Основы языка SimpleSCP

Параметры

Для того, чтобы передать в SimpleSCP-функцию параметры, нужно в сигнатуре функции указать идентификаторы:

function example(имена, параметров)

Здесь *имена параметров* – перечисление идентификаторов параметров через запятую.

Для того, чтобы объявить в SimpleSCP-функции выходные параметры, нужно в теле функции указать:

return идентификатор;

Здесь *идентификатор* – название выходного параметра. Конструкций такого вида в программе может быть несколько.

Примеры правильного объявления входных и выходных параметров SimpleSCP-функции:

Пример SimpleSCP	**Соответствующий код SCP**

function example(_set) {
  return _set;
  return _answer;
}
	-> rrel_params: ... (*
	  -> rrel_1: rrel_in: _set;;
	  -> rrel_1: rrel_out: _set;;
	  -> rrel_2: rrel_out: _answer;;
	*);;

Тело программы

Тело SimpleSCP-программы состоит из линейных операторов и конструкций управления потоком.

Типичный линейный оператор языка выглядит следующим образом:

название(аргументы);

Где *название* – это ключевое слово языка SimpleSCP либо идентификатор пользовательской функции, а *аргументы* – конструкции вида:

[модификаторы, название] или []

Тут *название* – идентификатор используемой константы или переменной, состоящий из нижних подчеркиваний, букв латинского алфавита и цифр, либо строковый литерал, взятый в кавычки. *Модификаторы* – список ключевых слов, обозначающих роль аргумента в операторе. Модификаторы могут быть назначены по умолчанию. Кроме того, аргумент может быть пустым – тогда при трансляции в SCP он не будет использован в операторе.

Примеры правильного вызова операторов SimpleSCP можно увидеть ниже:

Пример SimpleSCP	**Соответствующий код SCP**

searchElStr3(
  [_set],
  [assign, _arc],
  [_element]
);
	-> ..operator1 (*
	  <- searchElStr3;;
	  -> rrel_1: rrel_fixed: rrel_scp_var: _set;;
	  -> rrel_2: rrel_scp_var: rrel_assign: _arc;;
	  -> rrel_3: rrel_fixed: rrel_scp_var: _element;;
	*);;

sys_search(
  [search_pattern],
  [assign, _result],
  [parameters],
  [assign, _all_elements]
);
	-> ..operator1 (*
	  <- sys_search;;
	  -> rrel_1: rrel_fixed: rrel_scp_const: search_pattern;;
	  -> rrel_2: rrel_scp_var: rrel_assign: _result;;
	  -> rrel_3: rrel_fixed: rrel_scp_const: parameters;;
	  -> rrel_4: rrel_scp_var: rrel_assign: _all_elements;;
	*);;

eraseEl([erase, _element]);	-> ..operator1 (*

	<- eraseEl;; -> rrel_1: rrel_fixed: rrel_scp_var: rrel_erase: _element;; *);;
genSetStr3([_node1], [assign, _arc], [assign, _node2], [], [], [assign, _set]);	-> ..operator1 (* <- genSetStr3;; -> rrel_1: rrel_fixed: rrel_scp_var: _node1;; -> rrel_2: rrel_scp_var: rrel_assign: _arc;; -> rrel_3: rrel_scp_var: rrel_assign: _node2;; -> rrel_set_3: rrel_assign: rrel_scp_var: _set;; *);;
proc_of_user_function([_user_argument], [nrel_some_relation]);	-> ..operator1 (* <- call;; -> rrel_1: rrel_fixed: rrel_scp_const: proc_of_user_function;; -> rrel_2: ... (* -> rrel_1: rrel_fixed: rrel_scp_var: _user_argument;; -> rrel_2: rrel_fixed: rrel_scp_const: nrel_some_relation;; *);; -> rrel_3: rrel_scp_var: rrel_assign: _process;; => nrel_goto: ..operator2;; *);; -> ..operator2 (* <- waitReturn;; -> rrel_1: rrel_fixed: rrel_scp_var: _process;; *);;

Конструкциями управления потоком в SimpleSCP являются условия и циклы.
Условные конструкции используются следующим образом:

if (оператор) {действия; если;}
else {действия; в; другом; случае;}

Здесь *оператор* – линейный оператор языка SimpleSCP, *действия если* – набор операторов, которые необходимо выполнить в случае выполнения условия, *действия в другом случае* - набор операторов, которые необходимо выполнить в другом случае.
Пример правильного использования условной конструкции

Пример SimpleSCP

if(searchElStr3([some_class], [assign, _arc], [_node])) {
proc_do_some_actions([_node]);
}

Соответствующий код SCP
-> ..operator1 (* <- searchElStr3;; -> rrel_1: rrel_fixed: rrel_scp_const: some_class;; -> rrel_2: rrel_scp_var: rrel_assign: _arc;; -> rrel_3: rrel_fixed: rrel_scp_var: _node;; => nrel_then: ..operator2;; => nrel_else: ..operator4;; *);; -> ..operator2 (* <- call;; -> rrel_1: rrel_fixed: rrel_scp_const: proc_do_some_actions;;

```
  -> rrel_2: ... (*
    -> rrel_1: rrel_fixed: rrel_scp_var: _node;;
  *);;
  -> rrel_3: rrel_scp_var: rrel_assign: _process;;
  => nrel_goto: ..operator3;;
*);;
-> ..operator3 (*
  <- waitReturn;;
  -> rrel_1: rrel_fixed: rrel_scp_var: _process;;
*);;
```

Циклы с предусловием в SimpleSCP описываются следующим образом:

while(оператор) {действия;}

Здесь *оператор* – линейный оператор языка SimpleSCP, *действия* – набор операторов, которые необходимо выполнять до тех пор, пока выполняется условие.

Пример правильного использования циклической конструкции

Пример SimpleSCP

```
 while(searchElStr3([_set], [assign, _arc], [assign, _node])){
    printEl([_node]);
    proc_of_some_actions([_node]);
    eraseEl([erase, _arc]);
  }
```

Соответствующий код SCP

```
-> ..operator1 (*
  <- searchElStr3;;
  -> rrel_1: rrel_fixed: rrel_scp_var: _set;;
  -> rrel_2: rrel_scp_var: rrel_assign: _arc;;
  -> rrel_3: rrel_scp_var: rrel_assign: _node;;
  => nrel_then: ..operator2;;
  => nrel_else: ..operator6;;
*);;
-> ..operator2 (*
  <- printEl;;
  -> rrel_1: rrel_fixed: rrel_scp_var: _node;;
  => nrel_goto: ..operator3;;
*);;
-> ..operator3 (*
  <- call;;
  -> rrel_1: rrel_fixed: rrel_scp_const: proc_of_some_actions;;
  -> rrel_2: ... (*
    -> rrel_1: rrel_fixed: rrel_scp_var: _node;;
  *);;
  -> rrel_3: rrel_scp_var: rrel_assign: _process;;
  => nrel_goto: ..operator4;;
*);;
-> ..operator4 (*
  <- waitReturn;;
  -> rrel_1: rrel_fixed: rrel_scp_var: _process;;
  => nrel_goto: ..operator5;;
*);;
```

```
-> ..operator5 (*
  <- eraseEl;;
  -> rrel_1: rrel_fixed: rrel_scp_var: rrel_erase: _arc;;
  => nrel_goto: ..operator1;;
*);;
```

Приложение 1. Список операторов языка SimpleSCP
(Добавляются по мере необходимости)

Оператор	**Количество аргументов**
onepamop(rrel_1, rrel_2, rrel_3 ...)	
searchEl	1
searchElStr3	3
searchElStr5	5
genEl	1
genElStr3	3
genElStr5	5
eraseEl	1
eraseElStr3	3
eraseElStr5	5
sys_search	4
print	1
printNl	1
printEl	1
ifVarAssign	1
onepamop(rrel_1, rrel_2, rrel_3 …, rrel_set_1, rrel_set_2, rrel_set_3 ...)	
searchSet	2
searchSetStr3	6
searchSetStr5	10
genSet	2
genSetStr3	6
genSetStr5	10
eraseSet	2
eraseSetStr3	6
eraseSetStr5	10

Приложение 2. Модификаторы аргумента SimpleSCP
(Добавляются по мере необходимости)

fixed
assign
pos_const_perm
node
arc
erase
scp_var
scp_const

Приложение 3. Модификаторы аргумента по умолчанию.

Роль	Модификаторы
Константа	fixed, scp_const
Переменная	fixed, scp_var

Приложение 4. Примеры программ SimpleSCP.

SimpleSCP

```
function example1() {
  print(["Hello world!"]);
}
```

SCP

```
scp_program -> example1 (*
  -> rrel_params: ... (*
  *);;
  -> rrel_operators: ... (*
    -> rrel_init: ..operator1 (*
      <- print;;
      -> rrel_1: rrel_fixed: rrel_scp_const: [Hello world!];;
      => nrel_goto: ..operator2;;
    *);;
    -> ..operator2 (*
      <- return;;
    *);;
  *);;
*);;
```

SimpleSCP

```
function example2(_scp_program) {
  searchSetStr5(
    [_scp_program],
    [assign, _tmp_arc_1],
    [assign, _pattern],
    [assign, _tmp_arc_2],
    [nrel_error_pattern],
```

```
    [], [], [assign, _patterns], [], []
  );
  while(searchElStr3([patterns], [assign, _arc], [assign, _pattern])) {
    sys_search(
      [_pattern],
      [assign, _result],
      [pattern_params],
      [assign, _all_elements]
    );
    if(ifVarAssign([_result])) {
      proc_generate_error([_scp_program]);
      print(["Error!"]);
    }
    eraseEl([erase, _arc]);
  }
}
```

SCP

```
scp_program -> example2 (*
  -> rrel_params: ... (*
    -> rrel_1: rrel_in: _scp_program;;
  *);;
  -> rrel_operators: ... (*
    -> rrel_init: ..operator1 (*
      <- searchSetStr5;;
      -> rrel_1: rrel_fixed: rrel_scp_var: _scp_program;;
      -> rrel_2: rrel_scp_var: rrel_assign: _tmp_arc_1;;
      -> rrel_3: rrel_scp_var: rrel_assign: _pattern;;
      -> rrel_4: rrel_scp_var: rrel_assign: _tmp_arc_2;;
      -> rrel_5: rrel_fixed: rrel_scp_const: nrel_error_pattern;;
      -> rrel_set_3: rrel_scp_var: rrel_assign: _patterns;;
      => nrel_goto: ..operator2;;
    *);;
    -> ..operator2 (*
      <- searchElStr3;;
      -> rrel_1: rrel_fixed: rrel_scp_const: patterns;;
      -> rrel_2: rrel_scp_var: rrel_assign: _arc;;
      -> rrel_3: rrel_scp_var: rrel_assign: _pattern;;
      => nrel_then: ..operator3;;
      => nrel_else: ..operator10;;
    *);;
    -> ..operator3 (*
      <- sys_search;;
      -> rrel_1: rrel_fixed: rrel_scp_var: _pattern;;
      -> rrel_2: rrel_scp_var: rrel_assign: _result;;
      -> rrel_3: rrel_fixed: rrel_scp_const: pattern_params;;
      -> rrel_4: rrel_scp_var: rrel_assign: _all_elements;;
      => nrel_goto: ..operator4;;
    *);;
    -> ..operator4 (*
      <- ifVarAssign;;
      -> rrel_1: rrel_fixed: rrel_scp_var: _result;;
```

```
    => nrel_then: ..operator5;;
    => nrel_else: ..operator8;;
  *);;
  -> ..operator5 (*
    <- call;;
    -> rrel_1: rrel_fixed: rrel_scp_const: proc_generate_error;;
    -> rrel_2: ... (*
      -> rrel_1: rrel_fixed: rrel_scp_var: _scp_program;;
    *);;
    -> rrel_3: rrel_scp_var: rrel_assign: _process;;
    => nrel_goto: ..operator6;;
  *);;
  -> ..operator6 (*
    <- waitReturn;;
    -> rrel_1: rrel_fixed: rrel_scp_var: _process;;
    => nrel_goto: ..operator7;;
  *);;
  -> ..operator7 (*
    <- print;;
    -> rrel_1: rrel_fixed: rrel_scp_const: [Error!];;
    => nrel_goto: ..operator8;;
  *);;
  -> ..operator8 (*
    <- print;;
    -> rrel_1: rrel_scp_const: rrel_fixed: [...];;
    => nrel_goto: ..operator9;;
  *);;
  -> ..operator9 (*
    <- eraseEl;;
    -> rrel_1: rrel_fixed: rrel_scp_var: rrel_erase: _arc;;
    => nrel_goto: ..operator2;;
  *);;
  -> ..operator10 (*
    <- print;;
    -> rrel_1: rrel_scp_const: rrel_fixed: [...];;
    => nrel_goto: ..operator11;;
  *);;
  -> ..operator11 (*
    <- return;;
  *);;
 *);;
*);;
```