

18. Control flow actions 1

This clause describes the following control flow actions:

- Conditional actions 5
- Iterative actions
- File iteration actions
- Actions for controlling the program flow 10

18.1 Conditional actions

Conditional actions are used to specify code segments which execute only when a specific condition is met. 15

18.1.1 if then else 15

Purpose	Perform an action block if a given Boolean expression is TRUE or a different action if the expression is FALSE 20				
Category	Action				
Syntax	if <i>bool-exp</i> [then] { <i>action</i> ; ...} [else if <i>bool-exp</i> [then] { <i>action</i> ; ...}] [else { <i>action</i> ; ...}]				
Parameters	<table style="width: 100%; border: none;"> <tr> <td style="border: none;"><i>bool-exp</i></td> <td style="border: none;">A Boolean expression. 25</td> </tr> <tr> <td style="border: none;"><i>action</i>; ...</td> <td style="border: none;">A list of zero or more actions separated by semicolons (;) and enclosed in curly braces ({}).</td> </tr> </table>	<i>bool-exp</i>	A Boolean expression. 25	<i>action</i> ; ...	A list of zero or more actions separated by semicolons (;) and enclosed in curly braces ({}).
<i>bool-exp</i>	A Boolean expression. 25				
<i>action</i> ; ...	A list of zero or more actions separated by semicolons (;) and enclosed in curly 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. 30

Because the **if then else** clause comprises one action, the semicolon (;) comes at the end of the clause and not after each action block within the clause; do not put a semicolon (;) between the closing curly bracket (}) for the action block and the **else** keyword. 35

NOTE—Since the **else if then** clauses can be used for multiple Boolean checks (comparisons), consider using a **case bool-case-item** action (see 18.1.3) when there are a large number of comparisons to perform. 40

Syntax example: 40

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

45

50

55

18.1.2 case labeled-case-item

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

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

Syntax example:

```

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

```

18.1.3 case bool-case-item

Purpose	Execute an action block based on whether a given comparison is TRUE
Category	Action
Syntax	<code>case {bool-case-item; ... [default {default-action; ...}]}</code>
Parameters	<i>bool-case-item</i> <i>bool-exp</i> [:] <i>action-block</i> where <i>bool-exp</i> is a Boolean expression <i>action-block</i> is a list of zero or more actions separated by semicolons and enclosed in curly braces. Syntax: { <i>action</i> ;...} The entire <i>bool-case-item</i> is repeatable, not just the <i>action-block</i> related to the <i>bool-exp</i> .
	<i>default-action</i> ; ... A list of zero or more actions separated by semicolons (;) and enclosed in curly 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 18.1.2).

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"); };
};

```

18.2 Iterative actions

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

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

18.2.1 while

Purpose	Execute an action block repeatedly as long as a Boolean expression evaluates to TRUE	
Category	Action	
Syntax	while <i>bool-exp</i> [do] { <i>action</i> ; ...}	
Parameters	<i>bool-exp</i>	A Boolean expression.
	<i>action</i> ; ...	A list of zero or more actions separated by semicolons (;) and enclosed in curly 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 {a += 1};
```

18.2.2 repeat until

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

18.2.3 for each in

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

18.2.4 for from to

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

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
```

18.2.5 for

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

- 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).
- 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=0; i<=10; i+=1} do {out(i);};
```

18.3 File iteration actions

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

18.3.1 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 <i>file-name-exp</i> [do] { <i>action</i> ; ...}
Parameters	<i>name</i> Variable referring to the current line in the file.
	<i>file-name-exp</i> A string expression that gives the name of a text file.
	<i>action</i> ; ... A list of zero or more actions separated by semicolons (;) and enclosed in curly braces ({}).

This executes the *action* block for each line in the text file *file-name*. Inside the block, **it** (or *name*) refers to the current line (as a string) without the final \n (the final new line [CR]) or EOF (end of file) character.

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
```

18.3.2 for each file matching

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

For each file (in the file search path) whose name matches *file-name-exp*, this executes the *action* block. Inside the block, **it** (or *name*) refers to the matching file name.

Syntax example:

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

18.4 Actions for controlling the program flow

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

18.4.1 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
```

18.4.2 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
```