21. Control flow actions

This clause describes the ***e*** control flow actions.

21.1 Conditional actions

Conditional actions are used to specify code segments that execute only when a specific condition is met.

21.1.1 if then else

Purpose	Perform an action block if a given Boolean expression is TRUE or a different action if the expression is FALSE	
Category	Action	
Syntax	**if** *bool-exp* [**then**] **{***action*; ...**}** [**else if** *bool-exp* [**then**] **{***action*; ...**}**] [**else {***action*; ...**}**]	
Parameters	*bool-exp*	A Boolean expression.
	action; ...	A list of zero or more actions separated by semicolons (`;`) and enclosed in braces (`{}`).

If the first *bool-exp* is TRUE, the **then** *action* block is executed. If the first *bool-exp* is FALSE, the **else if** clauses are executed sequentially: if an **else if** *bool-exp* is found that is TRUE, its **then** *action* block is executed; otherwise, the final **else** *action* block is executed.

Because **if then else** is a single action, no semicolons (`;`) should appear between **if** and **else**, unless they are required to separate two or more actions within one of the action blocks.

NOTE—Since **else if then** clauses can be used for multiple Boolean checks (comparisons), consider using a **case** *bool-case-item* action (see 21.1.3) when there are a large number of comparisons to perform.

Syntax example:

```
if a > b then {
    print a, b
} else if a == b then {
    print a
} else {
    print b, a
}
```

21.1.2 case labeled-case-item

Purpose	Execute an action block based on whether a given comparison is TRUE	
Category	Action	
Syntax	**case** *case-exp* **{***labeled-case-item*; ... [**default**[**:**] **{***default-action*; ...**}**]**}**	
Parameters	*case-exp*	A legal ***e*** expression.
	labeled-case-item	*label-exp*[:] *action-block* where *label-exp* is a legal e expression or an enumerated constant range, as follows: if *case-exp* is of a numeric type, *label-exp* must be of a numeric type, or of a numeric list type, or of the set type if *case-exp* is of an enumerated type, *label-exp* must be of the same or comparable enumerated type, or of a list type thereof, or it must be a range of enumerated item constants thereof if *case-exp* is of another type, *label-exp* must be of a comparable type or of a list type thereof *action-block* is a list of zero or more actions separated by semicolons and enclosed in braces. Syntax: {*action*;...} The entire *labeled-case-item* is repeatable, not just the *action-block* related to the *label-exp*.
	default-action; ...	A list of zero or more actions separated by semicolons (`;`) and enclosed in braces (`{}`).

This evaluates the *case-exp* and executes the first *action-block* for which *label-exp* matches the *case-exp*. If no *label-exp* matches the *case-exp*, it executes the *default-action* block, if specified.

Whether or not a label-exp matches the case-exp is determined as follows:

— If *case-exp* and *label-exp* are of comparable types, that is, the equality operator (==) is applicable to two operands of these types, then matching is determined by applying the equality operator to the two expressions: *label-exp* matches the *case-exp* if *case-exp*==*label-exp* returns TRUE.

— Otherwise, matching is determined by applying the inclusion operator (**in**) to the two expressions: *label-exp* matches the *case-exp* if *case-exp* **in** *label-exp* returns TRUE.

After an *action-block* is executed, the ***e*** program proceeds to the line that immediately follows the entire **case** statement.

Syntax example:

```
case packet.length {
    64          : {out("minimal packet")       };
    [65..256]   : {out("short packet")         };
    [257..512]  : {out("long packet")          };
    default     : {out("illegal packet length")}
}
```

21.1.3 case bool-case-item

Purpose	Execute an action block based on whether a given comparison is TRUE	
Category	Action	
Syntax	**case** {*bool-case-item*; ... [**default** {*default-action*; ...}]}	
Parameters	*bool-case-item*	*bool-exp*[:] *action-block* where *bool-exp* is a Boolean expression *action-block* is a list of zero or more actions separated by semicolons and enclosed in braces. Syntax: {*action*;...} The entire *bool-case-item* is repeatable, not just the *action-block* related to the *bool-exp*.
	default-action; ...	A list of zero or more actions separated by semicolons (;) and enclosed in braces ({}).

This evaluates the *bool-exp* conditions one after the other and executes the *action-block* associated with the first TRUE *bool-exp*. If no *bool-exp* is TRUE, it executes the *default-action-block*, if specified. After an *action-block* is executed, the ***e*** program proceeds to the line that immediately follows the entire case statement.

Each of the *bool-exp* conditions is independent of the other *bool-exp* conditions and there is no main *case-exp* to which all cases refer, unlike the case labeled-case-item (see 21.1.2).

NOTE—This case action has the same functionality as a single **if then else** action, where each *bool-case-item* is specified as a separate **else if then** clause.

Syntax example:

```
case {
    packet.length == 64        {out("minimal packet")};
    packet.length in [65..255] {out("short packet")  };
    default                    {out("illegal packet")}
}
```

21.2 Iterative actions

Iterative actions are used to specify code segments that execute in a loop, for multiple times, in a sequential order.

NOTE 1—A **repeat until** action performs the action block at least once. A **while** action might not perform the action block at all.

NOTE 2—The optional **do** syntax used in some of the constructs of this subclause is purely syntactic sugar.

21.2.1 while

Purpose	Execute an action block repeatedly as long as a Boolean expression evaluates to TRUE	
Category	Action	
Syntax	**while** *bool-exp* [**do**] **{***action; ...***}**	
Parameters	*bool-exp*	A Boolean expression.
	action; ...	A list of zero or more actions separated by semicolons (`;`) and enclosed in braces (`{}`).

This executes the *action* block repeatedly in a loop while *bool-exp* is TRUE. This construct can be used to set up a perpetual loop as `while TRUE {}`. The loop shall not execute at all if the *bool-exp* is FALSE when the **while** action is encountered.

Syntax example:

```
while a < b do {
    a += 1
}
```

21.2.2 repeat until

Purpose	Execute an action block repeatedly as long as a Boolean expression evaluates to FALSE	
Category	Action	
Syntax	**repeat {***action*; ...**} until** *bool-exp*	
Parameters	*action*; ...	A list of zero or more actions separated by semicolons (`;`) and enclosed in braces (`{}`).
	bool-exp	A Boolean expression.

This executes the *action* block repeatedly in a loop until *bool-exp* is TRUE. The action block is executed at least once.

Syntax example:

```
repeat {
    i += 1
} until i == 3
```

21.2.3 for each in

Purpose	Execute an action block once for every element of a list expression	
Category	Action	
Syntax	**for each** [*type*] [(*item-name*)] [**using index (***index-name***)**] **in** [**reverse**] *list-exp* [**do**] **{***action*; ...**}**	
Parameters	*type*	A type of the struct comprising the list specified by *list-exp*. Elements in the list shall match this type.
	item-name	The name of the current item in *list-exp*. If this parameter is not specified, the item can be referenced using the implicit variable **it**.
	index-name	The name of the index of the current list item. If this parameter is not specified, the item can be referenced using the implicit variable **index**.
	list-exp	An expression that results in a list.
	action;...	A list of zero or more actions separated by semicolons (`;`) and enclosed in braces (`{}`).

For each item in *list-exp*, if its type matches *type*, this executes the *action* block. Inside the *action* block, the implicit variable **it** (if no *item-name* is specified) or the optional *item-name* (when specified) refers to the matched item, and the implicit variable **index** (or the optional *index-name*) reflects the index of the current item. If **reverse** is specified, *list-exp* is traversed in reverse order, from last to first. The implicit variable **index** (or the optional *index-name*) starts at zero (`0`) for regular loops and at `list.size()-1` for reverse loops.

Each **for each in** action defines two new local variables for the loop, named by default **it** and **index**. The following restrictions also apply:

a) When loops are nested inside one another, the local variables of the internal loop hide those of the external loop. To overcome this, assign each *item-name* and *index-name* unique names.

b) Within the action block, a value cannot be assigned to **it** or **index**—or to *item-name* or *index-name*.

Syntax example:

```
for each transmit packet (tp) in sys.pkts do {
    print tp            // "transmit packet" is a type
}
```

21.2.4 for each in set

Purpose	Execute an action block once for every element included in a set	
Category	Action	
Syntax	**for each** [*type*] [(*item-name*)] **in** [**reverse**] **set** *set-exp* [**do**] **{***action*; ...**}**	
Parameters	*type*	A numeric type.
	item-name	The name of the current set element. If this parameter is not specified, the element can be referenced using the implicit variable **it**.
	set-exp	An expression that results in a set.
	action;...	A list of zero or more actions separated by semicolons (`;`) and enclosed in braces (`{}`).

For each numeric included in *set-exp*, this executes the *action* block. If *type* is specified, the element is automatically cast to *type*; otherwise, an unbounded integer is used. Inside the *action* block, the implicit variable **it** (if no *item-name* is specified) or the optional *item-name* (when specified) refers to the element. If **reverse** is specified, the elements are traversed in decreasing order; otherwise, they are traversed in increasing order.

Each **for each in set** action defines a new local variable for the loop, named by default **it**. The following restrictions also apply:

a) When loops are nested inside one another, the local variable of the internal loop hide that of the external loop. To overcome this, assign each *item-name* a unique name.

b) Within the action block, a value cannot be assigned to **it** or to *item-name* .

Syntax example:

```
for each uint (n) in my_set do {
    print n
}
```

21.2.5 for from to

Purpose	Execute a for loop for the number of times specified	
Category	Action	
Syntax	**for** *var-name* **from** *from-exp* [**down**] **to** *to-exp* [**step** *step-exp*] [**do**] **{***action*; ...**}**	
Parameters	*var-name*	A temporary variable of type **int**.
	from-exp, *to-exp*, *step-exp*	Valid ***e*** expressions that resolve to type **int**. The default value for *step-exp* is `1`.
	action; ...	A list of zero or more actions separated by semicolons (`;`) and enclosed in braces (`{}`).

This creates a temporary variable *var-name* of type **int** and repeatedly executes the *action* block while incrementing (or decrementing if **down** is specified) its value from *from-exp* to *to-exp* in interval values specified by *step-exp* (which defaults to `1`), i.e., the loop is executed until the value of *var-name* is greater than the value of *to-exp* or less than *to-exp*.

The temporary variable *var-name* is visible only within the **for from to** loop where it was created.

Syntax example:

```
for i from 5 down to 1 do {
    out(i)
}           // Outputs 5,4,3,2,1
```

21.2.6 for

Purpose	Execute a C style for loop	
Category	Action	
Syntax	**for {***initial-action; bool-exp; step-action***}** [**do**] **{***action*; ...**}**	
Parameters	*initial-action*	An action.
	bool-exp	A Boolean expression
	step-action	An action.
	action;...	A list of zero or more actions separated by semicolons (`;`) and enclosed in braces (`{}`).

The **for** loop works similarly to the `for` loop in the C language. This **for** loop executes the *initial-action* once and then checks the *bool-exp*. If the *bool-exp* is `TRUE`, it executes the *action* block followed by the *step-action*. It repeats this sequence in a loop for as long as *bool-exp* is `TRUE`. The following restrictions also apply:

a) When a loop variable is used within a **for** loop, it needs to be declared before the loop (unlike the temporary variable of type **int** automatically declared in a **for from to** loop).

b) Although this action is similar to a C-style `for` loop, the *initial-action* and *step-action* need to be ***e*** style actions.

Syntax example:

```
for i from 5 down to 1 do {
    out(i)
}           // Outputs 5,4,3,2,1
```

21.3 File iteration actions

This subclause describes *loop constructs*, which are used to manipulate general ASCII files.

21.3.1 for each line in file

Purpose	Iterate a for loop over all lines in a text file	
Category	Action	
Syntax	**for each** [**line**] [(*name*)] **in file** *file-name-exp* [**do**] **{***action*; ...**}**	
Parameters	*name*	Variable referring to the current line in the file.
	file-name-exp	A string expression that gives the name of a text file.
	action; ...	A list of zero or more actions separated by semicolons (`;`) and enclosed in braces (`{}`).

This executes the *action* block for each line in the text file *file-name*. Inside the block, **it** (if no name is specified) or *name* (if specified) refers to the current line (as a string) without the final line feed (`\n`) character.

NOTE—The optional **line** syntax is purely syntactic sugar.

Syntax example:

```
for each line in file "signals.dat" do {
    '(it)' = 1
} // Reads a list of signal names and assigns to each the value 1
```

21.3.2 for each file matching

Purpose	Iterate a for loop over a group of files	
Category	Action	
Syntax	**for each file** [(*name*)] **matching** *file-name-exp* [**do**] **{***action*; ...**}**	
Parameters	*name*	Variable referring to the current line in the file.
	file-name-exp	A string expression that gives the name of a text file.
	action; ...	A list of zero or more actions separated by semicolons (`;`) and enclosed in braces (`{}`).

For each file (in the file search path) whose name matches *file-name-exp*, this executes the *action* block. Inside the block, **it** (if no *name* is specified) or *name* (if specified) refers to the matching file name.

Syntax example:

```
for each file matching "*.e" do {
    out(it)
} // lists the 'e' files in the current directory
```

21.4 Actions for controlling the program flow

These actions alter the flow of the program in places where the flow would otherwise continue differently.

21.4.1 break

Purpose	Break the execution of a loop
Category	Action
Syntax	**break**

This breaks the execution of the nearest enclosing iterative action (**for** or **while)**. When a **break** action is encountered within a loop, the execution of actions within the loop is terminated and the next action to be executed is the first one following the loop.

break actions shall not be placed outside the scope of a loop (or the compiler shall report an error).

Syntax example:

```
break
```

21.4.2 continue

Purpose	Stop executing the current loop iteration and start executing the next loop iteration
Category	Action
Syntax	**continue**

This stops the execution of the nearest enclosing iteration of a **for** or **while** loop, and continues with the next iteration of the same loop. When a **continue** action is encountered within a loop, the current iteration of the loop is aborted, and execution continues with the next iteration of the same loop.

continue actions shall not be placed outside the scope of a loop (or the compiler shall report an error).

Syntax example:

```
continue
```