Руководство по использованию языка HSCP v1.2.2

Причины создания и краткий обзор языка:

Как известно, для написания SC-программ используется язык SCP. Однако, описание программ на нём затруднительно по нескольким причинам:

1. Программа на SCP – часть базы знаний, а следовательно, необходимо *вручную* описывать все сущности программы и вязи между ними
2. Ввиду большого количества исходного кода, высок шанс, сделать опечатку или ввести неправильные отношения. Интерпретатор языка, далеко не всегда уведомляет об этом.
3. Существует составные (состоящие из нескольких операторов) конструкции, которые требуют много кода, при этом употребляются часто, что приводит к ещё большему увеличению объема исходного кода.

HSCP (High-level SCP) предлагает более высокоуровневый синтаксис и наличие конструкций, которые описывает сразу несколько операторов SCP. Также имеется определенное количество проверок на уровне анализа кода, которые помогают избегать некоторых ошибок.

Самое важное свойство HSCP в том, что это – транслируемый в SCP язык, что позволяет использовать HSCP везде, где доступно использование SCP.

Содержание:

Процедуры:

Для описания процедуры на языке HSCP, используется код следующей формы:

procedure <procedure_name>($param1, $param2, …)

Где *procedure_name* – имя процедуры, а *$param1* – имя одного из её параметров. Также после параметра может стоять символ ссылки *&* - при его наличии, параметр интерпретируется, как выходящий, иначе – входящий.

После заголовка процедуры, может идти либо блок высказываний (block of statements), либо точка с запятой. Так же сразу после списка парметров может находиться ключевое слово *implicit* – оно делает имя процедуры видимым только в пределах одного файлы (транслирует имя с двумя точками перед ним).

Если после процедуры идёт блок – он интерпретируется, как тело процедуры. Если же – точка с запятой – декларация процедуры интерпретируется, как предварительная декларация (forward declaration). Это нужно, для того чтобы указать транслятору, на наличие процедуры объявленной в другом файле.

```
//forward declaration
procedure someProc();
```

Процедура должна содержать оператор *return*, для возврата из неё.

Имена:

Переменные и процедуры должны иметь имена, которые состоят из цифр букв английского алфавита верхнего и нижнего регистра, знака нижнего подчеркивания (underscore). Первым символом не должна быть цифра. Имена начинающиеся на два знака нижнего подчеркивания '__' зарезервированы для нужд реализации. Имена процедур транслируются без изменений.

Комментарии:

Язык поддерживает два вида комментариев:

```
//однострочный комментарий

/*мно-
 -гострочный
 комментарий
*/
```

Переменные:

Для того чтобы объявить переменную, необходимо предварить её имя оператором *new*.

```
new $some_variable;
```

Также язык поддерживает составное объявление переменных (compound declaration). При таком объявлении вы можете сразу задать значение переменной.

```
new $awesome_var = rrel_arcs;
```

Виды значений:

1. Константа – описывается как системный идентификатор элемента. Возможно использование … для создания анонимного элемента (константного узла). (В таком случае генерируется код эквивалентный *genEl(new $var{_e, _a})*).
   ```
   new $c = some_identifier;
   ```

2. Строковый литерал – заключённое в двойные кавычки, значение, содержащее любые символы, помимо двойных кавычек. Если же, необходимо хранить среди символов строки и двойные кавычки, нужно предварить их обратным слешем.
   ```
   new $s = "some string";
   ```

3. Другая переменная – в качестве значения, можно использовать ранее объявленные переменные.
   ```
   new $v = $another_var;
   ```

4. *null* – для того чтобы сбросить значение переменной, необходимо присвоить ей null. При этом на самом деле, переменной не присваивается никакого значение, следовательно на уровне SCP, переменная – пуста.

Чтобы присвоить значение объявленной переменной, необходимо сначала указать переменную, которой будет присвоено значение, затем, после знака равно '=', присваиваемое значение.
$var = const_value;

Также переменные имееют область видимости. Область видимости ограничивается блоком в котором они были объявлены и всеми вложеннными блоками. Переменные не могут быть использованы вне области видимости и не могут быть объявлены в области видимости больше чем один раз. Переменные должны быть объявлены до первого использования. Если переменная объявлена во время перечисления аргументов вызова (call) (не if3/5/e и не for3/5), её областью видимости считается блок, в котором происходит вызов.

Условные конструкции:

Всего доступно три вида условных конструкций: *if3* , *if5* и *ife.*

if3 имеет форму:

```
if3 ($begin; new $arc; $end)
        //then statement
[else [if3/if5(...)]
        //else statement
]
```

Принимает три аргумента описывающих некоторую конструкцию в SC-памяти. Если соответствующая конструкция, будет найдена, выполняется высказывание - *then statement*, иначе – *else statement*.

Примечание: высказыванием (statement) – может выступать как блок, так и отдельный оператор (даже просто ;).

Ключевое слово *else* может быть не представлено. Елси оно наличствует, то после него, могут идти другие условия – *if3/if5.* Эти условия проверяются, если искомая конструкция не была найдена, и соответствующий блок выполняется, если выполнено соответствующее условие.

if5 – отличается от *if3* – лишь тем, что принимает пять аргументов и ищет соответственно пятиэлементные конструкции.

Отношение *rrel_assign* сгенерируется автоматически на тех аргументах, чьи переменные были объявлены до условия. *rrel_fixed* – на тех, что были объявлены прямо в списке аргументов.

Если после *if* перед условием стоит восклицательный знак '!' условие инвертируется.

Конструкция *ife*:

```
ife($elem1; $elem2)
        //then statement
else
        //else statement
```

Данная конструкция проверяет равенство первого и второго аргумента и, если они равны, переходит к *then-statement*, иначе *else-statement*.

Аргументы:

Аргументом считается любое значение, помимо *null*, описанное в списке аргументов. Если переменнная была объявлена в месте описания аргумента, она считается аргументом. При этом нельзя использовать составное объявление.

Переменные, когда используются в качестве аргументов, могут иметь модификаторы. Чтобы указать модификаторы необходимо после имени переменной в фигурных скобках, через запятую перечислить константы, описывающие модификаторы:

some_call($var{rrel_some_rel, rrel_fixed});

Данные модификаторы будут транслированы в отношения к аргументу, при его трансляции. Доступны также предопределённые модификаторы (predefined modifiers):

1. _a – транслируется в *rrel_assign*
2. _f – транслируется в *rrel_fixed*
3. _a_p – транслируется в *rrel_pos_const_perm*
4. _a_c – транслируется в *rrel_common*
5. _e – транслируется в *rrel_node*, вместе с *rrel_const*

Имена данных модификаторов начинаются на знак нижнего подчеркивания. Если был использован предопределенный модификатор который не указан выше, произойдет ошибка лексического анализа. В связи с этим, рекомендуется предпочесть обычным модификаторам, предопределенные.

Для аргументов-переменных, будет автоматически сгенерировано отношение *rrel_scp_var*, для аргументов-констант *rrel_scp_const* и *rrel_fixed*.

Если в качестве аргумента выступает не-ссылочный параметр процедуры, то для него автоматически генерируется отношение *rrel_fixed*.

Вместо аргумента может идти ключевое слово *auto* – оно работает таким образом, как если бы вместо *auto* была бы подставлена, анонимная переменная. Для *auto* доступны модификаторы, также для auto автоматически генерируется *rrel_assign*.

Примечание: для всех аргументов for3, for5, if3 и if5 явно заданные модификаторы _f, a, rrel_assign, rrel_fixed удаляются.

Циклы:

Имеется два вида циклов *for3* и *for5*. Общий вид *for3*:

for3($used; new $arc; $elem)[new $unused]
//loop statement

Данный цикл изначально запишет все *искомые элементы* в множество *$unused*. Затем, на каждой итерации, он будет записывать в переменную, олицетворяющую искомый элемент. После выполнения *loop-statement* данный элемент будет удалён из *$unused*. Переменная для еще необработанных элементов является аргументом. Она может отсутствовать, тогда пользователь не будет иметь доступа к множеству необработанных элементов.

По умолчанию, искомым элементом считается **третий** элемент. Если же, для переменной необработанных элементов задан модификатор rrel_set_X, то искомым элементом будет считаться элемент номер X (начиная с 1).

for5 отличается от *for3* наличием пяти аргументов вместо трех.

Также, доступна мягкая модификация циклов (soft loop). Она включается, если после названия цикла, перед списком аргументов поставить плюс ‘+’. Её суть состоит в следующем:

предполагается, что элемент на который ссылается, итерационная переменная, будет удален вручную. Это полезно при очистке памяти.

Вызовы:

Вызов состоит из имени *вызываемой сущности* затем списка аргументов и модификаций вызова.

callable_entity($some_var) async;

В качестве вызываемой сущности может выступать имя нативно-поддерживаемого оператора, или объявленной (с помощью, предварительного объявления, например) процедуры.

Нативно поддерживаемые операторы:

- *searchElStr3*
- *searchElStr5*
- *genEl*
- *genElStr3*
- *genElStr5*
- *eraseEl*
- *eraseElStr3*
- *eraseElStr5*

Если же, необходимо вызвать какой-либо оператор, не поддерживающийся нативно, нужно указать модификацию вызова *force,* после списка аргументов.

Если происходит вызов, процедуры, то по умолчанию он – синхронный, чтобы сделать его асинхронным – необходимо указать модификацию доступа *async* после списка аргументов.

HSCPT:

Для того, чтобы произвести трансляцию, необходимо запустить транслятор hsctp, со следующими параметрами:

./hscpt –build-dir “директория в которую будут записаны файлы scs” --src-dir “директория с исходными файлами HSCP” --build

Исходные файлы должны иметь расширение “.hscp”. Структура директорий из папки с исходным кодом, сохраняется в папке со сборкой.

Внимание: *директория сборки будет полностью очищена перед сборкой*

Пример:

```
//dfs: arcs – дуги графа, $current – текущая вершина, $used – предварительно
инициализированное пустое множество,

procedure dfs($arcs, $current, $used) {
   if3 !($used; auto{_a_p}; $current) {
      genElStr3($used, auto{_a_p}, $current);

      for5($current; auto{_a_c}; new $elem; auto{_a_p}; $arcs)
         dfs($arcs, $elem, $used);
   }

   return;
}
```