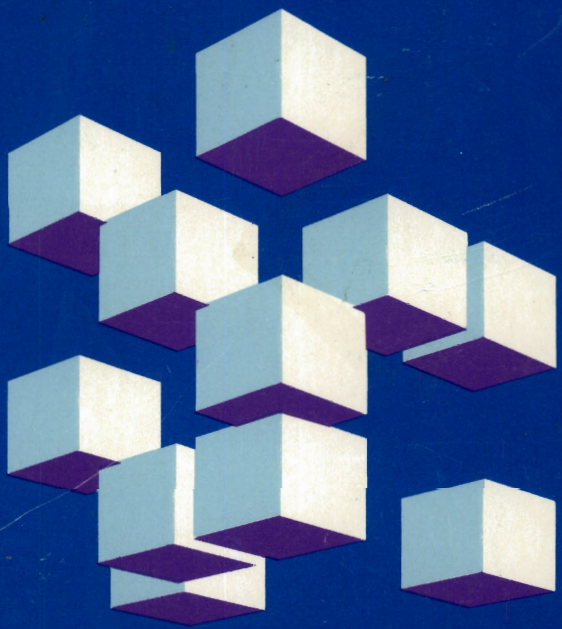
IBM

VS Pascal Reference Summary

Licensed Program



Order Number
SX26-3760-0

Program Numbers
5668-767
5668-717

Release 1

IBM

VS Pascal Reference Summary

Licensed Program

Order Number
SX26-3760-0

Program Numbers
5668-767
5668-717

Release 1

First Edition (June 1987)

This edition applies to Release 1 of VS Pascal, Licensed Programs 5668-767 (Compiler and Library) and 5668-717 (Library only), and to any subsequent releases until otherwise indicated in new editions or technical newsletters. This reference summary contains basic information from *VS Pascal Language Reference*, SC26-4320, and *VS Pascal Application Programming Guide*, SC26-4319.

Changes are made periodically to this publication; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's program may be used. Any functionally equivalent program may be used instead.

Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. If you request publications from the address given below, your order will be delayed because publications are not stocked there.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Contents

Operators

NOT Operators

Operator	Operation	Operands	Result
NOT (¬)	Boolean NOT	BOOLEAN	BOOLEAN
NOT (¬)	Logical one's complement	INTEGER	INTEGER
NOT (¬)	Set complement	SET OF t	SET OF t

Multiplication Operators

Operator	Operation	Operands	Result
*	Multiplication	INTEGER SHORTREAL REAL Mixed	INTEGER SHORTREAL REAL REAL
/	Real division	INTEGER SHORTREAL REAL Mixed	REAL SHORTREAL REAL REAL
DIV	Integer division	INTEGER	INTEGER
MOD	Modulo	INTEGER	INTEGER
AND (&)	Boolean AND	BOOLEAN	BOOLEAN
AND (&)	Logical AND	INTEGER	INTEGER
*	Set intersection	SET OF t	SET OF t
||	String concatenation	STRING	STRING
< <	Logical left shift	INTEGER	INTEGER
> >	Logical right shift	INTEGER	INTEGER

Addition Operators

Operator	Operation	Operands	Result
+	Addition	INTEGER SHORTREAL REAL Mixed	INTEGER SHORTREAL REAL REAL
+	Set union	SET OF t	SET OF t
-	Subtraction	INTEGER SHORTREAL REAL Mixed	INTEGER SHORTREAL REAL REAL
-	Set difference	SET OF t	SET OF t
OR (|)	Boolean OR	BOOLEAN	BOOLEAN
OR (|)	Logical OR	INTEGER	INTEGER
XOR (&&)	Boolean XOR	BOOLEAN	BOOLEAN
XOR (&&)	Logical XOR	INTEGER	INTEGER
XOR (&&)	Set exclusive union	SET OF t	SET OF t

Relational Operators

Operator	Operation	Operands	Result
=	Compare equal	Any set, scalar, pointer, or string	BOOLEAN
< > (¬ =)	Not equal	Any set, scalar, pointer, or string	BOOLEAN
<	Less than	Scalar type or string	BOOLEAN
< =	Compare < or =	Scalar type or string	BOOLEAN
< =	Subset	SET OF t	BOOLEAN
>	Compare greater	Scalar type or string	BOOLEAN
> =	Compare > or =	Scalar type or string	BOOLEAN
> =	Superset	SET OF t	BOOLEAN
IN	Set membership	t and SET OF t	BOOLEAN

Special Symbols

Special Symbol	Meaning
+	Addition and set union operator
-	Subtraction and set difference operator
*	Multiplication and set intersection operator
/	Division operator, REAL result only
¬	Boolean NOT operator, one's complement on INTEGER or set complement
|	Boolean OR operator, logical OR on INTEGER
&	Boolean AND operator, logical AND on INTEGER
&&	Boolean XOR operator, logical XOR on INTEGER and set exclusive union
=	Equality operator
<	Less than operator
< =	Less than or equal operator and set subset operator
> =	Greater than or equal operator and set superset operator
>	Greater than operator
< > or ¬ =	Not equal operator
< <	Left logical shift operator on INTEGER
> >	Right logical shift operator on INTEGER

Special Symbol	Meaning
||	Concatenation operator
:=	Assignment symbol
.	Period to end a unit or a field separator in a record
,	Comma, used as a list separator
:	Colon, used to specify a definition
;	Semicolon, used as a statement separator
..	Subrange notation
'	Quote, used to begin and end string constants
@ or ->	Pointer symbol
(	Left parenthesis
)	Right parenthesis
[or (.	Left square bracket for array indexes and set constructors
] or .)	Right square bracket for array indexes and set constructors
{ or (*	Comment left brace (standard)
} or *)	Comment right brace (standard)
/*	Comment left brace (alternate form)
*/	Comment right brace (alternate form)

Predefined Variables

Variable	Description
INPUT	Default input file.
OUTPUT	Default output file.

Predefined Constants

Constant	Description
ALFALEN	Length of type ALFA, value is 8
ALPHALEN	Length of type ALPHA, value is 16
FALSE	Constant of type BOOLEAN, FALSE < TRUE
MAXINT	Maximum value of type INTEGER: 2147483647
MAXREAL	Maximum value of type REAL: '7FFFFFFFFFFFFFF'XR
MININT	Minimum value of type INTEGER: -2147483648
MINREAL	Minimum nonzero value of type REAL: '0010000000000000'XR
TRUE	Constant of type BOOLEAN, TRUE > FALSE

Predefined Types

Type	Description
ALFA	PACKED ARRAY [1..ALFALEN] OF CHAR
ALPHA	PACKED ARRAY [1..ALPHALEN] OF CHAR
BOOLEAN	Data type composed of the values TRUE and FALSE
CHAR	Character data type
INTEGER	Integers in the range MININT..MAXINT
REAL	Long floating-point number represented in a 64-bit value
SHORTREAL	Short floating-point number represented in a 32-bit value
STRING	Array of CHAR whose length varies at run time up to a specified maximum
STRINGPTR	Pointer to a STRING whose maximum length is determined at run time
TEXT	File of CHAR

Predefined Functions

a = an array variable	*s* = a string expression
e = any expression	*t* = a type name or a variable name
f = a file variable	*v* = a variable
n = a positive integer expression	*x* = any arithmetic expression
p = pointer valued variable	

Predefined Function	Description
ABS(*x*)	Computes the absolute value of *x*.
ADDR(*v*)	Returns the location of variable *v*.
ARCTAN(*x*)	Returns the arctangent of *x*.
CHR(*n*)	Returns the EBCDIC character whose ordinal value is *n*.
CLOCK	Returns the number of microseconds of execution.
COLS(*f*)	Returns current column of file *f*.
COMPRESS(*s*)	Replaces multiple blanks in *s* with one blank.
COS(*x*)	Returns the cosine of *x*.
DELETE(*x*,*n1*[,*n2*])	Returns *s* with the *n2* characters starting at position *n1* removed.
EOF(*f*)	Test file *f* for end-of-file condition.
EOLN(*f*)	Test file *f* for end-of-line condition.

Predefined Function	Description
EXP(*x*)	Computes the base of the natural log (e) raised to the power *x*.
FLOAT(*x*)	Converts integer expression *x* to a floating-point value.
HBOUND(*a*[,*n*])	Determines the upper bound of array *a*.
HIGHEST(*t*)	Determines the maximum value of the type of a scalar *t*.
INDEX(*s1*,*s2*)	Returns the first location, if present, of *s2* in *s1*.
LBOUND(*a*[,*n*])	Determines the lower bound of array *a*.
LENGTH(*s*)	Determines the current length of string *s*.
LN(*x*)	Returns the natural logarithm of *x*.
LOWEST(*t*)	Determines the minimum value of the type of a scalar *t*.
LTRIM(*s*)	Returns *s* with leading blanks removed.
MAX(*e*[,*e*]...)	Determines the maximum value of scalar expression *e*.
MAXLENGTH(*s*)	Returns the maximum length of the parameter string *s*.
MIN(*e*[,e]...)	Determines the minimum value of scalar expression *e*.
ODD(*x*)	Returns TRUE if integer expression *x* is odd.
ORD(*e*)	Converts a scalar or pointer expression *e* to an integer.
PARMS	Returns the system dependent invocation parameters.
PRED(*e*)	Obtains the predecessor of scalar expression *e*.

Predefined Function	Description
RANDOM(*x*)	Returns a pseudorandom number, integer expression *x* is the seed value or zero.
RINDEX(*s1,s2*)	Returns the last location, if present, of *s2* in *s1*.
ROUND(*x*)	Converts floating-point expression *x* to an integer value by rounding.
SIN(*x*)	Returns the sine of *x*.
SIZEOF(*t*)	Determines the memory size of a variable or type *t*.
SQR(*x*)	Returns the square of *x*.
SQRT(*x*)	Returns the square root of *x*.
STR(*a*)	Converts array of characters *a* to a string.
SUBSTR(*s,n1*[,*n2*])	Returns the substring of *s* starting at *n1* with length *n2*.
SUCC(*e*)	Obtains the successor of scalar expression *e*.
TRIM(*s*)	Returns *s* with trailing blanks removed.
TRUNC(*x*)	Converts floating-point expression *x* to an integer by truncating.

Predefined Procedures

Predefined Procedure	Description
CLOSE(*f*)	Closes file *f*.
DATETIME(*a1*,*a2*)	Returns the current date in *a1* and time of day in *a2*.
DISPOSE(*p*)	Deallocates the dynamic variable pointed to by *p*.
GET(*f*)	Advances file pointer to the next element of input file *f*.
HALT	Halts the program execution.
LTOKEN(*v*,*s1*,*s2*)	Extracts tokens from string *s1* updating starting position *v*. The result is returned in string *s2*.
MARK(*p*)	Creates a new heap, *p*.
NEW(*p*)	Allocates a dynamic variable in the current heap and sets *p* to point to the variable.
PACK(*a1*,*e*,*a2*)	Copies array *a1* starting at index *e* to packed array *a2*.
PAGE[(*f*)]	Skips to the top of the next page.
PDSIN(*f*[,*s*])	Opens file *f* for input. *s* designates the open options and the member name of the PDS.
PDSOUT(*f*[,*s*])	Opens file *f* for output. *s* designates the open options and the member name of the PDS.
PUT(*f*)	Advances the file pointer to the next element of output file *f*.

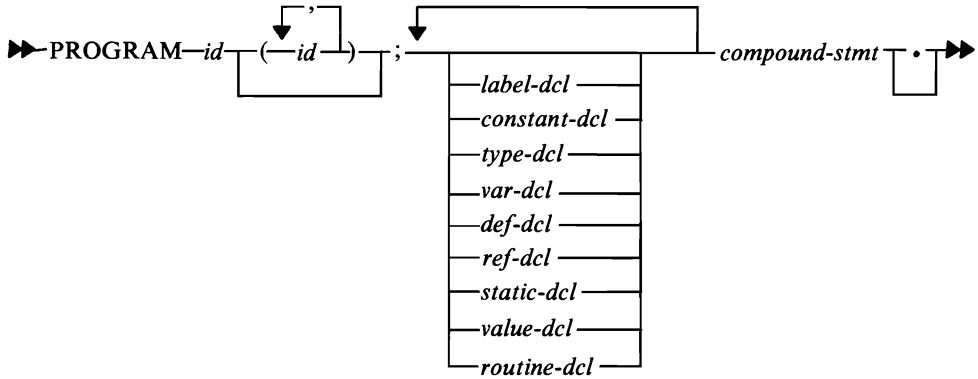
Predefined Procedure	Description
READ([*f*,]*v*[,*v*]...)	Reads from file *f* into variable *v*.
READLN[([*f*,]*v*[,*v*]...)]	Reads variable *v* and then skips to end-of-line of TEXT file *f*.
READSTR(*s*,*v*[,*v*]...)	Reads data from string *s* into variable *v*.
RELEASE(*p*)	Destroys one or more heaps. *p* is the last heap to be destroyed.
RESET(*f*[,*s*])	Opens file *f* for input with open options *s*.
RETCODE(*n*)	Sets the system return code.
REWRITE(*f*[,*s*])	Opens file *f* for output with open options *s*.
SEEK(*f*,*n*)	Positions file *f* to component number *n*.
TERMIN(*f*[,*s*])	Opens file *f* for input from the terminal with open options *s*.
TERMOUT(*f*[,*s*])	Opens file *f* for output to the terminal with open options *s*.
TOKEN(*v*,*s*,*a*)	Extracts tokens from string *s* updating starting position *v*. The result is returned in ALPHA *a*.
TRACE(*f*)	Writes the procedure and function invocation history to file *f*.
UNPACK(*a1*,*a2*,*e*)	Copies packed array *a1* to array *a2* beginning at index *e*.
UPDATE(*f*[,*s*])	Opens file *f* for both input and output with open options *s*.
WRITE([*f*,]*e*[,*e*]...)	Writes the value of *e* to file *f*.
WRITELN[([*f*,]*e*[,*e*]...)]	Writes the value of *e* and then writes an end-of-line to TEXT file *f*.
WRITESTR(*s*,*e*[,*e*]...)	Writes the value of *e* to string *s*.

Reserved Words

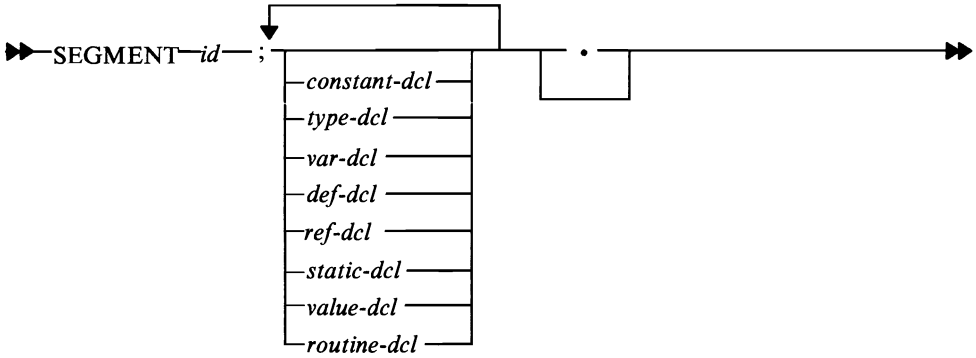
AND	END	OF	SPACE
ARRAY	FILE	OR	STATIC
ASSERT	FOR	OTHERWISE	THEN
BEGIN	FUNCTION	PACKED	TO
CASE	GOTO	PROCEDURE	TYPE
CONST	IF	PROGRAM	UNTIL
CONTINUE	IN	RANGE	VALUE
DEF	LABEL	RECORD	VAR
DIV	LEAVE	REF	WHILE
DO	MOD	REPEAT	WITH
DOWNTO	NIL	RETURN	XOR
ELSE	NOT	SET	

Compilable Units

Program-Unit:



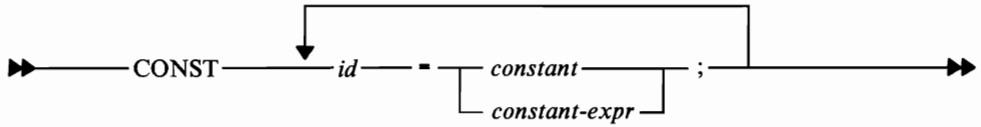
Segment-Unit:



Declarations

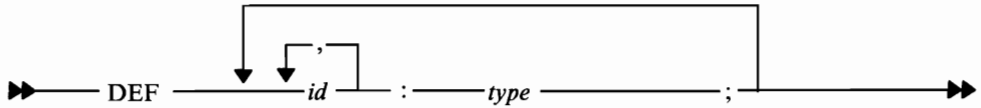
CONST Declaration

Used to assign identifiers for constant expressions.



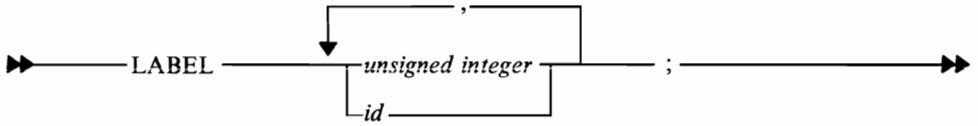
DEF Declaration

Used to declare external variables.



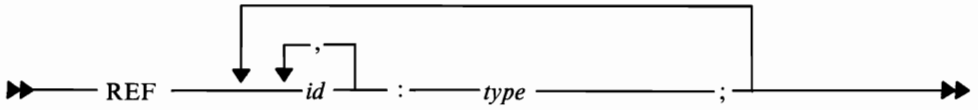
LABEL Declaration

Used to declare labels which will appear in the routine.



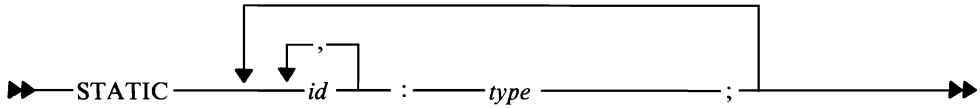
REF Declaration

Used to declare external variables.



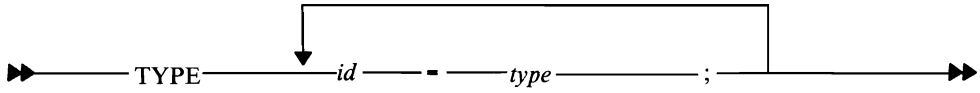
STATIC Declaration

Used to declare static variables.



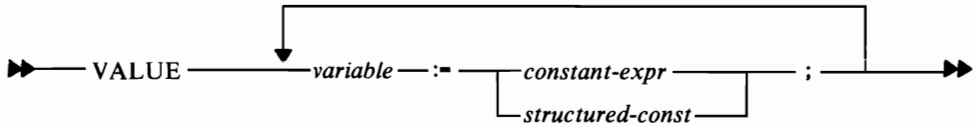
TYPE Declaration

Used to define a data type and assign a name to that type.



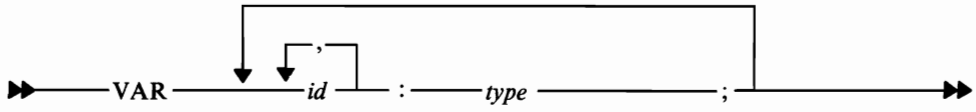
VALUE Declaration

Used to specify an initial value for STATIC and DEF variables.



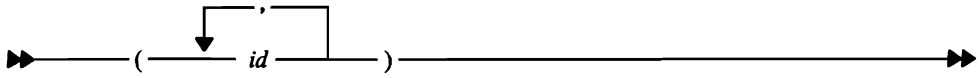
VAR Declaration

Used to declare automatic variables.

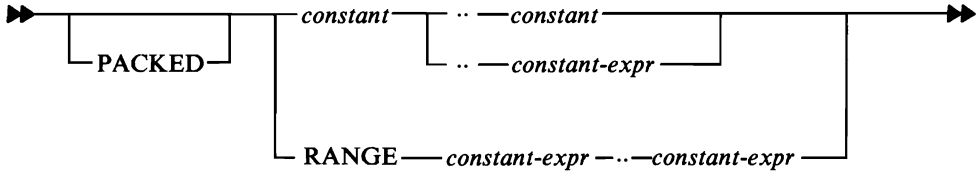


Data Types

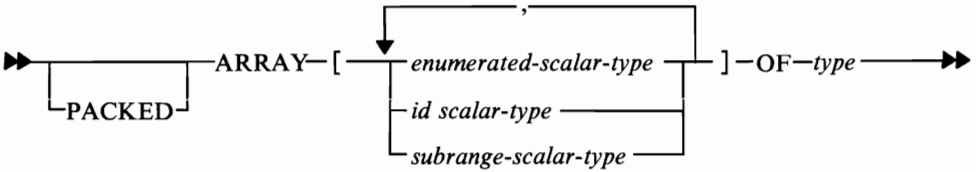
Enumerated Scalar Data Type



Subrange Scalar Data Type

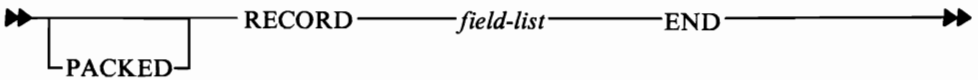


ARRAY Data Type

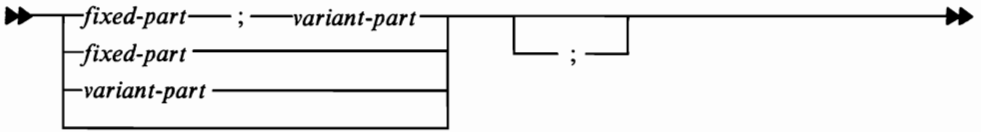


RECORD Data Type

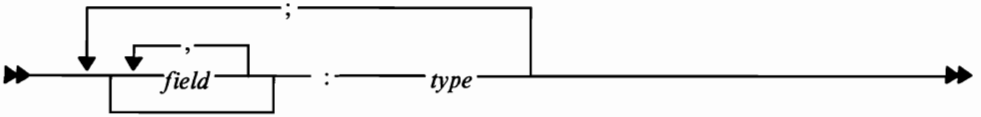
Record-type:



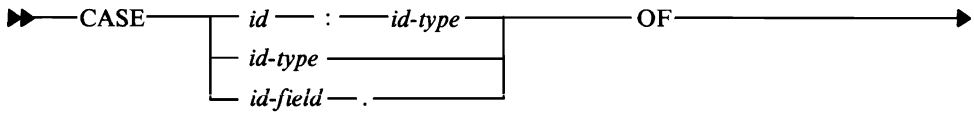
Field-list:

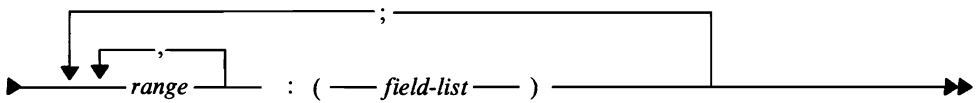


Fixed-part:

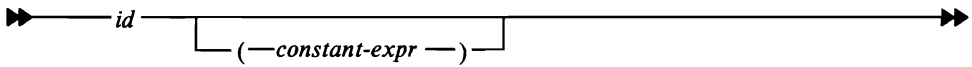


Variant-part:

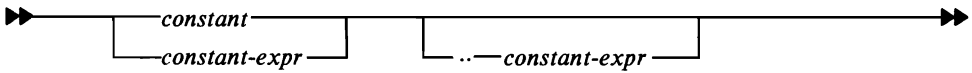




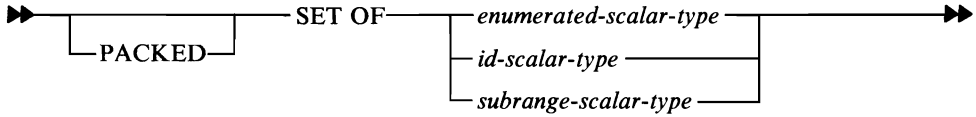
Field:



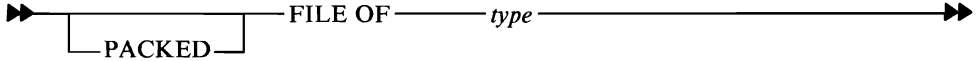
Range:



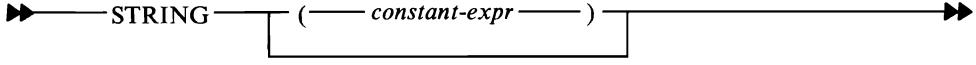
SET Data Type



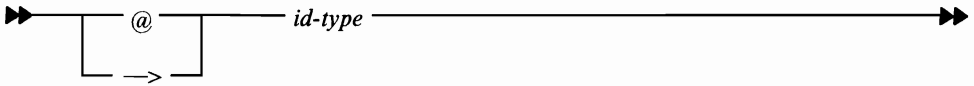
FILE Data Type



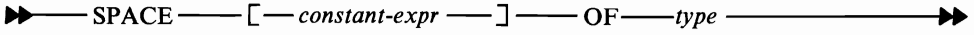
STRING Data Type



Pointer Data Type

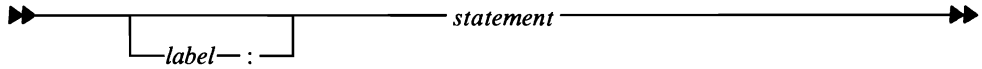


SPACE Data Type



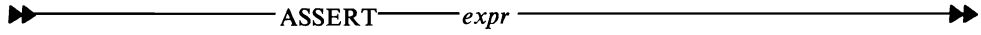
Statements

The diagram below shows the form for the VS Pascal statements.



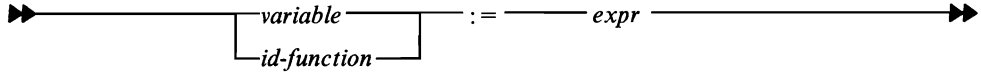
ASSERT

Used to check for a specific condition and signal a run-time error if the condition is not met.



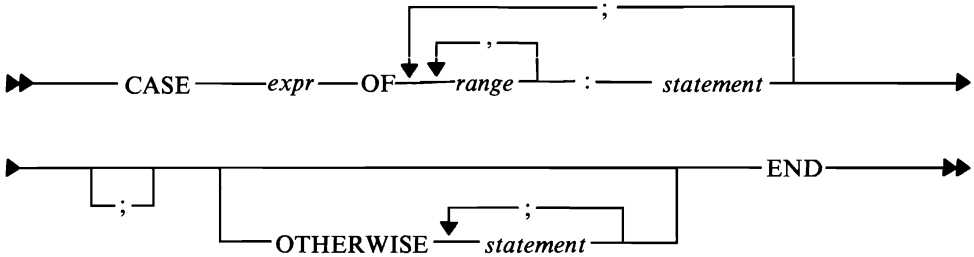
assignment

Used to assign a value to a variable.

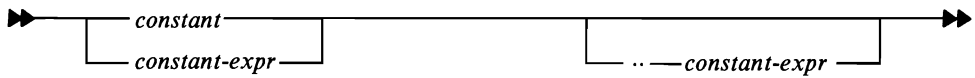


CASE

Provides you with a multiple branch based upon the evaluation of an expression.

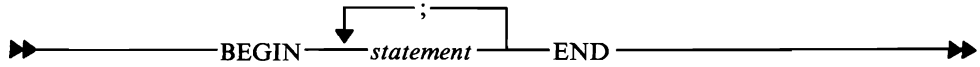


Range:



Compound

Serves to bracket a series of statements that are to be executed sequentially.



CONTINUE

Causes a jump to the loop-continuation portion of the innermost enclosing FOR, WHILE, or REPEAT statement.

empty

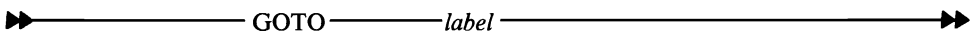
Used as a place holder and has no effect on the execution of the program.

FOR

Repeatedly executes a statement while a control variable is assigned a series of values.

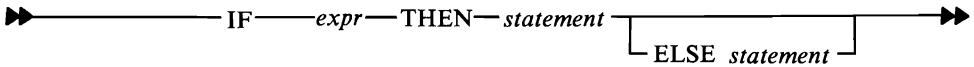
GOTO

Change the flow of control within the program.



IF

Allows you to specify that one of two statements is to be executed depending on the evaluation of a Boolean expression.

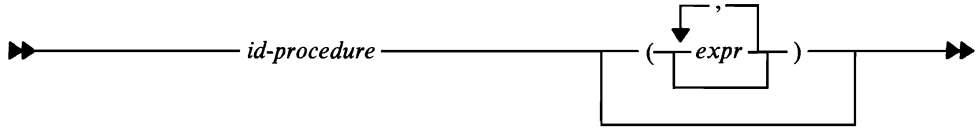


LEAVE

Causes an immediate, unconditional exit from the innermost enclosing FOR, WHILE, or REPEAT loop.

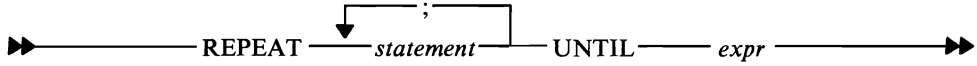
Procedure Call

Causes the invocation of a procedure.



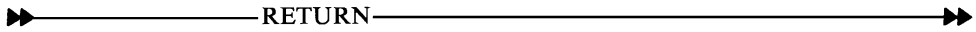
REPEAT

Causes the statements between the REPEAT and UNTIL keywords to be executed until the control expression evaluates to true.



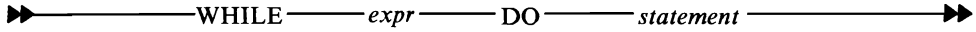
RETURN

Permits an exit from a procedure or function.



WHILE

Allows you to specify a statement that is to be executed while a control expression evaluates to true.



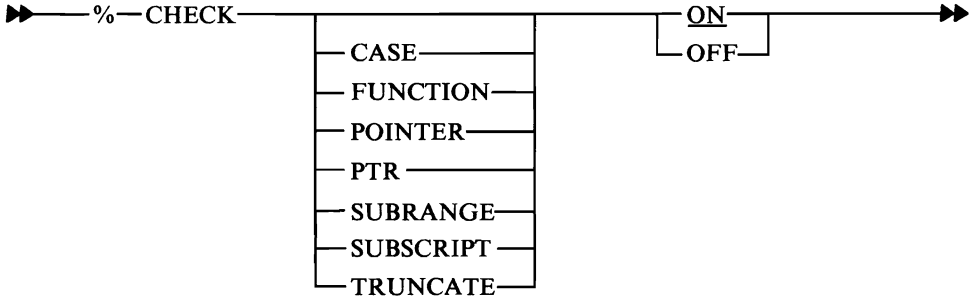
WITH

Used to simplify references to a record variable by eliminating an addressing description on every field reference.

Compiler Directives

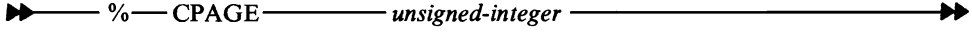
%CHECK

Allows you to enable or disable run-time checking features of VS Pascal.



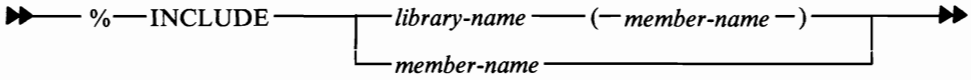
%CPAGE

Forces a page eject if there are less than a specified number of lines left on the current page of the output listing.



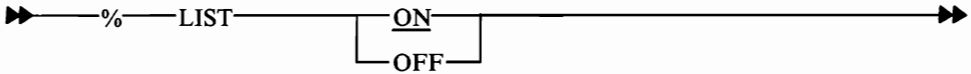
%INCLUDE

Causes source from a library file to be inserted into the input stream immediately after the current line.



%LIST

Allows you to enable or disable the pseudo-assembler listing of the VS Pascal compiler.

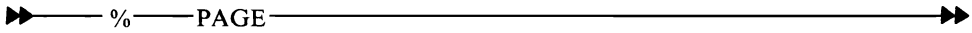


%MARGINS

Redefines the left and right margins of the compiler input.

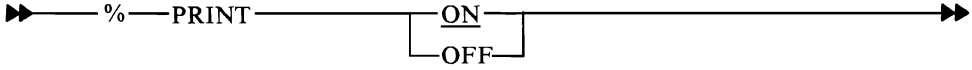
%PAGE

Forces a skip to the next page on the output listing.



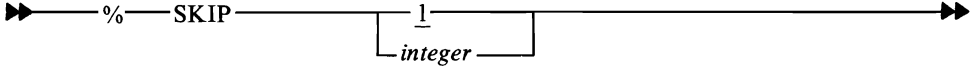
%PRINT

Used to print or not print the source statements.



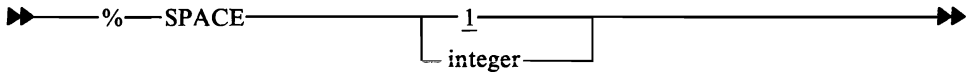
%SKIP

Used to insert one or more blank lines in the source listing.



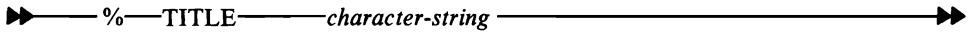
%SPACE

Used to insert one or more blank lines in the source listing.



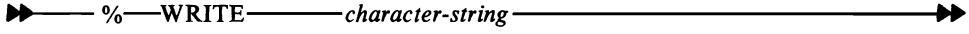
%TITLE

Used to set the title in the listing.

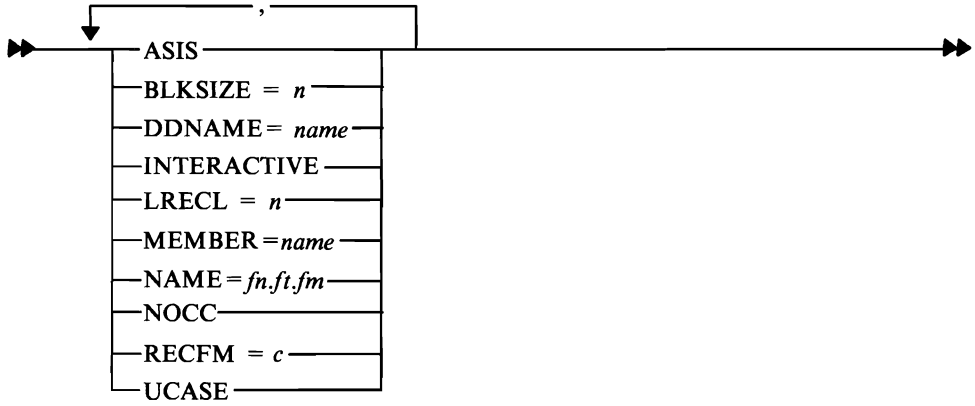


%WRITE

Allows a message to be written to the terminal at a specified location in the program during compilation.



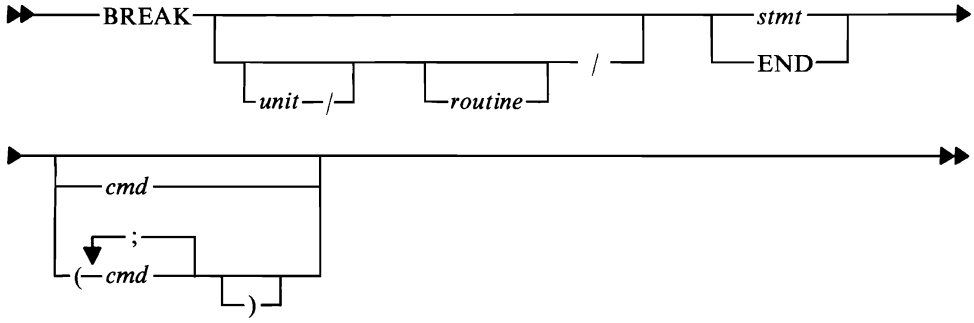
The Open Options



Interactive Debugging Tool Commands

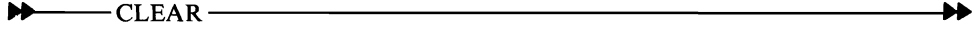
BREAK

Causes a break point to be set at the indicated statement.



CLEAR

Used to remove all break points.

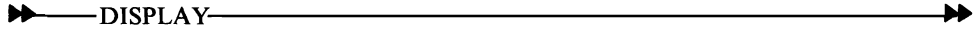


CMS

Activates CMS subset mode.

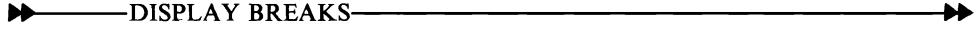
DISPLAY

Used to display information about the current debugging session.



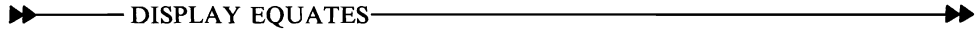
DISPLAY BREAKS

Used to produce a list of all break points which are currently set.



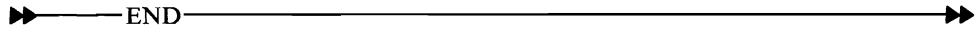
DISPLAY EQUATES

Used to produce a list of all equate symbols and their current definitions.



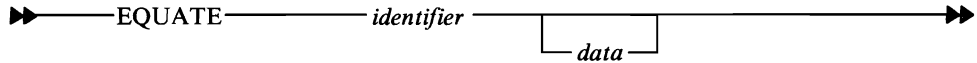
END

Causes the program to terminate immediately.



EQUATE

Equates an identifier name to a data string.

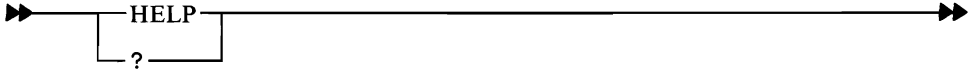


GO

Causes the program to either start or resume executing.

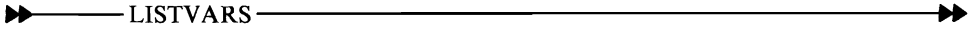
HELP

Lists all the debugging commands.



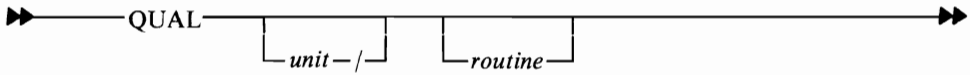
LISTVARS

Displays the values of all variables which are local to the currently active routine.



QUAL

Used to specify the scope of variables to be viewed.

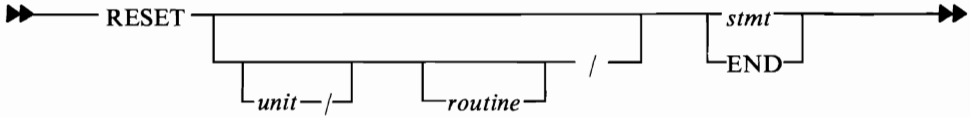


QUIT

Causes the program to terminate immediately.

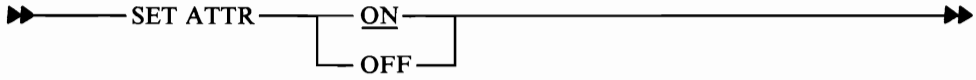
RESET

Used to remove a break point.



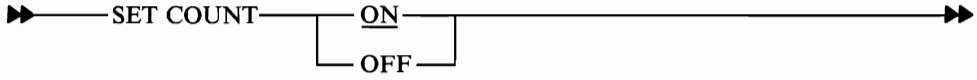
SET ATTR

Used to set the default way in which variables are viewed.



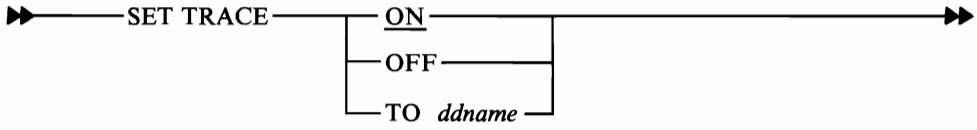
SET COUNT

Used to initiate and terminate statement counting.



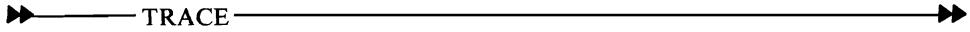
SET TRACE

Used to either activate or deactivate program tracing.



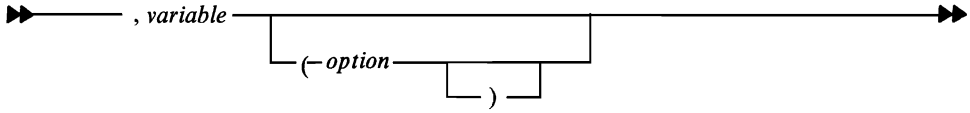
TRACE

Used to produce a routine trace at the terminal.



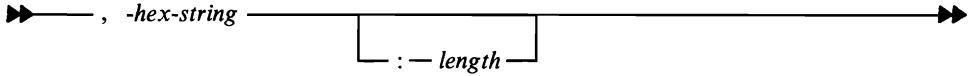
Viewing Variables

Allows you to obtain contents of a variable during program execution.



Viewing Storage

Used to display contents of a specific storage location.



WALK

Causes the program to execute one statement.

Run-Time Options

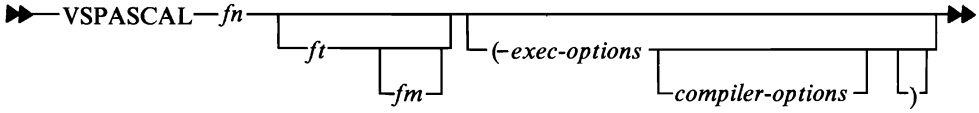
Run-Time Option	Description
COUNT	Causes statement frequency information to be collected during program execution and written to file OUTPUT. This option is effective only if the program was compiled with the DEBUG option and link-edited with the debugging library.
DEBUG	Causes the interactive debugging tool to gain initial control when you invoke your program.
ERRCOUNT=*n* ERRCOUNT(*n*)	Specifies the number of nonfatal run-time errors that may occur before terminating the program.
ERRFILE=*ddname* ERRFILE(*ddname*)	Specifies the file that error diagnostics are written to.
HEAP=*n* HEAP(*n*)	Specifies the number of kilobytes the heap is to be extended on overflow.
MAINT	Includes system run-time routines in any error track backs.
NOCHECK	Causes all checking errors to be ignored.
NOSPIE	Suppresses the interception of program exceptions.
SETMEM	Initializes a routine's local storage to a set value on each entry.
STACK=*n* STACK(*n*)	Specifies the number of kilobytes the stack is to be extended on overflow.

Compiler Options

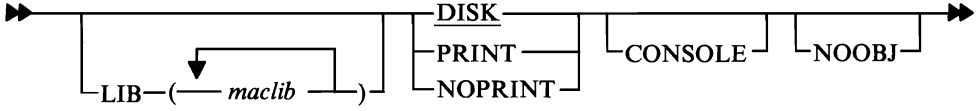
Compiler Option	Abbreviated Name	Default
CHECK | NOCHECK	—	CHECK
DDNAME(COMPAT) | DDNAME(UNIQUE)	—	DDNAME(COMPAT)
DEBUG | NODEBUG	—	NODEBUG
FLAG(I|W|E|S)	—	FLAG(I)
GOSTMT | NOGOSTMT	GS|NOGS	GOSTMT
GRAPHIC | NOGRAPHIC	—	NOGRAPHIC
LANGLVL(ANSI83) | LANGLVL(EXTENDED)	—	LANGLVL(EXTENDED)
LINECOUNT(*n*)	LC(*n*)	LINECOUNT(60)
LIST | NOLIST	—	NOLIST
MARGINS(*m,n*)	MAR(*m,n*)	MARGINS(1,72)
OPTIMIZE | NOOPTIMIZE	OPT | NOOPT	OPTIMIZE
PAGEWIDTH(*n*)	PW(*n*)	PAGEWIDTH(128)
PXREF | NOPXREF	—	PXREF
SEQUENCE(*m,n*) | NOSEQUENCE	SEQ(*m,n*) | NOSEQ	SEQUENCE(73,80)

Compiler Option	Abbreviated Name	Default
SOURCE | NOSOURCE	S|NOS	SOURCE
STDFLAG(I|W|E|S)	—	STDFLAG(E) for LANGLVL(ANSI83)
WRITE | NOWRITE		NOWRITE
XREF(SHORT) | XREF(LONG) | NOXREF	X | NOX	XREF(SHORT)

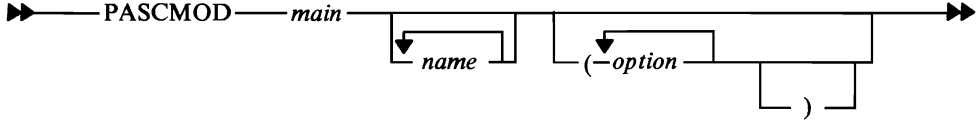
VSPASCAL EXEC



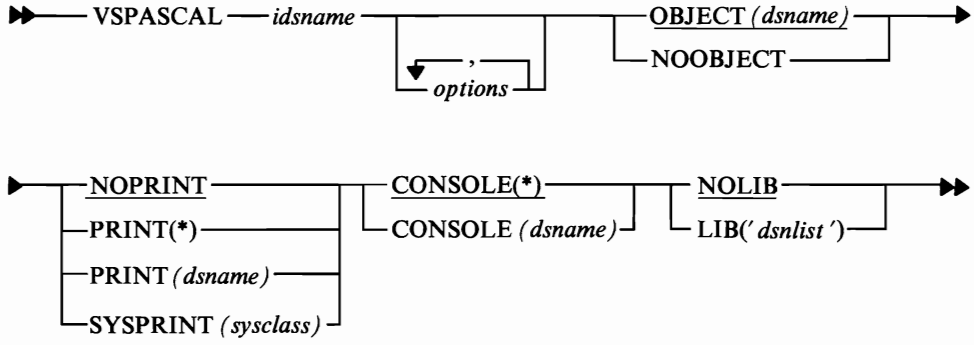
Exec-options:



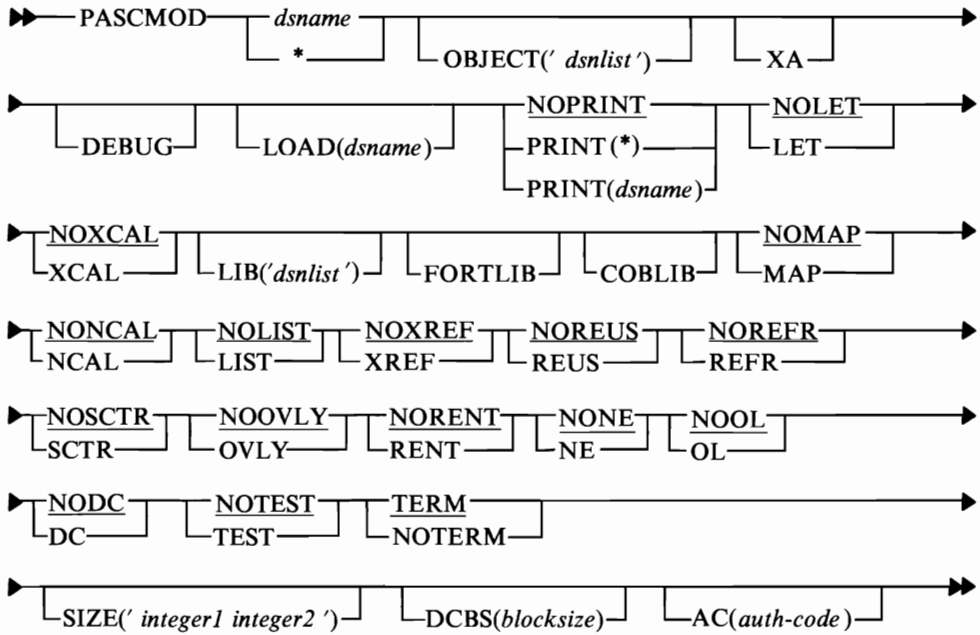
PASCMOD EXEC



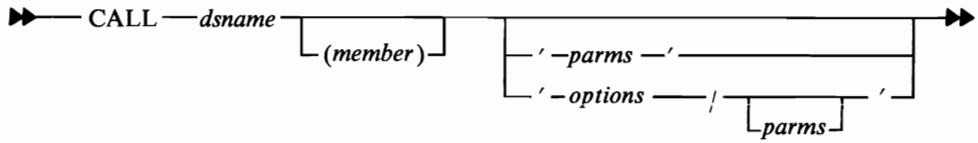
VSPASCAL CLIST



PASCMOD CLIST



CALL Command



VS Pascal
Reference Summary

File Number
S370-40

Printed in U.S.A.

SX26-3760-00