМИ В ИНСТРУМЕНТАЛЬНОЙ СРЕДЕ РАЗРАБОТКИ ПРОГРАММНЫХ МОДУЛЕЙ

А.Д. Ковалев<sup>1</sup>, С.В. Никитин<sup>1</sup>

Рассматриваются классы множеств, применяемые для создания программных систем, обеспечивающих поддержку динамических древовидных и сетевых структур данных. Излагаются формы представления множеств и операции над ними. Обсуждаются примеры работы с неупакованными и упакованными списками номеров элементов, а также с неупакованными битовыми шкалами.

**Ключевые слова:** классы множеств, древовидные структуры данных, сетевые структуры данных, инструментальные программные средства.

Множества широко используются в командах Главного управляющего языка инструментальной среды для создания систем обработки и отображения сложно организованной информации [1], разрабатываемой в НИВЦ МГУ. Особенно активно они применяются в процессе создания программных модулей, обеспечивающих поддержку динамических древовидных и сетевых структур данных [2].

В Главном управляющем языке реализованы три вида множеств:

- неупакованный список номеров элементов;
- неупакованная битовая шкала;
- упакованный список номеров элементов.

Список номеров и битовая шкала имеют разное функциональное назначение. Список номеров используется для хранения результатов выборки и представляет собой упорядоченное множество, к которому может быть применена процедура сортировки. Битовая шкала, используемая для выполнения логических операций над множествами и анализа состояния определенных элементов, не упорядочена. Упакованный список номеров элементов предназначен для более компактного хранения данных. Он не используется непосредственно при анализе и отборе элементов структур и в операциях с множествами.

Множество представляет собой структуру данных, состоящую из служебного заголовка (длиной четыре байта) и тела. Младшие 29 бит заголовка содержат целое число без знака, которое в зависимости от типа множества имеет различный смысл. Для упакованного и неупакованного списков номеров элементов это число означает количество элементов в списке. Для неупакованной битовой шкалы это число означает размер битовой шкалы. В трех старших битах заголовка хранится информация о типе множества.

Команды Главного управляющего языка, предназначенные для работы с множествами, позволяют выполнять следующие операции:

- операции порождения множеств;
- операции с элементами множества;
- операции с битовыми шкалами;
- преобразование множеств из одного типа в другой.

Порождение множеств выполняется командой `set_conv`, имеющей следующий синтаксис:

```
set_conv bv_set type size,
```

где `bv_set` — имя переменной, обозначающей порождаемое множество;

`type` — тип порождаемого множества:

- `uindex` — неупакованный список номеров элементов;
- `ubit` — неупакованная битовая шкала;
- `pindex` — упакованный список номеров элементов;

`size` — размер множества.

В обозначении типа множества допускается использование только первых двух букв.

Если в качестве типа множества в команде `set_conv` указано `ubit` или `ub`, то в переменной `bv_set` создается пустая битовая шкала размером `size` бит. Все биты новой шкалы имеют значение ноль. В случае множеств типа неупакованного (`type=uindex`) или упакованного списков номеров (`type=pindex`), создается список из `size` элементов с номерами от 0 до (`size-1`). Например, команда

<sup>1</sup> Научно-исследовательский вычислительный центр, Московский государственный университет им. М.В. Ломоносова, 119899, Москва; e-mail: kovalev\_ad@list.ru

```
set_conv unl ui 100
```

создает неупакованный список номеров 0, 1, 2, ... 97, 98, 99, состоящий из 100 элементов, в переменной unl.

Команда

```
set_conv uba ub 100
```

создает в переменной uba пустую битовую шкалу размером 100 бит.

Для выполнения операций с элементами множества используются команды `set_func` и `set_int_let`. Первая команда выполняет операции над одним элементом множества, в то время как вторая работает с интервалом номеров. Команда `set_func` имеет следующий синтаксис:

```
set_func bv_set op [ind],
```

где `bv_set` — имя переменной, обозначающей множество;

`op` — строка из одного символа, которая может принимать следующие значения:

`i` — выдача информации о множестве;

`@` — проверка наличия элемента множества в множестве;

`|` — добавление элемента в множество;

`-` — удаление элемента из множества;

`^` — добавление элемента в множество, если этот элемент отсутствует, и удаление, если он присутствует.

`ind` — номер элемента множества (если команда применяется для получения информации о множестве (`op=i`), то параметр `ind` не используется).

Для получения полной информации о множестве используется команда `set_func` при `op=i`. В результате выполнения команды системная переменная `RET_CODE` будет содержать количество элементов для неупакованного (упакованного) списка номеров или размер шкалы для множеств типа битовой шкалы. В системную переменную `RET_STR` записывается строка с названием типа множества: `uindex`, `pindex` или `ubit`.

Например, если после рассмотренной выше команды создания неупакованного списка из 100 элементов

```
set_conv unl ui 100
```

выполнить команду

```
set_func unl i,
```

то системная переменная `RET_CODE` будет содержать число 100, а в переменную `RET_STR` будет записана строка “`uindex`”.

Если в команде `set_func` значение `op` равно `@`, то предоставляется информация об отдельных элементах внутри множества, при этом проверяется либо наличие определенных элементов в неупакованном и упакованном списках, либо состояние бита с номером `ind` в битовой шкале. Для списка номеров элементов параметр `ind` представляет собой значение проверяемого элемента.

Если множество является списком элементов, то в системную переменную `RET_CODE` записывается 1, когда элемент со значением `ind` присутствует в списке, и 0 — в противном случае. Для битовой шкалы переменная `RET_CODE` содержит информацию о бите с порядковым номером `ind`. Это свойство выполняется для всех значений `op`, кроме `op=i`.

Операция добавления (`op=|`) позволяет дополнять указанное множество новыми элементами. Параметр `ind` играет разную роль в зависимости от типа множества. Для неупакованной битовой шкалы он является порядковым номером бита, значение которого устанавливается равным единице. Нумерация битов ведется с нуля. Для списков номеров параметр `ind` представляет собой значение элемента, заносимое в список.

Операция удаления элементов из множества устанавливает значение бита с порядковым номером `ind` равным нулю для неупакованной битовой шкалы, и удаляет элемент, значение которого равно `ind`, для списков элементов.

В случае битовой шкалы операция добавления элемента интерпретируется как логическое сложение данной шкалы со шкалой, у которой бит с номером `ind` содержит 1, а остальные биты установлены равными 0. Операция удаления элемента в этом случае представляет собой логическое умножение данной шкалы на шкалу, у которой бит с номером `ind` равен нулю, а остальные биты равны единице.

Операции добавления/удаления элементов могут использоваться в цикле для генерации множеств, имеющих регулярную структуру.

В качестве примера рассмотрим набор команд для создания пустого неупакованного списка и дополнения его номерами в следующей последовательности: 5, 10, 15, 20, ... . Предположим, что размер множества задается переменной `size=100`. Тогда сначала создается пустой список номеров элементов

```
set_conv unl ui 0,
```

а затем этот список дополняется в цикле номерами 5, 10, 15, ... :

```
= i 1
```

```
#:Label_1
set_func un1 | &( &i. 5 * &)
= i &( &i. 1 + &)
if_then Label_1 i > &size
```

Здесь параметру цикла *i* присваивается начальное значение 1. Значения элементов множества задаются выражением  $i \times 5$ . При каждом проходе значение параметра цикла увеличивается на 1. Выход из цикла происходит в том случае, когда выполняется условие  $i > \text{size}$ .

Особенностью выражений в Главном управляющем языке является использование обратной польской записи. Выражение заключается в операторные скобки, обозначаемые символами `&` и `&`. При записи выражения сначала указываются операнды, а затем знак операции.

Исключим из полученного списка элементы, имеющие, например, значения 10, 20, 30, ... :

```
= i 1
#:Label_2
set_func un1 - &( &i. 10 * &)
= i &( &i. 1 + &)
if_then Label_2 i > &hsize
```

В результате получится множество 5, 15, 25 ... .

Операция `~` проверяет наличие в множестве элемента, заданного параметром *ind*. Если такой элемент есть, то он удаляется из списка. В противном случае указанный в команде `set_func` элемент добавляется в множество. Для неупакованной битовой шкалы эта операция представляет собой логическую операцию “исключающее ИЛИ” данной шкалы со шкалой, у которой бит с порядковым номером *ind* равен 1, а остальные биты равны 0.

Выполним операцию `~` над множеством, полученным в предыдущем примере:

```
set_func un ~ 10
```

В результате выполнения этой команды в неупакованный список *un1* будет добавлен элемент со значением 10, поскольку такого элемента не было в списке. Следует заметить, что новые элементы добавляются всегда в конец списка.

Команда `set_int_let` похожа на команду `set_func`, но выполняет операции над интервалом номеров. Она имеет следующий синтаксис:

```
set_int_let bv_set op ind_min ind_max,
```

где *bv\_set* — имя переменной, обозначающей множество;

*op* — строка из одного символа, которая может принимать следующие значения:

- = — присваивание;
- | — логическое сложение (ИЛИ);
- @ — логическое умножение (И);
- — операция обнуления;
- ^ — операция “исключающее ИЛИ”;

*ind\_min* — номер начального элемента интервала;

*ind\_max* — номер конечного элемента интервала.

В случае списка номеров в данной команде используется только операция присваивания.

Операция `=` заполняет множество элементами, значения которых задаются интервалом номеров от *ind\_min* до *ind\_max*. Для битовой шкалы эта операция аналогична операции логического сложения `|`. Она устанавливает все биты с порядковыми номерами от *ind\_min* до *ind\_max* равными 1.

Например, создадим пустую битовую шкалу *ubs* длины *sizeb* со значениями битов 3, 4, 11, 12, 19, 20, ... равными единице (это — средние два бита внутри каждого байта) с помощью команды операций над интервалом номеров `set_int_let`:

```
= i 1
#:Label_3
set_int_let ubs = &( &i. 8 * 5 - &). &( &i. 8 * 4 - &)
= i &( &i. 1 + &)
if_then Label_3 i > &sizeb
```

Заметим, что того же результата можно было бы достичь с помощью команды `set_func`, но это было бы менее эффективно.

Операция `|` выполняет логическое сложение данной битовой шкалы с битовой шкалой того же размера,

у которой биты с указываемыми номерами `ind_min, ..., ind_max` установлены равными 1. Она идентична операции присваивания `=`.

Операция `@` логически умножает данную битовую шкалу на шкалу, у которой все биты равны 0, за исключением битов с указанными номерами `ind_min, ..., ind_max`. Эта операция позволяет “вырезать” требуемую часть исходной битовой шкалы.

Операция `-` выполняет в исходной битовой шкале обнуление битов с номерами `ind_min, ..., ind_max`. Для иллюстрации установим равными 0 биты 4, 12, 20, ... из предыдущего примера:

```
= i 1
#:Label_4
set_int_let uba - &( &i. 8 * 4 - &). &( &i. 8 * 1 - &)
= i &( &i. 1 + &)
if_then Label_4 i > &sizeb
```

Операция “исключающее ИЛИ” оказывает действие на биты с номерами `ind_min, ..., ind_max` исходной шкалы. Если бит содержал 1, то он устанавливается равным 0, и наоборот.

В результате выполнения приведенной ниже последовательности команд над битовой шкалой из предыдущего примера получится шкала, у которой биты 2, 10, 18, ... будут установлены равными 1, а остальные будут равны нулю:

```
= i 1
#:Label_5
set_int_let uba ^ &( &i. 8 * 6 - &). &( &i. 8 * 5 - &)
= i &( &i. 1 + &)
if_then Label_5 i > &sizeb
```

Для выполнения операций над битовыми шкалами используются команды `set_let` и `set_map_let`. Первая команда выполняет логические операции над битовыми шкалами одного размера, а вторая команда используется для выполнения операций над битовыми шкалами разного размера. Команда `set_let` имеет следующий синтаксис:

```
set_let bv_setr op bv_set1 [bv_set2],
где bv_setr — имя переменной с результирующей битовой шкалой;
op — код операции:
| — логическое сложение (ИЛИ);
@ — логическое умножение (И);
- — обнуление;
^ — исключающее ИЛИ;
~ — инверсия;
```

`bv_set1` - имя переменной с первой битовой шкалой;  
`bv_set2` - имя переменной со второй битовой шкалой.

Команда `set_let` предназначена для выполнения логических операций над битовыми шкалами одинакового размера. В результате операции в переменной, имя которой задается в параметре `bv_setr`, образуется результирующая битовая шкала, размер которой равен размеру битовых шкал операндов. Операция (инверсия) одноместная, поэтому параметр `bv_set2` для данной операции не используется.

Предположим, что переменная `uba` содержит шкалу, у которой биты 0, 2, 4, 6, 8, ... установлены равными 1, а остальные равны 0. В результате выполнения операции инверсии

```
set_let ubar ~ uba
```

в результирующей шкале `ubar` биты с нечетными порядковыми номерами будут установлены равными 1, а значения остальных битов будут равны 0.

Если вторым символом кода операции `op` является символ `=`, то параметр `bv_set2` для данной операции не используется. В качестве второго операнда используется результирующая битовая шкала и в нее же записывается результат операции.

Сложим битовые шкалы `uba` и `ubar` из предыдущего примера:

```
set_let | = uba
```

В результате выполнения этой команды в переменной `ubar` получится битовая шкала, у которой все биты полагаются равными 1.

Команда `set_map_let` используется для выполнения операций над битовыми шкалами разного размера. Она имеет следующий синтаксис:

```
set_map_let bv_setr op bv_sets bv_mapr bv_maps,
```

`noindent` где `bv_setr` — имя переменной с результирующей битовой шкалой;

`op` — код операции:

- = — присваивание;
- | — логическое сложение (ИЛИ);
- @ — логическое умножение (И);
- — гашение;
- ^ — исключающее ИЛИ;

`bv_sets` — имя переменной с битовой шкалой;

`bv_mapr` — имя переменной со списком номеров соответствия для результирующей битовой шкалы;

`bv_mapa` — имя переменной со списком номеров соответствия для битовой шкалы операнда.

Параметры `bv_mapr` и `bv_mapa` задают списки номеров элементов для результирующей битовой шкалы и битовой шкалы операнда, по которым устанавливается соответствие между битовыми шкалами разного размера. Списки номеров элементов `bv_mapr` и `bv_mapa` должны содержать одинаковое количество элементов. Заметим, что соответствие между номерами элементов позиционное.

В качестве примера рассмотрим заполнение одной битовой шкалы с помощью маски, заданной другой шкалой. Приводимые ниже команды Главного управляющего языка сопровождаются комментариями. Строка комментария начинается с символа `*` в первой позиции.

```
* Создаем пустые битовые шкалы ubsr размером lsize бит
* для размещения результатов операций и ubss размером
* 8 бит для размещения операнда
  set_conv ubsr ub &lsize
  set_conv ubss ub 8
* Создаем списки соответствия номеров шкалы-результата и
* шкалы-операнда
  set_conv unlr ui 0
  set_conv unls ui 0
* Заполняем шкалу-операнд номерами 2 3 4
  set_int_let ubss = 2 4
* Заполняем список номеров шкалы-операнда номерами
* 0 1 2 ... 5 6 7
  set_int_let unls = 0 7
* Заполняем шкалу-результат с помощью шкалы-операнда
  = i 1
#:Label_6
  set_conv unlr ui 0
  set_int_let unlr = (&i. 1 - 8 * &). (&i. 8 * 1 - &)
  set_map_let ubsr = ubss unlr unls
  = i (&i. 1 + &)
  if_then Label_6 i > &sizeb
```

В цикле каждый байт шкалы `ubsr` заполняется маской, заданной в шкале `ubss`.

Преобразование типа множества выполняется командой `set_conv`, которая имеет следующий синтаксис:

```
set_conv bv_setr type size bv_sets,
```

где `bv_setr` — имя переменной с результирующим множеством;

`type` — тип результирующего множества: `uindex` — неупакованный список номеров элементов;

`pindex` — упакованный список номеров элементов;

`ubit` — неупакованная битовая шкала;

`size` — размер множества;

`bv_sets` — имя переменной с исходным множеством.

Вместо полного названия типа множества допускается использование двух первых букв.

Рассмотрим взаимные преобразования неупакованного списка номеров `unl` в упакованный список `pnl`.

```
* Преобразуем неупакованный список из lsize элементов в
* упакованный
  set_conv pnl pi &lsize unl
* Выполняем обратное преобразование упакованного списка
* в неупакованный
  set_conv unlc ui &lsize pnl
```

Множество, полученное в переменной `unlc`, идентично исходному множеству `unl`.

На практике часто используется генерация битовой шкалы из некоторого регулярного списка номеров, выполняемая командой

```
set_conv uba ub &sizeb unl
```

Здесь `unl` — каким-либо образом сформированный список номеров, а `uba` — результирующая битовая шкала.

#### СПИСОК ЛИТЕРАТУРЫ

1. Арушанян О.Б., Богомолов Н.А., Волченкова Н.И., Ковалев А.Д., Синицын М.Н. Общее описание Главного управляющего языка Базовой технологии создания систем обработки и отображения сложно организованной информации // Библиотеки и пакеты прикладных программ. М.: Изд-во Моск. ун-та, 1996.
1. Богомолов Н.А., Ковалев А.Д., Никитин В.В. Встроенная база данных инструментального графического пакета // Конструирование библиотек и пакетов программ. М.: Изд-во Моск. ун-та, 1995.

Поступила в редакцию  
31.01.2000

---