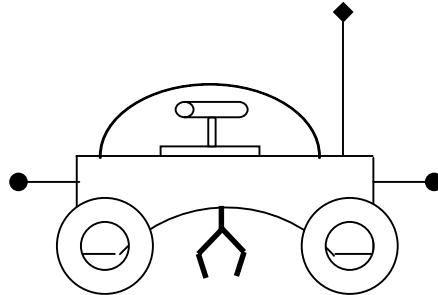


Robbie the Robot

Robbie is the entry-level model in the RUR line, offering simple mobility and a single claw for grabbing and carrying objects that weigh up to 10 kg.

Robbie may be controlled by programs written in the Robbie Language, described later in this document. Robbie can store and execute Robbie Language programs of any length. Unfortunately, Robbie does not deal well at all with incorrect programs. In fact, if Robbie attempts to execute a program that contains a syntax error, Robbie will explode. Such failures are NOT covered by the warranty, nor is RUR liable for the resulting contamination.

Robbie Features



Robbie is powered by a small quantum reactor, which provides enough energy for Robbie to operate for an essentially unlimited amount of time. For movement, Robbie has four independent drive wheels, which incorporate the latest in high traction technology, enabling Robbie to climb surfaces at any angle (including ceilings). Robbie has two sensors, mounted fore and aft, for detecting contact with immovable objects.

Robbie has a dorsal control module, which includes optical sensors and communications antennas. A ventrally mounted claw allows Robbie to pick up objects that lie beneath his central body, carry those objects, and put them down. The claw can be extended up to one meter, and can lift and carry weights up to 10 kg. An integrated scale allows Robbie to determine the weight of an object, in 1g increments.

Robbie also includes a simple sound system, which can generate frequencies in the range 60Hz to 30,000Hz.

A Sample Program in Robbie Language

The Robbie Language, RL for short, allows the user to program Robbie to perform a variety of tasks. RL bears some resemblance to traditional high-level languages, such as Pascal or C, but is much simpler. The following RL program causes Robbie to pace back and forth in a straight line forever (not terribly useful, really):

```

program "Pace Forever"      ; 1:  program name
start                       ; 2:  begin executing the program
  while true do             ; 3:  do the following instruction forever:
    forward 50              ; 4:   move forward 50 cm
    turn left 180           ; 5:   turn right 180 degrees (about-face)
  endwhile                 ; 6:
stop                        ; 7:  stop executing the program

```

RL programs may contain comments as shown above. Any text that follows a semi-colon on a line is considered to be a comment (useful to human readers, but not part of the instructions Robbie will execute).

Line 1 begins the RL program, specifying a name for the program. The name has no special significance in RL, and is actually optional, but it is good practice to give RL programs descriptive names.

Lines 2 and 7 mark the beginning and end of execution of the RL program body. Both are required for every RL program. An RL program may contain multiple `stop` instructions, but only one `start` instruction.

Line 3 begins a `while` loop, and line 6 marks the end of the `while` loop. The body of the `while` loop (the instructions between lines 3 and 6 here, is executed repeatedly as long as the condition specified in the `while` loop header (line 3) is true. Lines 4 and 5 specify the actions that Robbie should take during each pass through the `while` loop.

This example program illustrates only a small part of the RL language. The following section provides a more detailed look at the various elements of RL.

RL: the Robbie Language

Typographical convention: RL reserved words are shown in `Courier` font, expressions that must be defined by the user are shown in brackets: `<expression>`.

RL includes two commands for starting and stopping the execution of a RL program:

```
start
stop
```

RL includes three simple movement commands:

```
forward <integer distance>
back    <integer distance>
turn    [ left | right ]   <integer rotation>
```

The `forward` command causes Robbie to move forward the specified distance (in cm), if possible. If Robbie encounters an obstruction, he will stop moving forward. The `back` command is similar. The `turn` command causes Robbie to turn the specified angle (in degrees) in the specified direction. Robbie cannot be blocked from turning through any specified angle. Distance and rotation values must be non-negative.

Examples:

```
forward 20      ; move forward 20 cm
back 8         ; move backward 8 cm
turn left 45   ; turn 45 degrees to the left
```

RL includes two commands for the claw hand:

```
grab
drop
```

The first causes Robbie to extend his claw downward until an object is found (or until it cannot be extended further) and to close the claw to pick up an object and retract to its original location. The second causes Robbie to put the object down (gently despite the name), and retract the claw. If there is no object, then the claw will return to its original location. If there is an object, but it exceeds the maximum weight Robbie can lift, the claw automatically releases the object and retracts to its original location.

RL includes two tests that use Robbie's internal sensors:

```
blocked
weight
```

The former is true if there is an object blocking Robbie's path in the forward direction, and false otherwise. The second equals the weight of the object the claw is holding, to the nearest gram. If the claw is not holding an object, the weight test will equal zero.

RL includes two miscellaneous commands:

```
beep   <time>      <frequency>
pause  <time>
```

The first causes Robbie to emit a sound for the specified time (in milliseconds) at the specified frequency (in Hertz). The second causes Robbie to pause all activities for the specified time.

Examples:

```
beep 2000 4000 ; beep at 4000 Hz for 2 seconds
pause 500      ; pause for 0.5 seconds
```

Robbie includes an essentially unlimited number of internal registers, or memory locations, each of which can be used to store a single value. RL programs can include named variables, each of which is associated with one of Robbie's registers. Registers can hold integer values, real numbers (decimal values), character strings, or Boolean values (true or false). Before using a variable in an RL program, you must "register" it:

```
register [ integer | real | boolean | string ] <name>
```

The command causes Robbie to associate one of his registers with the given name. The name can then be used in the program and Robbie will automatically store its value in the associated register. Variable names may consist of any sequence of letters and numbers; it is strongly recommended that variable names NOT be entirely lower case.

Examples:

```
register integer Counter
register real NetPay
register boolean Empty
register string CompetitorName
```

RL uses several reserved words, and arithmetic symbols, for Boolean expressions and operations. A Boolean expression is simply an expression whose value is true or false, rather than numerical. Any of the standard arithmetic symbols may be used:

```
+      -      /      *      <      <=     >      >=     =
```

The usual precedence rules apply, and parentheses may be used for grouping. In addition, the following reserved words may be used to combine Boolean terms:

```
and      or      not
```

Also, the reserved words `true` and `false` may be used with their usual significance.

RL provides the `set` command for assigning a value to a variable:

```
set   <name>      <value>
```

The name must be that of a variable that has already been registered, and the value may be any valid algebraic or arithmetic expression, including the use of registered variable names.

Examples:

```
set Counter 0
set Empty true
set CompetitorName "ACME Robot Corporation"
```

Finally, RL provides control structures. To select between different courses of action, RL provides the if-then command:

```
if <Boolean expression> then
  <commands>
endif
```

and the if-then-else command:

```
if <Boolean expression> then
  <commands>
else
  <commands>
endif
```

The first form allows the user to program Robbie to decide whether or not to carry out a command, or list of commands, depending on whether the Boolean expression is true when the if-then statement is reached. The second form allows Robbie to choose between two possible courses of action.

Examples:

```
if Empty then
  grab
endif

if Counter < 10 then
  grab
  forward 100
  drop
else
  beep 1000 80
endif
```

RL also provides two control structures for repetition of commands. The while-do command causes Robbie to carry out the commands in its body as long as the Boolean expression is true:

```
while <Boolean expression> do
  <commands>
endwhile
```

Example:

```
while true do
  forward 50
  turn left 90
  forward 15
  turn left 90
endwhile
```

The repeat command causes Robbie to carry out the command in its body the specified number of times:

```
repeat <integer expression> times
  <commands>
endrepeat
```

Example:

```
repeat 10 times
  forward 2
  beep 500 1000
  back 1
  beep 500 2000
endrepeat
```

In RL, spacing is up to the user, as long as things that should be separate are not “run together”. Unlike some programming languages, RL does not have any special notation to mark the end of a command. Instead, each line that is not blank or a comment, or a control structure line, is treated as being a single command.

RL Reserved Words

A reserved word is a word that has a special meaning in the language and cannot be used for any other purpose. RL uses the following reserved words (all lower case):

and	endrepeat	pause	string
back	false	program	then
beep	forward	real	times
blocked	grab	register	true
do	if	repeat	turn
drop	integer	right	weight
else	left	set	while
endif	not	start	
endwhile	or	stop	

Users may not use any of these words as variable names. Because all RL reserved words are lower case, it is recommended that users use mixed case or upper case variable names. This will help to avoid accidental collisions with the reserved word list, and make it easier to identify user variables in an RL program.

More Sample Programs

The following sample programs illustrate some of Robbie's capabilities.

This makes Robbie pace in a 50 by 15 cm rectangle forever:

```
program "Pace Forever in a Rectangle"
start
  while true do
    forward 50
    turn left 90
    forward 15
    turn left 90
  endwhile
stop
```

This makes Robbie pace back and forth on a 50 cm line ten times:

```
program "Pace Laps"
start
  register integer Laps      ; register a counter variable
  set Laps 0                 ; set it to zero initially

  while Laps < 10 do        ; do these commands until Laps
                            ; reaches ten
    forward 50              ; move forward 1/2 meter
    turn left 180           ; reverse course
    set Laps  Laps + 1      ; count this lap
  endwhile
stop
```

This makes Robbie pick up and move ten items a distance of one meter:

```
program "Move Ten Things"
start
  register integer numMoved  ; to count the items
  set numMoved 0             ;

  while numMoved < 10 do
    grab                    ; pick up an item
    turn left 180           ; turn around
    forward 100              ; move 1 meter forward
    drop                    ; put the item down
    turn left 180           ; turn back
    forward 100              ; move back to starting point
    set numMoved  numMoved + 1 ; count the item
  endwhile

  beep 10000 50             ; beep 10 seconds at 50 Hz
stop
```

Here, we assume that each item to be moved will be at the same location initially, perhaps being fed down an assembly line, and that each is dropped at the same location.

This makes Robbie demand attention until an object is available for him to pick up:

```
program "Bother"
start
  register boolean Ignored      ; is there an item yet?
  set Ignored true             ; not yet . . .
  register integer Beeps       ; number of beeps issued
  set Beeps 0                   ;

  while Ignored do             ; keep it up 'til you get an item

    set Beeps Beeps + 1        ; count beep

    if Beeps < 3 then          ; short beep first two times
      beep 1000 500
    else                        ; then out of patience
      beep 5000 500
    endif

    grab                       ; try to pick up item

    if weight > 0 then         ; see if you got one
      set Ignored false       ; if so, quit the loop
    endif

  endwhile
stop
```

Here, we assume that each item to be moved will be at the same location initially, perhaps being fed down an assembly line, and that each is dropped at the same location.