

Quick Reference Guide for Pascal language

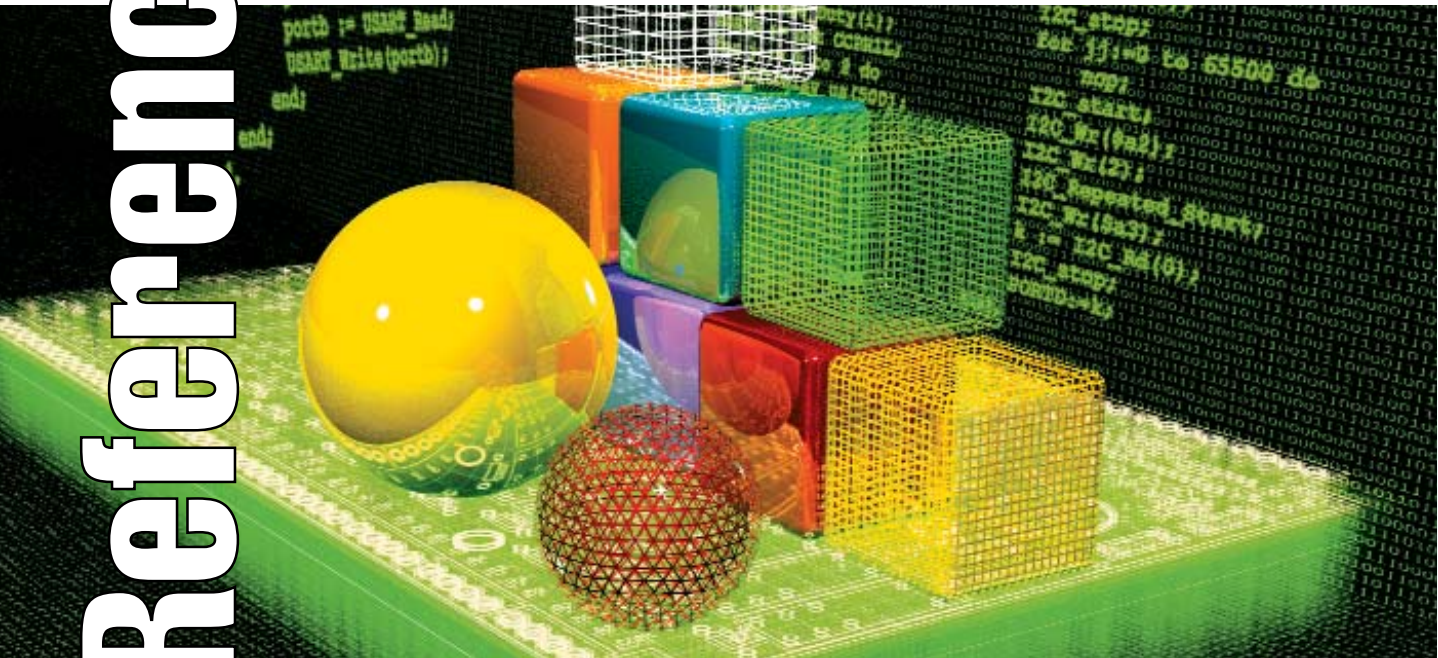
Quick Reference

with
EXAMPLES

This reference guide is intended to quickly introduce users to Pascal language syntax with the aim to easily start programming microcontrollers along with other applications.

Why Pascal in the first place? The answer is simple: it is legible, easy-to-learn, structured programming language, with sufficient power and flexibility needed for programming microcontrollers. Whether you had any previous programming experience, you will find that writing programs in Pascal is very easy.

Software and Hardware
solutions for Embedded World



mikroPascal Quick Reference Guide

COMMENTS

All text between a left brace and a right brace constitutes a comment. May span multiple lines.

Example:

```
{ Put your comment here! It may span multiple lines. }
```

Any text between a double-slash and the end of the line constitutes a comment. May span one line only.

Example:

```
// Put your comment here!  
// It may span one line only.
```

LITERALS

Character Literals

Character literal is one character from the extended ASCII character set, enclosed with apostrophes.

Example:

```
'A' // this is character A
```

String Literals

String literal is a sequence of up to 255 characters from the extended ASCII character set, enclosed with apostrophes.

Example:

```
'Hello!' // 6 chars long  
'C' // 1 char long
```

KEYWORDS

absolute	function	read
and	goto	record
array	if	repeat
asm	implementation	shl
begin	interrupt	shr
boolean	in	step
break	is	string
case	label	then
char	mod	to
continue	not	type
const	or	unit
div	org	until
do	otherwise	uses
downto	print	var
else	procedure	while
end	program	with
for	real	xor

Note:

User can not use keywords for variable or function names. Keywords are reserved only for making pascal language statements.



VARIABLES

Syntax:

```
var identifier_list : type;
```

Example:

```
var i, j, k : byte;
```

CONSTANTS

Syntax:

```
const constant_name [: type] = value;
```

Example:

```
const  
MAX : longint = 10000;  
MIN = 1000; // compiler will assume word type  
SWITCH = 'n'; // compiler will assume char type  
MSG = 'Hello'; // compiler will assume string type
```

mikroPascal Quick Reference Guide

LABELS

Syntax:

```
label_identifier: statement
```

Before marking a statement, you must first declare the label:

```
label label1, ..., labeln;
```

Example:

```
label loop;
...
loop: Beep; // infinite loop
goto loop; // that calls the
           // Beep procedure
```

FUNCTIONS

Syntax:

```
function function_name(parameter_list) : return_type;
{ local declarations }
begin
{ function body }
end;
```

Example:

```
function add(a, b : byte) : byte;
begin
result := a + b;
end;
```

We can call it to calculate sum of two numbers:

```
var c : byte;
c := add(4, 5);
```

Variable c will then be 9.

PROCEDURES

Syntax:

```
procedure procedure_name(parameter_list);
{ local declarations }
begin
{ procedure body }
end;
```

Example:

```
procedure add(var c, a, b : byte);
begin
c := a + b;
end;
```

We can call it to calculate sum of two numbers (last two parameters) and place result in first parameter:

```
var c : byte;
add(c, 4, 5);
```

Variable c will then be 9.

SIMPLE TYPES

Type	Size	Range
byte	8-bit	0 .. 255
char	8-bit	0 .. 255
word	16-bit	0 .. 65535
short	8-bit	- 128 .. 127
integer	16-bit	-32768 .. 32767
longint	32-bit	-2147483648 .. 2147483647
real	32-bit	$\pm 1.17549435082 * 10^{-38}$.. $\pm 6.80564774407 * 10^{38}$

ARRAYS

Syntax:
array[*index_start* .. *index_end*] **of** *type*

Example:
var
 weekdays : **array**[1..7] **of** byte;
 samples : **array**[10] **of** word;
begin
 // now we can access elements of array variables, for example:
 if weekdays[2] = 1 **then** *// if it is Tuesday then*
 samples[0] := 20 *// order 20 samples with index 0*
 else samples[3] := 10; *// else order 10 samples with index 3*
end;

CONSTANT ARRAYS

Example:
// Declare a constant array which holds no. of days
// in each month:
const MONTHS : **array**[1..12] **of** byte
 = (31,28,31,30,31,30,31,31,30,31,30,31);

STRINGS

Syntax:
string_name **string**[*length*];

Example:
var
 msg1 : **string**[20];
 msg2 : **string**[19];

POINTERS

To declare a pointer data type, add a carat prefix (^) before type. For example, if you are creating a pointer to an integer, you would write:

```
^integer;
```

A pointer can be assigned to another pointer. However, note that only the address, not the value, is copied.

begin
 msg1 := 'First message';
 msg2 := 'Second message';
 msg1 := msg2;

This is ok, but vice versa would be illegal (because of strings length).

Example:
var p : ^word;
 ...
 p^ := 5;

This will assign the pointed memory location value 5.

@ Operator

The @ operator returns the address of a variable or routine; that is, @ constructs a pointer to its operand. The following rules apply to @:

- If X is a variable, @X returns the address of X.
- If F is a routine (a function or procedure), @F returns F's entry point

RECORDS

Syntax:
type *recordTypeName* = **record**
 fieldList1 : *type1*;
 ...
 fieldListn : *typen*;
end;

Example:
type
 TDot = **record**
 x, y : real;
end;



mikroPascal Quick Reference Guide



Memory is allocated when you instantiate the record, like this:

```
var m, n: TDot;
```



Accessing Fields

Example:

```
m.x := 3.6;
m.y := 5;
```



Note:

You can commit assignments between complex variables, if they are of the same type:

```
n := m;
```

This will copy values of all fields.



OPERATORS

There are four types of operators in mikroPascal:

- Arithmetic Operators
- Bitwise Operators
- Boolean Operators
- Relational Operators

Operators Precedence and Associativity

Precedence	Operands	Operators	Associativity
4	1	@ not + -	right-to-left
3	2	* / div mod and shl shr	left-to-right
2	2	+ - or xor	left-to-right
1	2	= <> < > <= >=	left-to-right

Arithmetic Operators

Operator	Operation	Precedence
+	addition	2
-	subtraction	2
*	multiplication	3
/	division	3
div	division, rounds down to nearest integer (cannot be used with floating points)	3
mod	returns the remainder of integer division (cannot be used with floating points)	3

Relational Operators

Operator	Operation	Precedence
=	equal	1
<>	not equal	1
>	greater than	1
<	less than	1
>=	greater than or equal	1
<=	less than or equal	1



Note:

Use relational operators to test equality or inequality of expressions. All relational operators return TRUE or FALSE.

Bitwise Operators

Operator	Operation	Precedence
and	bitwise AND; returns 1 if both bits are 1, otherwise returns 0	3
or	bitwise (inclusive) OR; returns 1 if either or both bits are 1, otherwise returns 0	2
xor	bitwise exclusive OR (XOR); returns 1 if the bits are complementary, otherwise 0	2
not	bitwise complement (unary); inverts each bit	4
shl	bitwise shift left; moves the bits to the left, see below	3
shr	bitwise shift right; moves the bits to the right, see below	3

Examples:

```
operand1 :      %0001 0010
operand2 :      %0101 0110
-----
operator and :  %0001 0010
operator or  :  %0101 0110
operator xor  :  %0100 0100
```

Examples:

```
operand :      %0101 0110
-----
operator not  :  %1010 1001
operator shl  :  %1010 1100
operator shr  :  %0010 1011
```



Note:

With shift left (shl), left most bits are discarded, and "new" bits on the right are assigned zeroes. With shift right (shr), right most bits are discarded, and the "freed" bits on the left are assigned zeroes (in case of unsigned operand) or the value of the sign bit (in case of signed operand).

Boolean Operators

Operator	Operation
and	logical AND
or	logical OR
xor	logical exclusive OR
not	logical negation

These operators conform to standard Boolean logic. If used in conditional expressions they are compared with TRUE or FALSE.

Examples:

```
if (%1001 and %0111) = FALSE then LED1 := 1 else LED2 := 1;
```

Because expression (%1001 and %0111) gives %0001, when compared with FALSE (all zeros) it gives FALSE because they are not equal. It means that else statement will be executed and LED2 will be turned on. If it was written like this:

```
if (%1001 and %0111) then LED1 := 1 else LED2 := 1;
```

than expression (%1001 and %0111) is compared with TRUE (all ones) by default.

mikroPascal Quick Reference Guide

STATEMENTS

asm Statement

Syntax:

```
asm  
    block of assembly instructions  
end;
```

Assignment Statements

Syntax:

```
variable := expression;  
Example:  
counter := 1;
```

Compound Statements

Syntax:

```
begin  
    statements  
end;
```

Example:

```
i := 0;  
while i < 5 do  
    begin  
        temp := a[i];  
        a[i] := b[i];  
        b[i] := temp;  
        i := i + 1;  
    end;  
end.
```

This code will exchange values of 5 elements from arrays *a* and *b*.

Note:

In mikroPascal, the *end.* statement (the closing statement of every program) acts as an endless loop.



CONDITIONAL STATEMENTS

If Statement

Syntax:

```
if expression then statement1 [else statement2]
```

Example:

```
if movex = 1 then x := x+20 else y := y-10;
```

Note:

The *else* keyword with an alternate statement is optional.



Case Statement

Syntax:

```
case selector of  
    value_1 : statement_1;  
    ...  
    value_n : statement_n;  
    [else default_statement]  
end;
```

Example:

```
case input of  
    1 : LED1 := 1;  
    2 : LED2 := 1;  
    3 : LED3 := 1  
    else LED7 := 1;  
end;
```

This code will turn on LED depending of input value. If the value is different then ones mentioned in value list in case statement then else statement is executed by default. Semicolon not allowed if followed by else statement (applies for whole Pascal).

ITERATION STATEMENTS (LOOPS)

For Statement

Syntax:

```
for counter := initial_value to final_value do statement  
// or
```

```
for counter := initial_value downto final_value do statement
```

Example:

```
s := 0;  
for i := 0 to 4 do s := s + 2;
```

This code will add number 2 to variable *s* 5 times. At the end *s* will be 10.

JUMP STATEMENTS

While Statement

Syntax:

```
while expression do statement
```

This code will add number 2 to variable s 6 times. At the end s will be 12.

Example:

```
s := 0; i := 0;
while i < 6 do
begin
    s := s + 2; i := i + 1;
end;
```

Repeat Statement

Syntax:

```
repeat statement until expression
```

Example:

```
s := 0; i := 0;
repeat
begin
    s := s + 2;
    i := i + 1;
end;
until i = 7;
```

This code will add number 2 to variable s 7 times. At the end s will be 14.

Break Statement

Use the break statement within loops to pass control to the first statement following the innermost loop (for, while, or repeat block).

Example:

```
i := 0; s := 1; // initiate value of counter i and variable s
while TRUE do // infinite loop
begin
    if i = 4 then break;
    s := s * 2; i := i + 1;
end;
```

This code will multiply variable s with number 2 (until counter i becomes equal 4 and break statement executes). At the end s will be 16.

Continue Statement

You can use the continue statement within loops to skip the rest of the statements and jump to the first statement in loop.

Example:

```
i := 0; s := 1; // initiate value of counter i and variable s
while TRUE do // infinite loop
begin
    s := s * 2; i := i + 1;
    if i <> 4 then continue;
    break;
end;
```

This code will multiply variable s with number 2 (continue statement executes until counter i is not equal 4). At the end s will be 16.

Goto Statement

Syntax:

```
goto label_name;
```

Example:

```
label loop;
...
loop: Beep;
goto loop;
// infinite loop that
// calls the
// Beep procedure
```

Exit Statement

The exit statement allows you to break out of a routine (function or procedure). It passes the control to the first statement following the routine call.

Example:

```
procedure Proc1();
var error: byte;
begin
    ... // we're doing something here
    if error = TRUE then exit;
    ... // some code, which won't be
    .. // executed if error is true
end;
```