Tiny BASIC language exercises (ver 1)

Task 1:

Assignment:
Create a program to calculate the area of a rectangle. When the program is run, the user will be prompted to enter the lengths of the sides. Then the area will be printed. Calculate everything with integer values.

Advice:
Use the PRINT and INPUT statements. The area of a rectangle is calculated as the product of the lengths of its sides.

Solution example:

```
0 REM Calculate area of rectangle
1 PRINT “Enter length of sides:\n“
2 INPUT A; INPUT B
3 PRINT “The area is: “ , A*B
```

Solution notes:
The program could be written on one line:

```
1 INPUT A; INPUT B; PRINT “The area is: “ , A*B
```

Task 2:

Assignment:
Write a program to calculate the area of a circle to 2 decimal places. When the program is run, the user is prompted to enter the radius of the circle and then the area is printed.

Advice:
Use the PRINT, LET, IF-THEN, and INPUT statements. Tiny BASIC only calculates with whole numbers. Use the value of π as 314 to calculate the area, and then display the result in sequence. First, print the whole value obtained by dividing by 100, the decimal point symbol, and then the remainder after dividing by 100. Remember that the remainder after division must be displayed with the first zero - e.g. 08.
The formula for calculating the area is known:

$$S = \pi r^2$$

Solution example:

```
0 REM Calculate area of circle
1 PRINT “Enter radius:\n“
2 INPUT R
3 LET S=R*R*314; LET C=S/100
4 PRINT “The area is: “ , C, “.“
5 LET Z=S-C*100
6 IF Z<10 THEN PRINT “0“
7 PRINT Z
```

Solution notes:
Tiny BASIC does not have an operation for powers of numbers. That is why the square power is written as R*R (line 3). And it also does not work with decimal numbers. Multiplying by 314 instead of 3.14 gives a result 100 times larger. To get the whole part of the result, we must divide S by 100 (line 3). The remainder, which we calculate after subtracting the integer part multiplied by 100, must then be displayed, including any leading zero (line 6).

Task 3:

Assignment:
Create a program that prints all odd numbers up to 20.

Advice:
Use PRINT, LET, GOTO, and IF-THEN statements.

Solution example:

```
0 REM Odd numbers up to 20
1 LET A=1
2 PRINT A, “\n“
3 LET A=A+2
4 IF A<20 THEN GOTO 2
```

Solution notes:
This is a very simple counting loop. We start at 1 (line 1) and gradually print the numbers on separate lines (line 2) in increments of 2 (line 3). The evaluation condition (line 4) returns the program to print the number until 20 is reached.

Task 4:

Assignment:
Create a program that draws a rectangular triangle of the characters ***** 10 lines high on the screen.

Advice:
Use PUTCH, LET, GOTO, and IF-THEN statements. The newline character has the value 10 and the ***** character has the value 42.

Solution example:

```
0 REM Triangle *
1 PUTCH 10
2 LET A=1; REM Count lines
3 LET B=1; REM Stars on line
4 PUTCH 42; LET B=B+1
5 IF B<=A THEN GOTO 4
6 PUTCH 10; LET A=A+1
7 IF A<11 THEN GOTO 3
```

Solution notes:
There are 2 nested counting loops used here. In variable A, the lines are counted (lines 2,6,7) and in variable B, the characters printed on the line are counted (lines 3,4,5).

Task 5:

Assignment:
Create a program that draws a square with a side length of 10 characters on the screen.

Advice:
Use the PUTCH, PRINT, LET, GOTO, and IF-THEN statements. To save yourself some work, use a subroutine with GOSUB and RETURN statements to draw the horizontal line. Remember to end the program at the right point with the END statement.
The character to clear the display is 12, the new line is 10, the upper left corner is 144, the upper right corner is 145, the lower left corner is 146, the lower right corner is 147, the vertical line is 156, and the horizontal line is 157.

Solution example:

```
0 REM Square
1 PUTCH 12
2 LET A=1; REM Count lines
3 PUTCH 144; GOSUB 10
4 PUTCH 145; PUTCH 10
5 PUTCH 156; PRINT “        “
6 PUTCH 156; PUTCH 10; LET A=A+1
7 IF A<8 THEN GOTO 5
8 PUTCH 146; GOSUB 10
9 PUTCH 147; PUTCH 10; END
10 REM Horizontal line
11 LET I=1
12 PUTCH 157; LET I=I+1
13 IF I<9 THEN GOTO 12
14 RETURN
```

Solution notes:
Lines 3 and 4 draw the top of the square, lines 8 and 9 draw the bottom of the square. The subroutine on lines 10 through 14 is used to draw a horizontal line. Lines 5, 6, and 7 draw vertical lines. The number of spaces in the PRINT statement on line 5 must be 8 to empty the body of the square.

Task 6:

Assignment:
Write a program that allows you to move the character **X** around the screen. Starting in the middle of the screen, pressing the key **2** moves the character **X** down, **4** moves it left, **6** moves it right, and **8** moves it up.

Advice:
Use the PUTCH command using its parameters, CURSOR, INKEY, GOTO, and IF-THEN.
The character to clear the display has a value of 12, the **X** character is 88, the blank character is 32, **2** is 50, **4** is 52, **6** is 54, and **8** is 56.

Solution example:

```
0 REM Move X
1 PUTCH 12
2 CURSOR 400; PUTCH .88
3 INKEY K
4 IF K=50 THEN PUTCH v32; PUTCH .88
5 IF K=52 THEN PUTCH <32; PUTCH .88
6 IF K=54 THEN PUTCH >32; PUTCH .88
7 IF K=56 THEN PUTCH ^32; PUTCH .88
8 GOTO 3
```

Solution notes:
The program first clears the screen and places an X in the middle of the display (position 400) using the PUTCH command. The dot symbol ensures that the cursor remains at that position. When the user presses the correct key, the old position is first cleared with a blank character and the cursor moves to the new position (the direction is determined by the symbol before the value 32). The X is then printed at the new position and the cursor remains.

Task 7:

Assignment:
Create a program that will automatically "play ping-pong". A ball in the shape of the letter **o** will fly along a line from the left side to the right side and vice versa. It will always bounce off the edge of the screen and return.

Advice:
Use the PUTCH command using the parameters, CURSOR, DELAY, GOTO and IF-THEN.
The character to clear the display has a value of 12, the character **o** is 111, the blank character is 32.
The position in the middle of the middle line has a value of 400, the character on the left edge 384, the right 415.

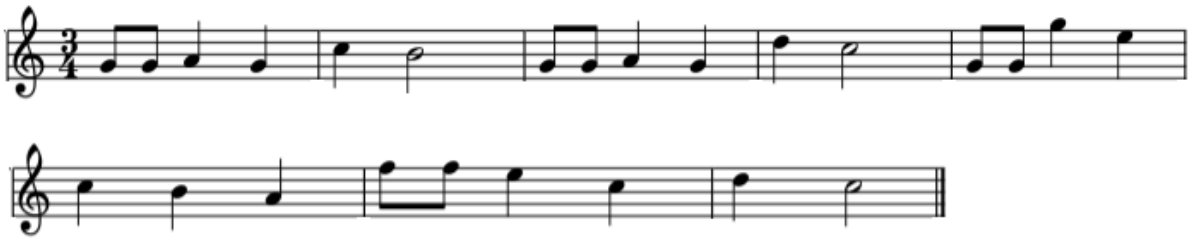
Solution example:

```
0 REM Ping-pong
1 PUTCH 12; LET C=400
2 CURSOR C; PUTCH .111
3 LET D=1
4 DELAY 100; PUTCH .32
5 LET C=C+D; CURSOR C; PUTCH .111
6 IF C=415 THEN LET D=-1
7 IF C=384 THEN LET D=1
8 GOTO 4
```

Solution notes:
First, the program is set to the initial position and the direction of the ball's movement in the variable D is set to the value 1 - i.e. the ball will fly to the right (lines 1 to 3). In the loop that starts on line 4, it waits for a while, deletes the old image of the ball and then moves the ball to a new position according to the value in D. Lines 6 and 7 monitor the edge of the screen and change the direction of flight there (D=1) or back (D=-1).

Task 8:

Assignment:
Write a program that plays a melody according to the notes in the image. At the beginning, the user enters the duration of a 1/8 note in the range 200 to 400 milliseconds.



Approximate frequencies of fundamental tones:

c‘: 262	d‘: 294	e‘: 330	f‘: 349	g‘: 392	a‘: 440	h‘: 494
c“: 523	d“: 587	e“: 659	f“: 698	g“: 784	a“: 880	h“: 989

Advice:
Use the BEEP command with the parameters, INPUT, GOTO and IF-THEN. If the same passages appear in the melody, you can use the GOSUB and RETURN subroutines to avoid having to copy the same parts. Don't forget to end the program with the END command.

Solution example:

```
0 REM Melody
1 PRINT “Enter the pace:\n“
2 INPUT T
3 IF T<200 THEN GOTO 1
4 IF T>400 THEN GOTO 1
5 GOSUB 15
6 BEEP 523,2*T; BEEP 494,4*T; GOSUB 15
7 BEEP 587,2*T; BEEP 523,4*T; BEEP 392,T
8 BEEP 392,T; BEEP 784,2*T; BEEP 659,2*T
9 BEEP 523,2*T; BEEP 494,2*T; BEEP 440,2*T
10 BEEP 698,T; BEEP 698,T; BEEP 659,2*T
11 BEEP 523,2*T; BEEP 587,2*T; BEEP 523,4*T
12 END
15 BEEP 392,T; BEEP 392,T; BEEP 440,2*T
16 BEEP 392,2*T; RETURN
```

Solution notes:
The user is first prompted to enter a tempo and the allowed range of values is checked (lines 1 to 4). Since the first four notes repeat in the third measure, the subroutine on lines 15 and 16 is used. The individual notes are specified by frequency and duration. A quarter note is double and a half note is quadruple the specified duration of an eighth note.

Task 9:

Assignment:
Write a program to create a very simple piano. When keys **1** to **9** are pressed, it will play the notes c' to d" for 200 to 600 milliseconds (the time is specified after the start).
Approximate frequencies of the fundamental notes:

c‘: 262	d‘: 294	e‘: 330	f‘: 349	g‘: 392	a‘: 440	h‘: 494
c“: 523	d“: 587					

Advice:
Use the BEEP command with the parameters, INPUT, INKEY, GOTO and IF-THEN. Try to take advantage of the fact that keys **1** to **9** return values 49 to 57 in the INKEY command. Then you can calculate the necessary jump in the GOTO command.

Solution example:

```
0 REM Piano
1 PRINT “Enter tone duration:\n“
2 INPUT T
3 IF T<200 THEN GOTO 1
4 IF T>600 THEN GOTO 1
5 INKEY K
6 IF K<49 THEN GOTO 5
7 IF K>57 THEN GOTO 5
8 GOTO K-40
9 BEEP 262,T; GOTO 5
10 BEEP 294,T; GOTO 5
11 BEEP 330,T; GOTO 5
12 BEEP 349,T; GOTO 5
13 BEEP 392,T; GOTO 5
14 BEEP 440,T; GOTO 5
15 BEEP 494,T; GOTO 5
16 BEEP 523,T; GOTO 5
17 BEEP 587,T; GOTO 5
```

Solution notes:
The user is first prompted to enter the tone duration and the allowed range of values is checked (lines 1 to 4). In the loop that starts on line 5, it is first checked whether the key is among the allowed ones and then the jump to the line with the corresponding tone is calculated on line 8. The value of the first allowed digit 49 must go to line 9, so the value 40 is subtracted.

Task 10:

Assignment:
Create a program that will represent a lottery device for the numbers 1 to 49. Each time you press any key, it will display a random number from the specified range.

Advice:
Use the TIME, INKEY, PRINT, GOTO and IF-THEN commands.
After pressing a key, take the current time. Its remainder after dividing by 49 will be a sufficiently random number. Remember that the remainder after dividing by 49 is in the range 0 to 48. The obtained value needs to be increased by 1.

Solution example:

```
0 REM Lottery
1 INKEY K
2 IF K=0 THEN GOTO 1
3 TIME T; LET A=T/49
4 PRINT T-A*49+1, "\n"
5 GOTO 1
```

Solution notes:
Lines 1 and 2 wait for any key to be pressed. When the user presses a key, the variable T is filled with the TIME command. The value obtained by the TIME command changes approximately 32,000 times per second. Since we need random numbers up to 49, we calculate the remainder after dividing by 49. We display the result increased by 1.

Task 11:

Assignment:
Create a program that displays the time elapsed since the computer was started in the form HH:MM:SS.

Advice:
Use the TIME, PRINT, PUTCH, CURSOR, GOTO and IF-THEN commands.
If you divide the value obtained by the TIME command by 31914, you will get the time since the computer was started in seconds. Then you just need to divide the obtained time by 3600 to get the number of hours that have passed. Then divide the remainder by 60 to get the minutes. Do not forget that in Assignment the number is displayed to 2 places. It is not enough to just print out e.g. 5, it needs to be displayed as 05. A subroutine with GOSUB and RETURN can be useful for this. The character **:** has the value 58, the character **0** has the value 48.

Solution example:

```
0 REM Clock
1 PUTCH 12
2 DELAY 300
3 TIME T; LET S=T/31914
4 LET A=S/3600; LET S=S-A*3600
5 CURSOR 0; GOSUB 10
6 LET A=S/60; LET S=S-A*60
7 PUTCH 58; GOSUB 10
8 LET A=S; PUTCH 58; GOSUB 10
9 GOTO 2
10 REM List A to 2 digits
11 IF A<10 THEN PUTCH 48
12 PRINT A; RETURN
```

Solution notes:
We divide the time obtained by the TIME command by 31914 to get the total time in seconds. We get the hours in variable A by dividing by 3600. The remainder after integer division will be minutes and seconds. By integer division by 60 we get minutes and the remainder from the calculation of minutes is seconds. We have to display everything to 2 places, so we add any leading zero for single-digit numbers when displaying the value from A in the subroutine (line 11).

Task 12:

Assignment:
Create a “kitchen timer” using a computer. After starting the program, the user enters a time from 1 to 99 minutes. After entering, the display will start counting down in minutes and when it reaches 0, a signal will sound for 10 seconds.

Advice:
Use the TIME, BEEP, INPUT, PRINT, GOTO and IF-THEN commands. After entering the desired time, you can calculate the time at which the timing will end. The target time is obtained by adding the number of minutes multiplied by 60*31914 (seconds in a minute times the number of ticks per second = 1914840) to the current time.

Solution example:

```
0 REM Kitchen timer
1 PRINT “Enter time [min]:\n“
2 INPUT T
3 IF T<1 THEN GOTO 1
4 IF T>99 THEN GOTO 1
5 TIME C; LET C=C+T*60*31914
6 PUTCH 12
7 DELAY 300
8 TIME T; LET D=C-T
9 IF D<0 THEN BEEP 800,10000; END
10 CURSOR 0; PRINT D/1914840, “ “
11 GOTO 7
```

Solution notes:
The user is first prompted to enter a time and the allowed range of values is checked (lines 1 to 4). After correct entry, the target time is stored in the variable C, which is calculated as the current time with the time entered by the user multiplied by 31914 (number of ticks per second) and 60 (number of seconds in a minute). In the loop that starts on line 7, we wait a while because it is not necessary to display the values so quickly that the user would not notice it anyway. On line 8, we find the difference between the target and current time. If the result is negative, we end the timing. Otherwise, we display the difference. The constant 1914840 is created by multiplying 60*31914.

Task 13:

Assignment:
Program the game "Better Perception" for two or more players. After pressing a key, it waits 5 seconds, while the timing starts *****. Then it displays the target character on the display. The key that was pressed first after the target character was displayed will be declared the winner.

Advice:
Use INKEY, DELAY, PUTCH, PRINT, GOTO and IF-THEN statements. When writing your program, remember to check if the player pressed a key before displaying the target character. The ***** character has the value 42, the target character has the value 143.

Solution example:

```
0 REM Perception
1 PRINT “\nGame start\n“
2 INKEY K
3 IF K=0 THEN GOTO 2
4 PUTCH 42; DELAY 5000; INKEY K
5 PUTCH 143; IF K>0 THEN GOTO 10
6 INKEY K
7 IF K=0 THEN GOTO 6
8 PRINT “\nWinner: “; PUTCH K; DELAY 2000
9 GOTO 1
10 PRINT “\nLoser: “; PUTCH K; DELAY 2000
11 GOTO 1
```

Solution notes:
On lines 1 to 3 we wait for the game to start. Then we display *, wait 5 seconds, and check if someone pressed a key prematurely (line 4). On line 5 we display the target character and check for premature keypresses. If everything is OK, on lines 6 and 7 we wait for a keypress, and on line 8 we declare the winner.

Task 14:

Assignment:
Create a program for an acoustic generator. The user enters the frequency of the desired tone in the range of 100 to 4000 Hz and the computer beeps the entered tone until the program is terminated.

Advice:
Use the BEEP, INPUT, PRINT, GOTO, and IF-THEN statements.
To be able to stop the program by inserting an END statement, use some acceptable value in the time parameter of the BEEP statement in the range of seconds.

Solution example:

```
0 REM Audio generator
1 PRINT “Insert freq. 100-4000\n“
2 INPUT F
3 IF F<100 THEN GOTO 1
4 IF F>4000 THEN GOTO 1
5 BEEP F,5000
6 GOTO 5
```

Solution notes:
The user is first prompted to enter a frequency and the allowed range of values is checked (lines 1 to 4). Then, a tone of the specified frequency is played continuously in a loop on lines 5 and 6.

Task 15:

Assignment:
Write a program in which the user enters a *name* followed by the Enter key. The computer then prints out that the *name* is his friend. The maximum length of the name can be 30 characters.

Advice:
Use the INKEY, PRINT, PUTCH, GOTO and IF-THEN commands. It is convenient to store the entered *name* in variables in the @ array. Count the number of letters stored.

Solution example:

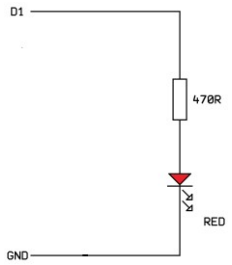
```
0 REM Computer friend
1 LET I=0; PRINT “\nEnter name:\n“
2 INKEY K
3 IF K=0 THEN GOTO 2
4 IF K=10 THEN GOTO 10
5 PUTCH K; LET @I=K; LET I=I+1
6 IF I=30 THEN GOTO 10
7 GOTO 2
10 LET J=0; PUTCH 10
11 PUTCH @J; LET J=J+1
12 IF J<I THEN GOTO 11
13 PRINT “ is my friend\n“
```

Solution notes:
Lines 2 and 3 wait for a character to be entered. Line 4 checks whether the input was terminated by pressing the Enter key. Line 5 stores the character in the array. Line 6 checks the maximum allowed length of the name. Lines 10 to 12 print the name again.

Task 16:

Assignment:
Create a program to control an LED connected to output D1 using the keyboard. The first press of any key turns the LED on, the second press turns it off, the next press turns it on again, and so on.

Additional HW:
Connect an LED with a resistor to the D1 output as shown in the following figure. The red LED will be suitable for the following task.



Advice:
Use the INKEY, DOUT, GOTO, and IF-THEN statements. The DOUT 1,1 statement turns the LED on, and the DOUT 0,1 statement turns it off.
Since the program doesn't have to be fast, you can insert a small delay into the keypress cycle with the DELAY statement.

Solution example:

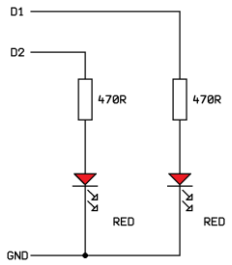
```
0 REM LED control
1 LET L=0
2 DELAY 100; INKEY K
3 IF K=0 THEN GOTO 2
4 IF L=0 THEN GOTO 6
5 LET L=0; DOUT 0,1; GOTO 2
6 LET L=1; DOUT 1,1; GOTO 2
```

Solution notes:
The variable L stores the current value of the output so that we know how to change the output next time. Lines 2 and 3 wait for a key press. Line 4 decides whether the LED turns on or off based on the state of L.

Task 17:

Assignment:
Create a program that turns TinyBasRV into a control for warning lights at a railroad crossing, including sound signaling.

Additional HW:
Connect two LEDs (preferably red) with resistors to outputs D1 and D2 as shown in the following figure.



Advice:
Use the BEEP, DELAY, DOUT and GOTO commands. Try out the sound signaling and adjust it to your liking.

Solution example:
```
0 REM Railroad crossing
1 DOUT 1,1; DOUT 0,2
2 BEEP 2000,100; BEEP 800,200; DELAY 200
3 DOUT 0,1; DOUT 1,2
4 BEEP 2000,100; BEEP 800,200; DELAY 200
5 GOTO 1
```

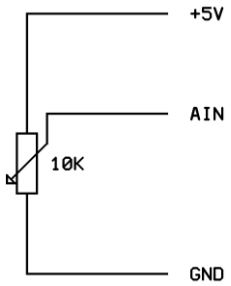
Solution notes:
Lines 1 and 3 always light up only one of the two LEDs. Lines 2 and 4 generate the same sound signal. The example given only very vaguely resembles the ringing of a bell. You can tune and adjust them to your ear...

Rev. C

Task 18:

Assignment:
Create a program to measure the input voltage at the AIN input in mV. Apply the voltage from the potentiometer slider to the input, which will be displayed with a period of ½ second.

Additional HW:
Connect a linear potentiometer with a resistance value in the range of 1 to 100 kΩ to the AIN output according to the following figure.



Advice:
Use the AINP, DELAY, CURSOR, PRINT, and GOTO commands. The value obtained from the A/D converter is in the range 0 to 1023. This represents a voltage of 0 to 5V – that is, 0 to 5000 mV. To convert the A/D converter range to mV, we must first multiply the value by 5000 and then divide by 1023.

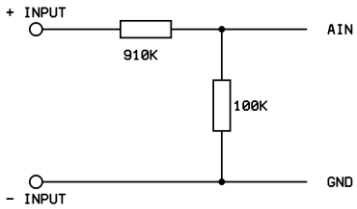
Solution example:
```
0 REM Analog input mV
1 PUTCH 12
2 CURSOR 397; AINP A
3 PRINT A*5000/1023, " mV   "
4 DELAY 500
5 GOTO 2
```

Solution notes:
On line 2, first set the cursor approximately to the center of the screen and then measure the voltage at the AIN input. The conversion is performed when the value is printed in the PRINT command according to the instructions in the **Advice** paragraph. After the units are printed, 3 spaces are added to erase any longer entries from the previous measurement.

Task 19:

Assignment:
Modify the TinyBasRV computer to work as a simple voltmeter with a range of up to 50 V. The voltage will be displayed with a period of ½ second and a resolution of 2 decimal places.

Additional HW:
Connect an input resistor divider to the AIN output with resistors with a resistance ratio of 9:1. For example, 900 kΩ and 100 kΩ would be ideal. However, the value of 900 kΩ is outside the normal range of available resistors. You can use the closest value of 910 kΩ from the E24 series. In an emergency, you can also use 1 MΩ and adjust the constant in the program.



Advice:
Use the AINP, DELAY, CURSOR, PUTCH, PRINT, GOTO, and IF-THEN commands. The value obtained by AINP from the A/D converter is in the range of 0 to 1023. This represents a voltage of 0 to 50V (we need to display 0.00 to 50.00 V). To convert the A/D converter range to V, we first need to multiply the value by 5000 and then divide by 1023. We display the resulting number with a decimal point between the digits.
The multiplication constant of 5000 is the ideal case, when an accurate reference voltage in the microcontroller and accurate resistance values of the two resistors in the divider are assumed. In practice, this will vary. Therefore, for a particular piece of computer, this value needs to be adjusted so that the displayed voltage is as close as possible to the voltage measured by the reference voltmeter.

Solution example:

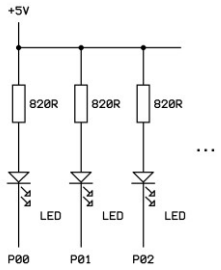
```
0 REM Voltmeter 50V
1 PUTCH 12; LET K=5320
2 CURSOR 397; AINP A
3 LET V=A*K/1023; LET C=V/100
4 LET Z=V-C*100
5 PRINT C, “.“
6 IF Z<10 THEN PRINT “0“
7 PRINT Z
8 DELAY 500
9 GOTO 2
```

Solution notes:
Line 1 sets the conversion constant, which you must adjust according to the specific computer and input resistor divider. The next step copies the instructions given in the **Advice** paragraph. First, the entire part of the variable C is printed, and then the remainder after dividing 100 from Z. If the remainder is less than 10, it is necessary to add a leading zero (line 6).

Task 20:

Assignment:
Using a PCF8575 expander connected to a TinyBasRV, make a running light on a series of LEDs connected to the expander outputs.

Additional HW:
Connect multiple LEDs to the outputs of the PSC8575 expander module as shown in the picture. It depends on the number of LEDs you have available, a maximum of 16 LEDs can be connected.



Advice:
Use the I2CW, DELAY, LET, GOTO and IF-THEN commands. To light the first LED, you need to send the binary number b1111111111111110 or decimal -2 to the expander. The second LED requires sending the binary number b1111111111111101 or decimal -3. The next would be the binary number b1111111111111011 or decimal -5. The LED lighting is moved to the next position by multiplying by 2 and adding 1.
After the last LED is lit, the light must return to the beginning.

Solution example (for 8 LEDs):

```
0 REM Running 8 LED
1 LET A=-2
2 I2CW A; DELAY 200
3 LET A=A*2+1
4 IF A < -300 THEN LET A=-2
5 GOTO 2
```

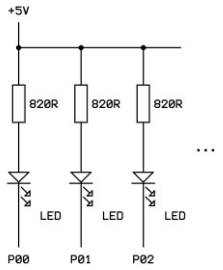
Solution notes:
The program follows the instructions given in the **Advice** paragraph. In the given example, only 8 LEDs were used, when reaching a value less than -300 the value in A returns to the beginning. Sending the number -257 to the expander (the last one sent in the cycle) will turn off all 8 LEDs for a while. If you do not want this, reduce the condition, for example, to A<-150.

Rev. C

Task 21:

Assignment:
Write a program that uses a PCF8575 expander connected to TinyBasRV to light up a specified number of LEDs (an image of your choice) when you press the **0** to **9** key.

Additional HW:
Connect multiple LEDs to the outputs of the PSC8575 expander module as shown in the picture. It depends on the number of LEDs you have available, a maximum of 16 LEDs can be connected.



Advice:
Use the I2CW, DELAY, INKEY, GOTO and IF-THEN commands. You can specify any pattern of LED lights for each digit. For example, pressing key **1** will light up each odd LED with the I2CW command b1010101010101010, key **2** will light up the bottom half of the LEDs with the I2CW command b111111100000000, etc.
Keys **0** to **9** return values 48 to 57 in the INKEY command.

Solution example:

```
0 REM LED patterns
1 DELAY 100; INKEY K
2 IF K<48 THEN GOTO 1
3 IF K>57 THEN GOTO 1
4 GOTO K-43
5 I2CW b1100110011001100; GOTO 1
6 I2CW b1010101010101010; GOTO 1
7 I2CW b1111111100000000; GOTO 1
8 I2CW b1111100001100101; GOTO 1
9 I2CW b0000000011001100; GOTO 1
10 I2CW b1100110000000000; GOTO 1
11 I2CW b1110111011101110; GOTO 1
12 I2CW b1101110111011101; GOTO 1
13 I2CW b1011101110111011; GOTO 1
14 I2CW b0111011101110111; GOTO 1
```

Solution notes:
First, lines 1, 2, and 3 ensure that only the allowed number keys are accepted. Line 4 then ensures a jump to the appropriate pattern of lit LEDs and a return to the beginning.

Task 22:

Assignment:
Program a user's perception meter. After pressing a key, the program waits a random number of seconds (1 to 10), with the timing starting at *. Then it displays the target character on the display. The user must press any key as quickly as possible. The display shows the reaction time in milliseconds.

Advice:
Use the INKEY, TIME, DELAY, PUTCH, PRINT, GOTO, and IF-THEN commands. When writing your program, remember to check to see if the player pressed a key before the target character is displayed. To measure time in milliseconds, we will assume that TIME returns a value 32 larger for each millisecond.
The ***** character has the value 42, the target character has the value 143.

Solution example:

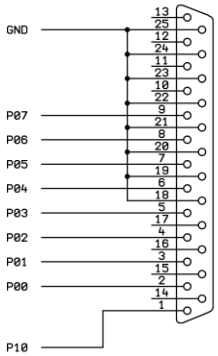
```
0 REM Perception meter
1 PRINT “\nStart\n“
2 INKEY K
3 IF K=0 THEN GOTO 2
4 TIME T; LET C=T/10
5 LET D=T-C*10+1; LET D=D*1000
6 PUTCH 42; DELAY D
7 INKEY K; PUTCH 143; TIME S
8 IF K>0 THEN GOTO 14
9 INKEY K
10 IF K=0 THEN GOTO 9
11 TIME E; LET R=E-S
12 PRINT “\nReaction: “, R/32
13 PRINT “ milisec“; GOTO 1
14 PRINT “\nPrematurely!“
15 GOTO 1
```

Solution notes:
Lines 2 and 3 wait for the measurement to start. Then, using the TIME command and the remainder after dividing by 10, a random time is obtained, which is stored in the variable D in milliseconds (lines 4 and 5). Line 6 displays * and waits. Line 7 reads the key state, displays the target character, and stores the starting time for the perception measurement. Line 8 checks for any premature key presses. Lines 9 and 10 wait for the user to respond. The time at which the user responded is stored in E, and then lines 11, 12, and 13 print the result.

Task 23:

Assignment:
If you have the choice to connect a dot matrix printer with a parallel port to the TinyBasRV via a PCF8575 expander, create a simple typewriter from it.

Additional HW:
Connect the Cannon 25 female connector to the outputs of the PCF8575 expander module as shown in the picture.



Advice:
Use the I2CW, DELAY, INKEY, GOTO and BEEP commands. The BEEP command is used to sound a short beep when a key is pressed. It is not necessary to use it.
On pin 1 of the parallel port there is a STROBE signal controlled by output P10, which confirms and writes data to the printer. The active value is logical 0. After writing the data, it is necessary to return this signal to the idle value 1. This is easiest to do by writing a value 256 greater than the data being written.

Solution example:
```
0 REM Typewriter
1 INKEY K
2 IF K=0 THEN GOTO 1
3 I2CW K; PUTCH K
4 DELAY 10
5 I2CW K+256; BEEP 1000,50
6 GOTO 1
```

Solution notes:
Lines 1 and 2 wait for a keystroke. Line 3 sends the character to the printer and copies it to the display. Line 4 waits for a moment (in case the printer is slow to respond). Line 5 must set the STROBE signal for the printer back to logic 1 (the character value is increased by the 9th bit set to 1) and a short beep is emitted.

Task 24:

Assignment:
Create a simple integer calculator from TinyBasRV. The user enters the first number, then the operand (+ - * /) and finally the second number. The calculated result appears on the display in the form, for example, 256*2=512.

Advice:
Use the INPUT, INKEY, PRINT, PUTCH, GOTO, and IF-THEN commands. The operand characters have the following values:
+ 43 - 45 * 42 / 47
Don't forget that we don't divide by zero!

Solution example:

```
0 REM Calculator
1 PUTCH 10; INPUT A
2 INKEY K
3 IF K=42 THEN GOTO 8
4 IF K=43 THEN GOTO 8
5 IF K=45 THEN GOTO 8
6 IF K=47 THEN GOTO 8
7 GOTO 2
8 PUTCH K; PUTCH 10; INPUT B;
9 PRINT A; PUTCH K; PRINT B
10 IF K=43 THEN PRINT "=",A+B; GOTO 1
11 IF K=45 THEN PRINT "=",A-B; GOTO 1
12 IF K=42 THEN PRINT "=",A*B; GOTO 1
13 IF B=0 THEN PRINT "0 not possible!"; GOTO 1
14 PRINT "=",A/B; GOTO 1
```

Solution notes:
The first value is stored in variable A on line 1. Then lines 3 through 7 wait for the operation type. Line 8 prints the operation character and waits for the second value. Lines 10 through 14 perform the calculation based on the state of K, with line 13 checking for division by 0.

Task 25:

Assignment:
Write a program to guess a secret number from 0 to 99. After starting, the computer chooses a random number and the player has to guess this number in the fewest number of attempts possible. After each attempt, the computer indicates whether the secret number is greater or less than the one entered by the player.

Advice:
Use the INPUT, PRINT, TIME, LET, GOTO, and IF-THEN statements. First, generate a random number using the TIME statement and the remainder after dividing by 100.

Solution example:

```
0 REM Guess number
1 TIME T; LET C=T/100; LET X=T-C*100
2 LET P=0; PRINT "\nGuess number\n"
3 INPUT A; LET P=P+1
4 IF A=X THEN GOTO 8
5 IF A>X THEN GOTO 7
6 PRINT P,". Guess more\n"; GOTO 3
7 PRINT P,". Guess less\n"; GOTO 3
8 PRINT "You guessed in ", P, " tries\n"
```

Solution notes:
On line 1, a randomly generated guessed number is stored in variable X using the TIME command and the remainder after integer division by 100. On lines 2 and 3, the user is asked to try to guess the number, and the number of attempts is counted in variable P. Lines 4 and 5 check the success of the attempt, and the remaining lines print instructions on where to go for the next guess or exit the program with the result.

Task 26:

Assignment:
Create a simple code lock from TinyBasRV. If the user enters the correct sequence of 4 or 5 digits on the keyboard, a logical 1 will appear on the D1 output for 3 seconds and a sound signal will sound.

Advice:
Use the INKEY, LET, DOUT, BEEP, GOTO and IF-THEN commands. You can choose the number of code digits yourself. Store the secret number combination in the @ array and each correctly guessed digit will move the pointer in the array to a higher value. If you make a mistake, the pointer will return to the beginning of the array.
The keys **0** to **9** return the values 48 to 57 in the INKEY command.

Solution example:

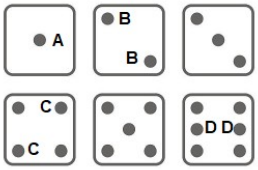
```
0 REM Code lock
1 LET @0=49,57,50,48,51
2 LET P=0
3 INKEY K
4 IF K=0 THEN GOTO 3
5 IF K<>@P THEN GOTO 2
6 LET P=P+1;
7 IF P<5 THEN GOTO 3
8 DOUT 1,1; BEEP 800,3000; DOUT 0,1
9 GOTO 2
```

Solution notes:
On line 1, the lock code is stored in the @ array. The variable P shows where the user got to when entering the code. Line 5 decides whether to advance to the next position or return to the first digit of the code. On line 7, if the procedure is correct and the P pointer reaches the value 5, it is possible to move to line 8, where the code lock opens for 3 seconds.

Task 27:

Assignment:
Write a program that will replace a dice. The program will display the points on the screen as they are located on the dice.

Advice:
If you look at the patterns on the sides of the cube, you will notice that, except for the center dot, the dots always appear in pairs. For example, 3 is a combination of A+B, 5 is A+B+C.



You can encode patterns into an array, for example using binary numbers, and then display them according to a randomly drawn number. Recommended cursor positions are for A: 400, for B: 235 and 565, for C: 245 and 555, and for D: 395 and 405. Use the LET, TIME, INKEY, GOTO, CURSOR, PUTCH, and IF-THEN commands.

Solution example:

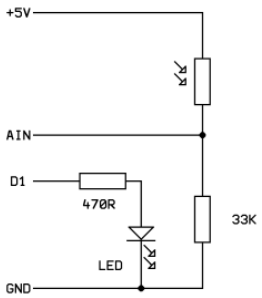
```
0 REM Dice
1 PUTCH 12; LET @0=1,2,3,6,7,14
2 INKEY K
3 IF K=0 THEN GOTO 2
4 TIME D; LET C=D/6; LET D=D-C*6
5 CURSOR 400; PUTCH .32; IF @D&1 THEN PUTCH 143
6 CURSOR 235; PUTCH 32; CURSOR 565
7 PUTCH .32; IF@D&2 THEN PUTCH 143
8 CURSOR 235; IF@D&2 THEN PUTCH 143
9 CURSOR 245; PUTCH 32; CURSOR 555
10 PUTCH .32; IF@D&4 THEN PUTCH 143
11 CURSOR 245; IF@D&4 THEN PUTCH 143
12 CURSOR 395; PUTCH 32; CURSOR 405
13 PUTCH .32; IF@D&8 THEN PUTCH 143
14 CURSOR 395; IF@D&8 THEN PUTCH 143
15 GOTO 2
```

Solution notes:
The @ array encodes the display of the numbers 0 to 5 using binary code. If you look at how the numbers are written in binary, you will see that the lowest bit represents dot A, the next higher bit represents both dots B, the next C, and the fourth bit is dot D. Lines 2 and 3 wait for a key to be pressed. Line 4 generates a random number 0 to 5 in variable D. Then all positions of the individual dots are traversed. For example, in line 5, the cursor is set to dot A and it is deleted first. If the array element with index D has the lowest bit set, the dot at the position is drawn.

Task 28:

Assignment:
Complete the TinyBasRV to function as an emergency lighting control panel. The photoresistor will be used to sense the ambient light level and the measured value will be displayed on the display. The user will initially enter the values at which the LED connected to D1 will turn on and off.

Additional HW:
Connect a divider with a photoresistor and an LED with a resistor to the AIN input and D1 output as shown in the figure.



Advice:
Use the INPUT, PRINT, DOUT, AINP, GOTO and IF-THEN commands. The measured light value will be just a number from the A/D converter. The user can set 2 limits at which the light turns on or off, thus setting the hysteresis of the circuit. Make sure that the entered limits are not outside a reasonable range.
You can use a type 5537 or similar as a photoresistor.

Solution example:

```
0 REM Emergency lighting
1 PRINT "\f0\n\n"
2 INPUT H
3 IF H<10 THEN GOTO 2
4 IF H>1000 THEN GOTO 2
5 INPUT L
6 IF L>=H THEN GOTO 5
7 AINP A; CURSOR 0; PRINT A,"   "
8 IF A<L THEN DOUT 1,1
9 IF A>H THEN DOUT 0,1
10 DELAY 200; GOTO 7
```

Solution notes:
Lines 2 to 6 require the entry of the limits at which the LED turns off or on (including checking the allowed values). The actual loop is on lines 7 to 10. After the measurement, the value is printed. The PRINT command is supplemented by deleting a longer number that could remain on the display from the previous measurement. Then, on lines 8 and 9, the LED is turned on or off.

Task 29:

Assignment:
Create a simple cryptographic device from TinyBasRV that will encode and decode characters from the alphabet to another character according to your chosen encryption table. The program must display both the encoded and the corresponding decoded character after pressing character on keyboard.

Advice:
Use the LET command to create an encryption table in the @ array. After checking whether the pressed key is among the allowed characters, select a character from the table, display it, and then search the table for decoding. Recommended commands are PRINT, INKEY, CURSOR, GOTO, and IF-THEN.

Solution example:

```
0 REM Cryptograph
1 LET @0=6,18,11,5,20,14,22,2,9,15
2 LET @10=17,8,1,10,19,0,13,4,23,24
3 LET @20=25,7,16,12,21,3
4 PRINT “\fChar   Coder   Decoder“
5 INKEY K
6 IF K<97 THEN GOTO 5
7 IF K>122 THEN GOTO 5
8 LET Z=K-97; CURSOR 34; PUTCH K
9 CURSOR 41; PUTCH 65+@Z; LET I=0
10 IF @I=Z THEN GOTO 12
11 LET I=I+1; IF I<26 THEN GOTO 10
12 CURSOR 50; PUTCH 65+I; GOTO 5
```

Solution notes:
The basis of the program is the encoding table in the @ array. All 26 characters of the English alphabet have their randomly chosen code. The character "a" has a code at position 0, "b" at 1, etc. The code numbers can be from 0 to 25.
We consider entering lowercase letters of the alphabet, their values will be 97 to 122. Lines 5 to 7 are waiting for this. We obtain a pointer to the encoding table in the variable Z by subtracting the value 97 from the character. We print the entered character, its encoded form in the form of an uppercase letter, which begin with the values 65 (lines 8 and 9). On lines 10 and 11 we search the table for the opposite function - decoding. Line 12 prints the decoded character.

Task 30:

Assignment:
Write a program for a simple stopwatch. The first press of any key will start the measured time, the next press will stop the value. It will be displayed in minutes:seconds,tenths of seconds.

Advice:
Use the INKEY, LET, PUTCH, PRINT, GOTO and IF-THEN commands. After starting the program, first wait for a key press. Save the current start time. Display the difference between the current time and the start time by dividing 3191 (the number of ticks per tenth of a second). First, the value divided by 600 – these will be minutes, then seconds and tenths of seconds. Remember to always display seconds to 2 decimal places. The character **0** has the value 48.
Another press stops the counting so that the time reached can be read. A third press of the key then returns the stopwatch to zero time and prepares it for starting.

Solution example:

```
0 REM Stopwatch
1 PUTCH 12; PRINT “0:00,0“
2 INKEY K
3 IF K=0 THEN GOTO 2
4 TIME B
5 DELAY 80
6 TIME T; LET T=T-B; LET S=T/3191
7 LET A=S/600; LET S=S-A*600
8 CURSOR 0; PRINT A,“:“
9 LET A=S/10; LET S=S-A*10
10 IF A<10 THEN PUTCH 48
11 PRINT A,“,“,S
12 INKEY K; IF K=0 THEN GOTO 5
13 INKEY K
14 IF K=0 THEN GOTO 13
15 GOTO 1
```

Solution notes:
The solution follows the instructions in the **Advice** paragraph. First, the stopwatch is waited for to start (lines 2 and 3). The start time is stored in variable B. The human eye is not very fast, so the display loop has a delay on line 5. The elapsed time in tenths of a second is calculated in variable S on line 6. Lines 7 to 11 display the time on the display. Line 12 checks to see if the stopwatch has stopped. Lines 13 and 14 wait for the program to restart.

Task 31:

Assignment:
Write a program that represents a simple game called "Horse Racing". The player can bet an amount on one of three horses with an initial capital. The horses will finish in a random order.

Advice:
Use, for example, the commands INPUT, PRINT, PUTCH, LET, CURSOR, DELAY, TIME, GOTO and IF-THEN. Store the position of the horse in the @ array. This will make it easier to refer to the horse, which will be moved randomly using the TIME command.
When betting on the winning horse, add three times the bet to the player.

Solution example:

```
0 REM Horse Racing
1 LET M=1000
2 PRINT “\fBet? 1-“,M,“\n“
3 INPUT B; IF B<1 THEN GOTO 2
4 IF B>M THEN GOTO 2
5 PRINT “\nHorse 1-3?\n“; INPUT K
6 LET @1=1; LET @2=1; LET @3=1
7 PRINT “\fRace“; PUTCH p49,64
8 PUTCH p50,128; PUTCH p51,192
9 TIME T; LET V=T/3; LET V=T-V*3+1
10 DELAY 100; LET @V=@V+1; PUTCH p143,V*64+@V
11 IF @V<31 THEN GOTO 9
12 CURSOR 395; PRINT “Win “,V; DELAY 1000
13 IF V=K THEN LET M=M+B*3; GOTO 2
14 LET M=M-B; IF M>0 THEN GOTO 2
15 PRINT “\nNo money\n“
```

Solution notes:
The initial capital is set on line 1. Then the user is asked to determine the value of the bet with a check to see if he is trying to cheat (lines 2 to 4). Line 5 asks to bet on one of the three horses. Here, the validity of the choice is not checked; if the player enters a meaningless number, the money is thrown out the window.
Line 6 places the horse in the starting position (the horse positions are in the corresponding element of the array @). Lines 7 and 8 prepare the race graphics. Line 9 generates which horse will randomly advance one space. This is plotted on line 10, with the horse's position being stored in its variable.
Line 11 handles the eventual victory of the horse and the end of the race. Lines 12 to 15 pay out the winnings, collect the lost money, and eventually end the game if there is a shortage of money.

Task 32:

Assignment:
Write a simple game for TinyBasRV called "Frog Jumpers". At the beginning, there will be 4 frogs of one type on the left and 4 frogs of another type on the right. There is one gap in the middle. The player can jump with any frog into the adjacent gap or jump over one of the neighboring frogs into the gap. The goal is to move the frogs from the left to the right side and vice versa in the fewest number of jumps.

Advice:
Use the INPUT, LET, PUTCH, PRINT, CURSOR, GOTO and IF-THEN commands. You can store the starting and target positions in the @ array. Choose the characters for the frogs on the left and right yourself. The player will always have to enter "from where" frog jumps "to where". This can be simply entered as a number, e.g. 34 - jump from position 3 to 4. You will have to take care that the player does not enter meaningless jump positions. They are also not allowed to jump over 2 or more frogs at once.

Solution example:

```
0 REM Frog Jumpers
1 LET P=0; LET @1=8,8,8,8,15,1,1,1,1
2 LET @10=1,1,1,1,15,8,8,8,8
3 PUTCH 12; LET I=1
4 CURSOR I*2; PUTCH v80+@I; PRINT I
5 LET I=I+1; IF I<10 THEN GOTO 4
6 PRINT “\n\n“; INPUT M; IF M>99 THEN GOTO 3
7 LET F=M/10; LET T=M-F*10; LET D=F-T
8 IF T<1 THEN GOTO 3
9 IF F<1 THEN GOTO 3
10 IF D<0 THEN LET D=-D
11 IF @T<>15 THEN GOTO 3
12 LET I=1; IF D>2 THEN GOTO 3
13 LET P=P+1; LET @T=@F; LET @F=15
14 LET K=I+9; IF @I<>@K THEN GOTO 3
15 LET I=I+1; IF I<10 THEN GOTO 14
16 PRINT “\nDone in “,P,“ jumps“
```

Solution notes:
The number of jumps is stored in the variable P. The @ first stores the frogs' starting positions and then the target position. The numbers are small to make them easy to write in the program. For display, the characters will be increased by 80, so that the left frogs will appear as Q (81), the right as X (88), and the free position will be an underscore (95). Lines 3 to 5 draw the playing field with the frogs and their positions.
On line 6, the player enters his jump. It is immediately checked whether the number is not too large. On line 7, it is divided into from (F) and to (T) the jump is made and how long the jump is (D). Lines 8 and 9 check whether a 0 or negative number has been entered somewhere. Line 10 corrects the jump length to a positive number.
Line 11 checks whether the jump is made to a free position, line 12 if the jump is not too long. A valid jump is made on line 13. Lines 14 and 15 check for a possible game end when the goal is reached.