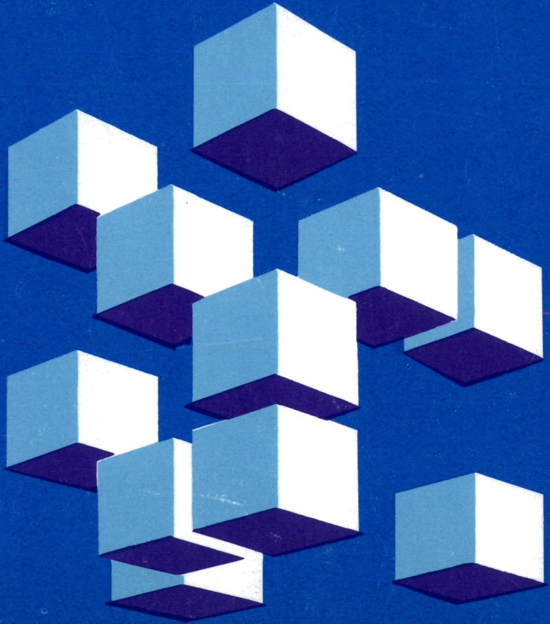
IBM

SX26-3760-1

VS Pascal

Reference Summary

Release 2



IBM

VS Pascal

Reference Summary

Release 2

SX26-3760-1

Second Edition (December 1988)

This edition replaces and makes obsolete the previous edition, SX26-3760-0.

This edition applies to Release 2 of VS Pascal, Program Number 5668-767 (Compiler and Library) and 5668-717 (Library only) and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

Specific changes for this edition are indicated by a vertical bar to the left of the change. These bars will be deleted at any republication of the page affected. Editorial changes that have no technical significance are not noted.

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, 4300, and 9370 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 Reader's Comment Form is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publishing, P. O. Box 49023, San Jose, California, U.S.A. 95161-9023. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Contents

Operators 1
  NOT Operators 1
  Multiplication Operators 2
  Addition Operators 3
  Relational Operators 4
Special Symbols 5
Predefined Variables and Constants 7
Predefined Types 8
Predefined Routines 9
Additional Routines 15
Open Options 16
Reserved Words 17
Compilable Units 18
Declarations 20
  Routine Declarations 20
  CONST 23
  DEF 24
  LABEL 24
  REF 25
  STATIC 25
  TYPE 26
  VALUE 26
  VAR 26
Data Types 27
  ARRAY 27

Enumerated Scalar 27
FILE 28
GSTRING 28
Pointer 28
RECORD 29
SET 31
SPACE 31
STRING 31
Subrange Scalar 31

Routine Directives 32

Statements 33
ASSERT 33
assignment 33
CASE 34
Compound 35
CONTINUE 35
empty 35
FOR 36
GOTO 36
IF 36
LEAVE 37
Procedure Call 37
REPEAT 38
RETURN 38
WHILE 38
WITH 39

Compiler Directives 40
%CHECK 40

%CPAGE . . . 40
| %ENDSELECT . . . 41
%INCLUDE . . . 41
%LIST . . . 41
%MARGINS . . . 42
%PAGE . . . 42
%PRINT . . . 42
| %SELECT . . . 43
%SKIP . . . 43
%SPACE . . . 43
%TITLE . . . 44
| %UHEADER . . . 44
| %WHEN . . . 44
%WRITE . . . 45
Interactive Debugging Tool Commands . . . 46
BREAK . . . 46
CLEAR . . . 47
CMS . . . 47
DISPLAY . . . 47
DISPLAY BREAKS . . . 48
| DISPLAY COUNTS . . . 48
DISPLAY EQUATES . . . 48
END . . . 49
EQUATE . . . 49
GO . . . 49
HELP . . . 50
LISTVARS . . . 50
QUAL . . . 50

QUIT 51
RESET 51
SET ATTR 51
SET COUNT 52
SET TRACE 52
TRACE 52
Viewing Storage 53
Viewing Variables 53
WALK 53
VSPASCAL EXEC 54
PASCMOD EXEC 55
| PASCRUN EXEC 56
VSPASCAL CLIST 57
PASCMOD CLIST 58
CALL Command 59
| Compile-Time Options 60
| Link-Edit Options 63
Run-Time Options 64

Operators

NOT Operators

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

Multiplication Operators

Operator	Operation	Operands	Result
*	Multiplication	INTEGER SHORTREAL REAL Mixed	INTEGER SHORTREAL REAL REAL
*	Set intersection	SET OF t	SET OF t
/	Real division	INTEGER SHORTREAL REAL Mixed	REAL SHORTREAL REAL REAL
DIV	Integer division	INTEGER	INTEGER
MOD	Modulo	INTEGER	INTEGER
AND or &	Boolean AND	BOOLEAN	BOOLEAN
AND or &	Logical AND	INTEGER	INTEGER
<<	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
+ or ||	String concatenation	STRING GSTRING	STRING GSTRING
−	Subtraction	INTEGER SHORTREAL REAL Mixed	INTEGER SHORTREAL REAL REAL
−	Set difference	SET OF t	SET OF t
OR or |	Boolean OR	BOOLEAN	BOOLEAN
OR or |	Logical OR	INTEGER	INTEGER
> < or XOR or &&	Boolean XOR	BOOLEAN	BOOLEAN
> < or XOR or &&	Logical XOR	INTEGER	INTEGER
> < or XOR or &&	Set symmetric difference	SET OF t	SET OF t

Relational Operators

Operator	Operation	Operands	Result
=	Compare equal	Any set, scalar, pointer, or string	BOOLEAN
< > or ¬=	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, set union, and string concatenation operator
-	Subtraction and set difference operator
*	Multiplication and set intersection operator
/	Division operator, real result only
¬	Boolean NOT operator, one's complement on integer, and set complement
|	Boolean OR operator, and logical OR on integer
&	Boolean AND operator, and logical AND on integer
> < or &&	Boolean XOR operator, logical XOR on integer, and set symmetric difference
=	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
||	Concatenation operator

Special Symbol	**Meaning**
:=	Assignment symbol
.	Period, used to end a unit, and 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, used for parameter lists and mathematical grouping
)	Right parenthesis, used for parameter lists and mathematical grouping
[or (.	Left square bracket, used for array indexes and set constructors
] or .)	Right square bracket, used 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 and Constants

Item	Type	Description
ALFALEN	Constant	Length of type ALFA, value is 8
ALPHALEN	Constant	Length of type ALPHA, value is 16
EPSREAL	Constant	Constant of type REAL, representing the smallest number such that 1.0 + EPSREAL > 1.0: '3310000000000000'XR
FALSE	Constant	Constant of type BOOLEAN, FALSE < TRUE
INPUT	Variable	Default input file
MAXCHAR	Constant	Maximum value of type CHAR: 'FF'XC
MAXINT	Constant	Maximum value of type INTEGER: 2147483647
MAXREAL	Constant	Maximum value of type REAL: '7FFFFFFFFFFFFFFF'XR
MININT	Constant	Minimum value of type INTEGER: -2147483648
MINREAL	Constant	Minimum nonzero value of type REAL: '0010000000000000'XR
OUTPUT	Variable	Default output file
TRUE	Constant	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
GCHAR	fit = 1.DBCS character data type
GSTRING	Array of GCHAR whose length varies at run time up to a specified maximum
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 Routines

a = an array variable	*p* = pointer valued variable
e = any expression	*r* = a floating-point value
f = a file variable	*s* = an SBCS or a DBCS string expression
gc = a DBCS character, character array or string	fit=1.*sc* = an SBCS character, character array or string
gs = a DBCS string expression	*ss* = an SBCS string expression
i = an integer value	*t* = a type name or a variable name
ms = a mixed DBCS/SBCS string expression	*v* = a variable
n = a positive integer expression	*x* = any arithmetic expression

Routine	Type	Description
ABS(*x*)	Function	Computes the absolute value of *x*
ADDR(*v*)	Function	Returns the location of *v*
ARCTAN(*x*)	Function	Returns the arctangent of *x*
CHR(*n*)	Function	Returns the EBCDIC character whose ordinal value is *n*
CLOCK	Function	Returns the number of microseconds of execution
CLOSE(*f*)	Procedure	Closes file *f*
COLS(*f*)	Function	Returns current column of file *f*
COMPRESS(*s*)	Function	Replaces multiple blanks in *s* with one blank
COS(*x*)	Function	Returns the cosine of *x*

Routine	**Type**	**Description**
DATETIME(*a1*,*a2*)	Procedure	Returns the current date in *a1* and time of day in *a2*
DELETE(*s*,*n1*[,*n2*])	Function	Returns *s* with the *n2* characters starting at position *n1* removed
DISPOSE(*p*)	Procedure	Deallocates the dynamic variable pointed to by *p*, which is the pointer returned from a previous call to NEW
DISPOSEHEAP(*p*)	Procedure	Deallocates the heap identified by *p*, which is the identifier returned from a previous call to NEWHEAP
EOF(*f*)	Function	Tests file *f* for end-of-file condition
EOLN(*f*)	Function	Tests file *f* for end-of-line condition
EXP(*x*)	Function	Computes the base of the natural log (e) raised to the power *x*
FLOAT(*i*)	Function	Converts *i* to a floating-point value
GET(*f*)	Procedure	Advances the file pointer to the next element of input file *f*
GSTR(*gc*)	Function	Converts *gc* to a GSTRING
GTOSTR(*gs*)	Function	Converts *gs* to a mixed string
HALT	Procedure	Halts the program execution
HBOUND(*a*[,*n*])	Function	Determines the upper bound of *a*
HIGHEST(*t*)	Function	Determines the maximum value of the type of a scalar *t*
INDEX(*s1*,*s2*)	Function	Returns the first location, if present, of *s2* in *s1*
LBOUND(*a*[,*n*])	Function	Determines the lower bound of *a*
LENGTH(*s*)	Function	Determines the current length of *s*
LN(*x*)	Function	Returns the natural logarithm of *x*

Routine	**Type**	**Description**
LOWEST(*t*)	Function	Determines the minimum value of the type of a scalar *t*
LPAD(*s*)	Procedure	Pads or truncates *s* on the left
LTOKEN(*v*,*ss1*,*ss2*)	Procedure	Extracts tokens from string *ss1* updating starting position *v* ; the result is returned in string *ss2*
LTRIM(*s*)	Function	Returns *s* with leading blanks removed
MARK(*p*)	Procedure	Creates a new subheap, *p*, in the active heap
MAX(*e*[,*e*]...)	Function	Determines the maximum value of scalar expression *e*
MAXLENGTH(*s*)	Function	Returns the maximum length of *s*
MCOMPRESS(*ms*)	Function	Returns *ms*, with sequences of SBCS blanks replaced with a single SBCS blank, and sequences of DBCS blanks replaced with a single DBCS blank
MDELETE(*ms*,*n1*[,*n2*])	Function	Returns *ms*, with the *n2* characters starting at position *n1* removed
MIN(*e*[,e]...)	Function	Determines the minimum value of scalar expression *e*
MINDEX(*ms1*,*ms2*)	Function	Returns the first location, if present, of *ms2* in *ms1*
MLENGTH(*ms*)	Function	Returns the length of *ms*
MLTRIM(*ms*)	Function	Returns *ms*, with leading SBCS and DBCS blanks removed
MRINDEX(*ms1*,*ms2*)	Function	Returns the last location, if present, of *ms2* in *ms1*
MSUBSTR(*ms*,*n1*[,*n2*])	Function	Returns the substring of *ms* starting at position *n1* with length *n2*
MTRIM(*ms*)	Function	Returns *ms* with trailing SBCS and DBCS blanks removed
NEW(*p*)	Procedure	Allocates a dynamic variable in the current heap and sets *p* to point to the variable

Routine	**Type**	**Description**
NEWHEAP(*p*[,*ss*])	Procedure	Allocates a new heap and returns *p*, the identifier of that heap; *ss* can include, in any combination separated by commas, LOC=ANY|BELOW (location above or below the 16-megabyte line), INIT=nnn (initial heap size), INCR=nnn (extension size on heap overflow), DISP=KEEP|FREE (disposal specifications)
ODD(*i*)	Function	Returns TRUE if *i* is odd
ORD(*e*)	Function	Converts a scalar or pointer expression *e* to an integer
PACK(*a1*,*e*,*a2*)	Procedure	Copies *a1* starting at index *e* to packed *a2*
PAGE[(*f*)]	Procedure	Skips to the top of the next page
PARMS	Function	Returns the system-dependent invocation parameters
PDSIN(*f*[,*ss*])	Procedure	Opens file *f* for input. *ss* designates the open options and the member name of the PDS
PDSOUT(*f*[,*ss*])	Procedure	Opens file *f* for output. *ss* designates the open options and the member name of the PDS
PRED(*e*)	Function	Obtains the predecessor of ordinal expression *e*
PUT(*f*)	Procedure	Advances the file pointer to the next element of output file *f*
QUERYHEAP(*p*)	Procedure	Sets *p* to the heap-id of the current heap
RANDOM(*i*)	Function	Returns a pseudorandom number; *i* is the seed value or zero
READ([*f*,]*v*[,*v*]...)	Procedure	Reads from file *f* into *v*
READLN[([*f*,]*v*[,*v*]...)]	Procedure	Reads *v* and skips to end-of-line of text file *f*
READSTR(*ss*,*v*[,*v*]...)	Procedure	Reads data from *ss* into *v*

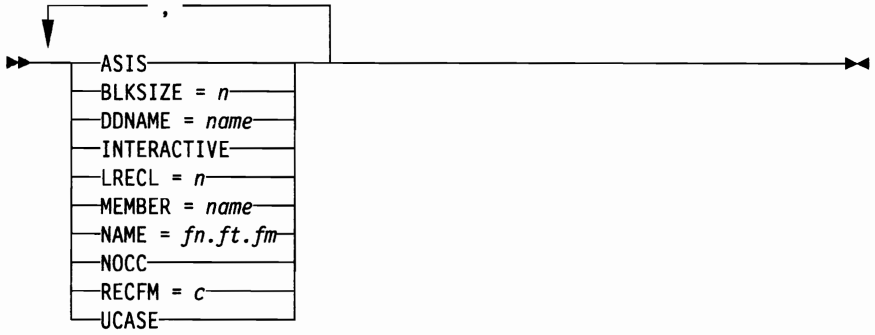
Routine	**Type**	**Description**
RELEASE(*p*)	Procedure	Deallocates one or more subheaps. *p* is the last subheap to be deallocated
RESET(*f*[,*ss*])	Procedure	Opens file *f* for input with open options *ss*
RETCODE(*i*)	Procedure	Sets the system return code
REWRITE(*f*[,*ss*])	Procedure	Opens file *f* for output with open options *ss*
RINDEX(*s1*,*s2*)	Function	Returns the last location, if present, of *s2* in *s1*
ROUND(*r*)	Function	Converts *r* to an integer by rounding
RPAD(*s*)	Procedure	Pads or truncates *s* on the right
SEEK(*f*,*n*)	Procedure	Positions file *f* to component number *n*
SIN(*x*)	Function	Returns the sine of *x*
SIZEOF(*t*)	Function	Determines the memory size of a variable or type *t*
SQR(*x*)	Function	Returns the square of *x*
SQRT(*x*)	Function	Returns the square root of *x*
STOGSTR(*ms*)	Function	Converts *ms* to a GSTRING
STR(*sc*)	Function	Converts *sc* to a STRING
SUBSTR(*s*,*n1*[,*n2*])	Function	Returns the substring of *s* starting at position *n1* with length *n2*
SUCC(*e*)	Function	Obtains the successor of ordinal expression *e*
TERMIN(*f*[,*ss*])	Procedure	Opens file *f* for input from the terminal with open options *ss*
TERMOUT(*f*[,*ss*])	Procedure	Opens file *f* for output to the terminal with open options *ss*

Routine	Type	Description
TOKEN(*v*,*ss*,*a*)	Procedure	Extracts tokens from *ss*, updating starting position *v*; the result is returned in ALPHA *a*
TRACE(*f*)	Procedure	Writes the procedure and function invocation history to file *f*
TRIM(*s*)	Function	Returns *s* with trailing blanks removed
TRUNC(*r*)	Function	Converts *r* to an integer by truncating
UNPACK(*a1*,*a2*,*e*)	Procedure	Copies packed *a1* to *a2* beginning at index *e*
UPDATE(*f*[,*ss*])	Procedure	Opens file *f* for both input and output with open options *ss*
USEHEAP(*p*)	Procedure	Changes the current heap to *p*, which is a heap-id returned from a previous call to NEWHEAP
WRITE([*f*,]*e*[,*e*]...)	Procedure	Writes the value of *e* to file *f*
WRITELN[([*f*,]*e*[,*e*]...)]	Procedure	Writes the value of *e* and then writes an end-of-line to text file *f*
WRITESTR(*ss*,*e*[,*e*]...)	Procedure	Writes the value of *e* to *ss*

Additional Routines

Routine	Type	Description
CMS(*ss*, *i*)	Procedure	Issues a CMS command and sets *i* to the command's return code
ITOHS(*i*)	Function	Converts *i* to a hexadecimal string
ONERROR	Procedure	When run-time errors occur, performs any necessary action before generating an error message
PICTURE(*r*, *ss*)	Function	Formats *r* according to a "picture" format *ss*

Open Options

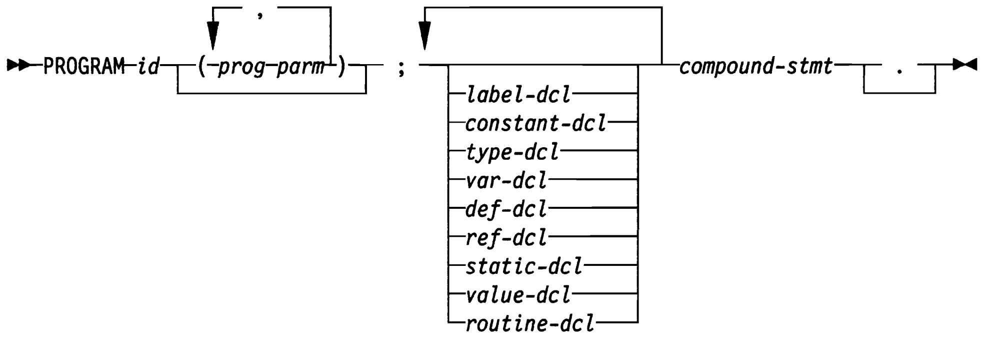


Reserved Words

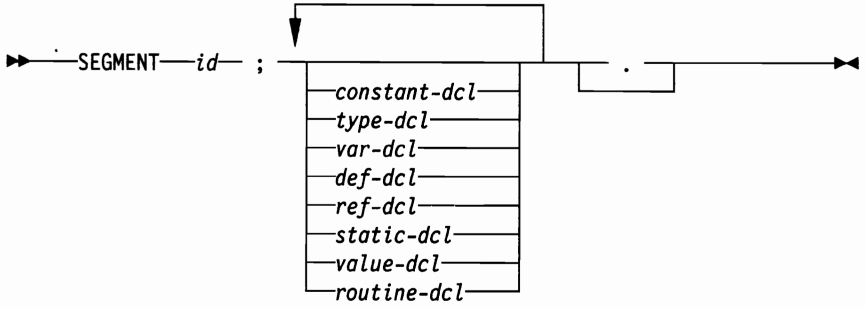
AND	END	OF	SPACE
ARRAY	FILE	OR	STATIC
ASSERT	FOR	OTHERWISE	THEN
BEGIN	FUNCTION	PACKED	TO
CASE	GOTO	PROCEDURE	TYPE
CONST	IF	PROGRAM	UNTIL
CONTINUE	IN	RANGE	VALUE
DEF	LABEL	RECORD	VAR
DIV	LEAVE	REF	WHILE
DO	MOD	REPEAT	WITH
DOWNTO	NIL	RETURN	XOR
ELSE	NOT	SET	

Compilable Units

Program Unit



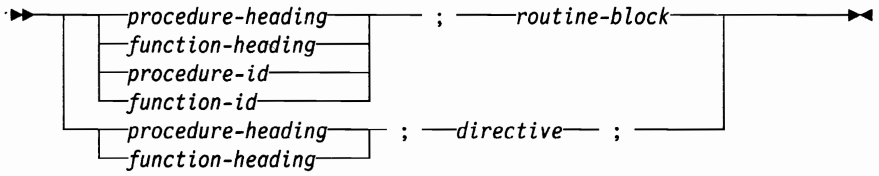
Segment Unit



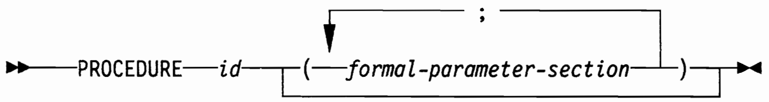
Declarations

Routine Declarations

Routine Declaration



Procedure Heading



Function Heading

```
                          ┌───────────── . ─────────────┐
                          ▼                              │
▶▶──FUNCTION──id──┬─(──formal-parameter-section──────────┴─)─┬──:─id-type──▶◀
                  └──────────────────────────────────────────┘
```

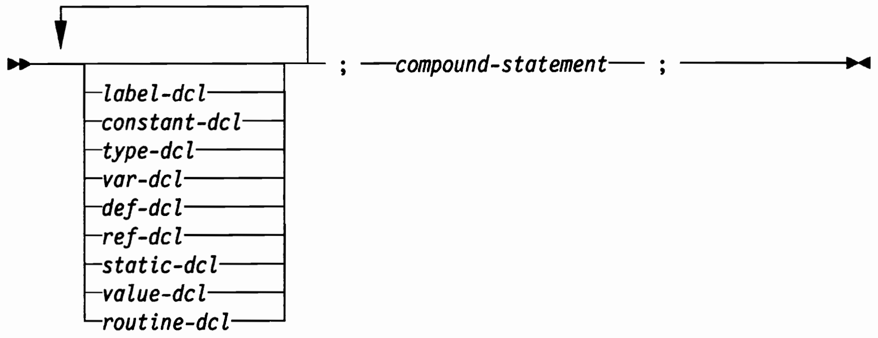
Procedure-id

```
▶▶──PROCEDURE──id──────────────────────────────────────────────▶◀
```

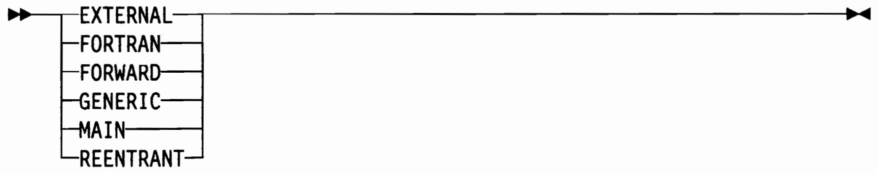
Function-id

```
▶▶──FUNCTION──id───────────────────────────────────────────────▶◀
```

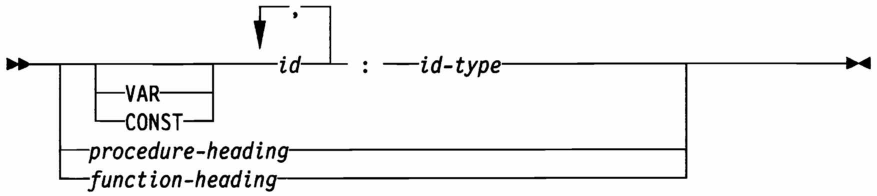
Routine Block



Directive

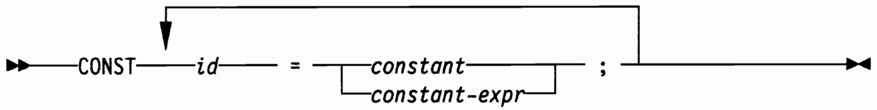


Formal Parameter Section



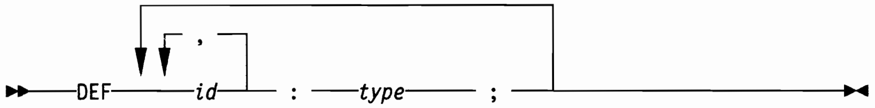
CONST

Assigns identifiers for constant expressions.



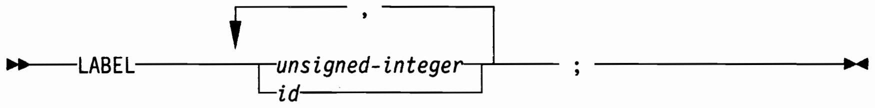
DEF

Declares external variables.



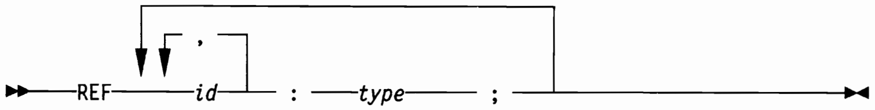
LABEL

Declares labels which will appear in the routine.



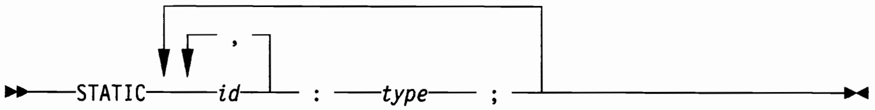
REF

Declares external variables.



STATIC

Declares static variables.



TYPE

Defines a data type and assigns a name to that type.

```
            ┌──────────────────────────┐
            ▼                          │
▶▶──TYPE────id── = ──type── ; ─────────┴──────────────▶◀
```

VALUE

Specifies an initial value for static and DEF variables.

```
            ┌───────────────────────────────────────────────────┐
            ▼                                                   │
▶▶──VALUE───variable── := ──┬─constant-expr────┬── ; ───────────┴─────▶◀
                            └─structured-const─┘
```

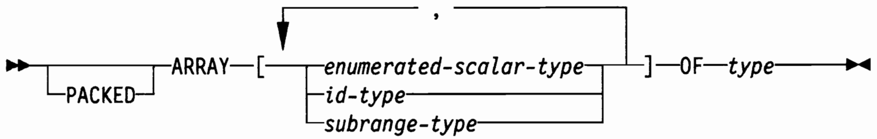
VAR

Declares automatic variables.

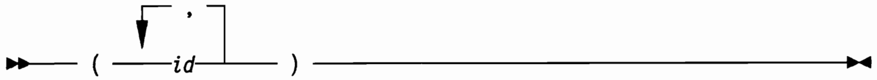
```
          ┌──────────────────────────────────┐
          │ ┌─ , ─┐                          │
          ▼ ▼     │                          │
▶▶──VAR─────id────┴── : ──type── ; ──────────┴───────────────▶◀
```

Data Types

ARRAY



Enumerated Scalar



FILE

```
►►──┬────────┬──FILE OF──type──────────────────────────►◄
    └─PACKED─┘
```

GSTRING

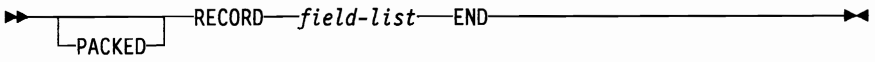
```
| ►►─GSTRING──(──constant-expr──)───────────────────────►◄
```

Pointer

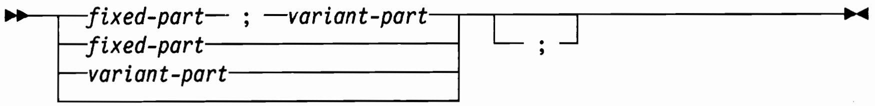
```
►►──┬─ @  ─┬──id-type───────────────────────────────────►◄
    └─ -> ─┘
```

RECORD

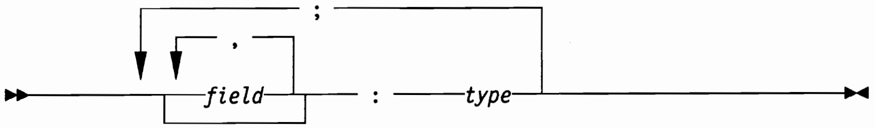
Record-type



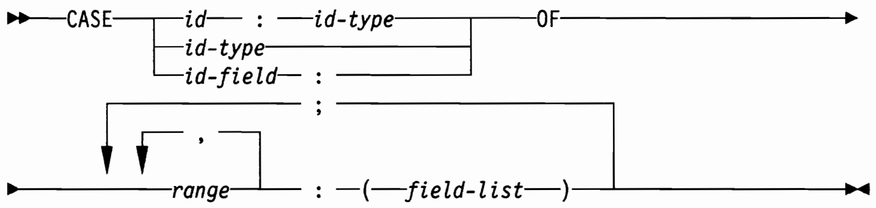
Field-list



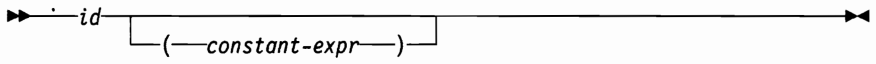
Fixed-part



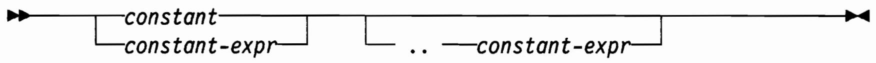
Variant-part



Field



Range



SET

```
►►──┬──────────┬──SET OF──┬─enumerated-scalar-type─┬──────►◄
    └─PACKED───┘          ├─id-type────────────────┤
                          └─subrange-type──────────┘
```

SPACE

```
►►──SPACE──[──constant-expr──]──OF──type───────────────────►◄
```

STRING

```
►►──STRING────(──constant-expr──)──────────────────────────►◄
```

Subrange Scalar

```
►►──┬──────────┬──┬─constant──┬─ .. ──constant──────┬──┬───►◄
    └─PACKED───┘  │           └─ .. ──constant-expr─┘  │
                  └─RANGE──constant-expr── .. ──constant-expr─┘
```

Routine Directives

Directive	Description
EXTERNAL	Identifies a procedure or function that can be invoked from outside of its lexical scope (such as another unit)
FORTRAN	Identifies a non-Pascal routine that is defined outside the unit being compiled
FORWARD	Identifies a routine whose heading is being declared in advance of its body
GENERIC	Identifies routines from other software products that can be called by VS Pascal
MAIN	Identifies a Pascal procedure that can be invoked as if it were a main program
REENTRANT	Identifies a Pascal procedure that can be invoked as if it were a main program

Statements

The following diagram shows the syntax of VS Pascal statements.

```
►►──────┬──────────────┬─────statement───────────────────────────►◄
        └─label─ : ────┘
```

ASSERT

Checks for a specific condition and signals a run-time error if the condition is not met.

```
►►────ASSERT────expr─────────────────────────────────────────────►◄
```

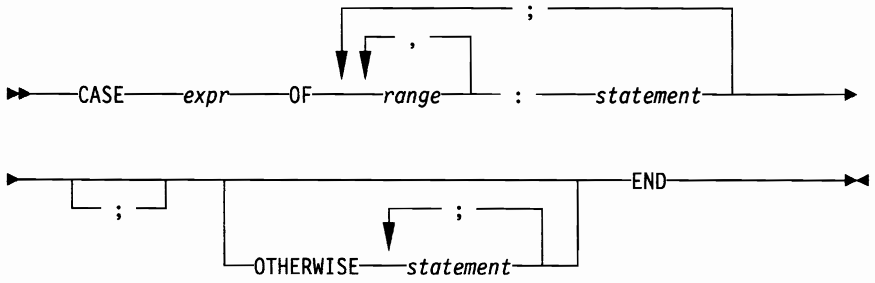
assignment

Assigns a value to a variable.

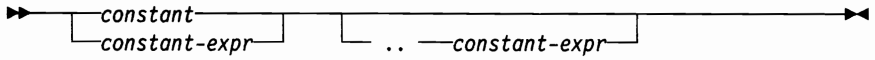
```
►►──────┬─variable─────┬───── := ───expr─────────────────────────►◄
        └─id-function──┘
```

CASE

Provides 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.

```
                  ┌──── ; ─────┐
                  ▼            │
►►───BEGIN───────statement─────┴───END────────────────────────────►◄
```

CONTINUE

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

```
►►───CONTINUE─────────────────────────────────────────────────────►◄
```

empty

A place holder; it has no effect on the execution of the program.

```
►►────────────────────────────────────────────────────────────────►◄
```

FOR

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

```
►►──FOR─id-var─ := ─expr──┬─TO─────┬─expr─DO─statement─────►◄
                          └─DOWNTO─┘
```

GOTO

Changes the flow of control within the program.

```
►►──GOTO───label─────────────────────────────────────────►◄
```

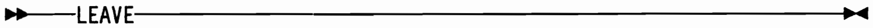
IF

Specifies that one of two statements is to be executed, depending on the evaluation of a Boolean expression.

```
►►──IF───expr───THEN───statement──┬────────────────┬─────►◄
                                  └─ELSE─statement─┘
```

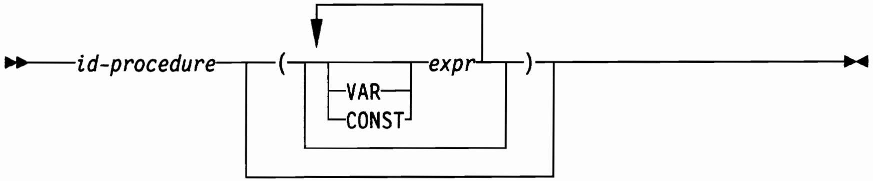
LEAVE

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



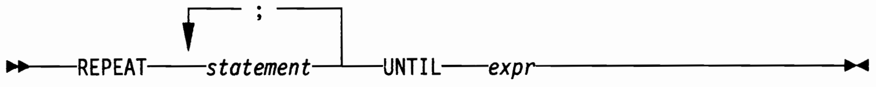
Procedure Call

Invokes a procedure.



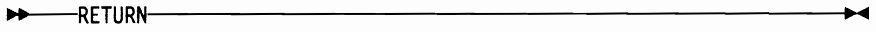
REPEAT

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



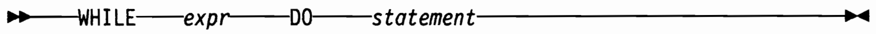
RETURN

Permits an exit from a procedure or function.



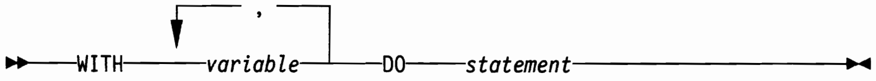
WHILE

Specifies a statement that is to be executed while a control expression evaluates to true.



WITH

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



Compiler Directives

%CHECK

Enables or disables run-time checking features of VS Pascal.

```
▶▶────%───CHECK──────┬───────────────┬─────┬─ON──┬──────────────────────▶◀
                     ├──CASE─────────┤     └─OFF─┘
                     ├──FUNCTION─────┤
                     ├──POINTER──────┤
                     ├──PTR──────────┤
                     ├──SUBRANGE─────┤
                     ├──SUBSCRIPT────┤
                     └──TRUNCATE─────┘
```

%CPAGE

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

```
▶▶────%───CPAGE─────unsigned-integer─────────────────────────────────────▶◀
```

%ENDSELECT

Marks the end of a section of code set aside for conditional compilation.

```
►►──%──ENDSELECT──────────────────────────────────────────────►◄
```

%INCLUDE

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

```
►►──%──INCLUDE──┬─library-name──(─member-name─)─┬──────────►◄
                └─member-name───────────────────┘
```

%LIST

Controls whether the pseudo-assembler listing is generated.

```
►►──%──LIST──┬─ON──┬──────────────────────────────────────►◄
             └─OFF─┘
```

%MARGINS

Redefines the left and right margins of the compiler input.

```
►►──%──MARGINS──integer1──integer2─────────────────►◄
```

%PAGE

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

```
►►──%──PAGE────────────────────────────────────────►◄
```

%PRINT

Controls whether source statements are printed.

```
►►──%──PRINT─┬─ON──┬───────────────────────────────►◄
             └─OFF─┘
```

%SELECT

Marks the start of a section of code set aside for conditional compilation.

```
▶▶──%──SELECT──────────────────────────────────────────────────▶◀
```

%SKIP

Inserts one or more blank lines in the source listing.

```
▶▶──%──SKIP───┬─1───────┬──────────────────────────────────────▶◀
              └─integer─┘
```

%SPACE

Inserts one or more blank lines in the source listing.

```
▶▶──%──SPACE──┬─1───────┬──────────────────────────────────────▶◀
              └─integer─┘
```

%TITLE

Places a title in the listing.

```
▶▶──%──TITLE──character-string──────────────────────▶◀
```

%UHEADER

Places a user-written character string in the routine header of the generated code.

```
▶▶──%──UHEADER──────character string──────────────────────▶◀
```

%WHEN

Controls the conditions under which a portion of code set aside for conditional compilation is actually compiled.

```
▶▶──%──WHEN ────Boolean-expression──────────────────▶◀
```

%WRITE

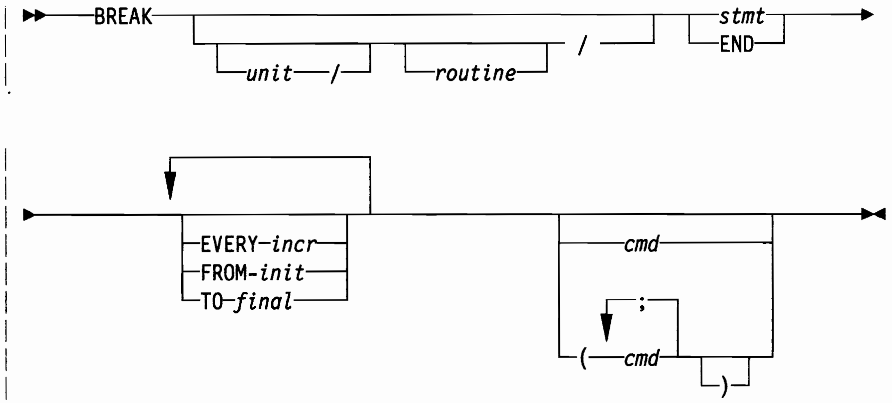
Allows a message to be written to the terminal at a specified location in the program during compilation.

```
►►——%——WRITE——character-string————————————————►◄
```

Interactive Debugging Tool Commands

BREAK

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



CLEAR

Removes all break points.

```
►►──CLEAR──────────────────────────────────────────────►◄
```

CMS

Activates CMS subset mode.

```
►►──CMS────────────────────────────────────────────────►◄
```

DISPLAY

Displays information about the current debugging session.

```
►►──DISPLAY────────────────────────────────────────────►◄
```

DISPLAY BREAKS

Produces a list of all break points which are currently set.

```
▶▶──DISPLAY BREAKS──────────────────────────────────▶◀
```

DISPLAY COUNTS

Displays the statistics kept by the COUNT run-time option.

```
▶▶──DISPLAY COUNTS──────────────────────────────────▶◀
```

DISPLAY EQUATES

Produces a list of all equate symbols and their current definitions.

```
▶▶──DISPLAY EQUATES─────────────────────────────────▶◀
```

END

Causes the program to terminate immediately.

```
►►──END──────────────────────────────────────────────────────►◄
```

EQUATE

Equates an identifier name to a data string.

```
►►──EQUATE──identifier──┬──────┬─────────────────────────────►◄
                        └─data─┘
```

GO

Causes the program to either start or resume executing.

```
►►──GO───────────────────────────────────────────────────────►◄
```

HELP

Lists all the debugging commands.

```
►►───┬─HELP─┬─────────────────────────────────────────────►◄
     └─?────┘
```

LISTVARS

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

```
►►─────LISTVARS───────────────────────────────────────────►◄
```

QUAL

Specifies the scope of variables to be viewed.

```
►►─────QUAL──┬────────────┬──┬───────────┬────────────────►◄
             └─unit───/───┘  └─routine───┘
```

QUIT

Causes the program to terminate immediately.

```
►►──QUIT──────────────────────────────────────────────────────►◄
```

RESET

Removes a break point.

```
►►──RESET──┬─────────────────────────────────────┬──┬─stmt─┬──────►◄
           └─┬──────────────┬─┬──────────────┬─/─┘  └─END──┘
             └─unit────/────┘ └─routine──────┘
```

SET ATTR

Sets the default for viewing variables.

```
►►──SET ATTR──┬─ON──┬─────────────────────────────────────────►◄
              └─OFF─┘
```

SET COUNT

Initiates and terminates statement counting.

```
►►──SET COUNT──┬─ON──┬──────────────────────────────►◄
               └─OFF─┘
```

SET TRACE

Either activates or deactivates program tracing.

```
►►──SET TRACE──┬─ON─────────┬───────────────────────►◄
               ├─OFF────────┤
               └─TO ddname──┘
```

TRACE

Produces a routine trace at the terminal.

```
►►──TRACE───────────────────────────────────────────►◄
```

Viewing Storage

Displays the contents of a specific storage location.

```
▶▶── , ─hex-string─┬──────────────┬──────────────────▶◀
                   └─ : ──length──┘
```

Viewing Variables

Displays the contents of a variable during program execution.

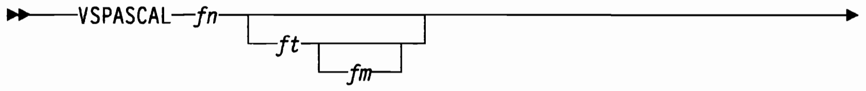
```
▶▶── , ─variable─┬─────────────────────┬──────────────▶◀
                 └─(──option──┬────┬───┘
                              └─)──┘
```

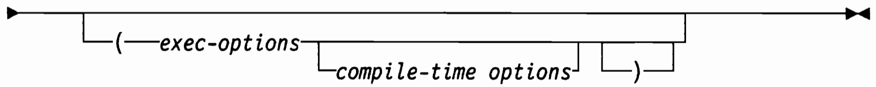
WALK

Causes the program to execute one statement.

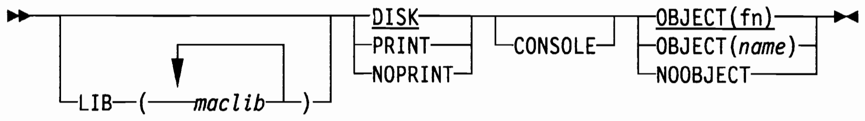
```
▶▶───WALK──────────────────────────────────────────────▶◀
```

VSPASCAL EXEC





Exec-options



PASCMOD EXEC

```
►►──PASCMOD─main─┬──────────┬───link-edit-options──┬───────────────┬──►◄
                 │ ┌─────┐  │                      │               │
                 │ ▼     │  │                      │               │
                 └─name──┴──┘                      └─NAME─modname──┘
```

PASCRUN EXEC

```
►►─PASCRUN─main──────────┬─parms─────────────────────────┬──────────────►◄
                         └─run-time-options─/─┬───────┬─┘
                                              └─parms─┘
```

VSPASCAL CLIST

```
►►──VSPASCAL──idsname──clist-options──┬──────────────────────┬──►◄
                                      └─compile-time-options─┘
```

Clist options

```
►►─┬─NOLIB───────────┬────┬─NOPRINT────────────┬────►
   └─LIB('dsnlist')──┘    ├─PRINT(*)───────────┤
                          ├─PRINT(dsname)──────┤
                          └─SYSPRINT(sysclass)─┘

►──┬─CONSOLE(*)──────┬────┬─OBJECT(dsname)─┬────►◄
   └─CONSOLE(dsname)─┘    └─NOOBJECT───────┘
```

PASCMOD CLIST

```
►►─PASCMOD──┬─dsname─┬──clist-options──linkage-editor─options──►◄
            └───*────┘
```

Clist-Options

```
►►─┬──────────────────┬─┬──────────────┬─┬─NODEBUG─┬─┬─NOTRANLIB─┬─┬─NOXA─┬─►◄
   └OBJECT('dsnlist')─┘ └LIB('dsnlist')┘ └─DEBUG───┘ └─TRANLIB───┘ └─XA───┘
```

CALL Command

```
►►──CALL──dsname──┬──────────┬──┬──────────────────────────────────────┬──►◄
                  └─(member)─┘  ├─'─parms──────'───────────────────────┤
                                └─'─run-time options─/─┬───────┬─'─────┘
                                                       └─parms─┘
```

Compile-Time Options

Compile-Time Option	Abbreviated Name	IBM-Supplied Default	Description
CHECK | NOCHECK	—	CHECK	Enables or disables run-time error checking
CONDPARM(*parmname1*= '*string*' [,*parmname2*= '*string*']...)	—	—	Controls when selected sections of source code are compiled
DDNAME(COMPAT|UNIQUE)	—	DDNAME(COMPAT)	Controls how VS Pascal generates ddnames for files
DEBUG | NODEBUG	—	NODEBUG	Controls whether the compiler prepares the module for debugging with the interactive debugging tool
FLAG(I|W|E|S)	—	FLAG(I)	Controls which messages (informational, warning, error, or severe error) are listed
GOSTMT | NOGOSTMT	GS|NOGS	GOSTMT	Controls whether a statement table is included within the object code
GRAPHIC | NOGRAPHIC	—	NOGRAPHIC	Controls whether the compiler recognizes the shift-out and shift-in characters as bracketing double-byte characters in string literals, comments and compiler directives

Compile-Time Option	Abbreviated Name	IBM-Supplied Default	Description
HEADER | NOHEADER	—	HEADER	Controls whether a header appears above the generated code of each routine
LANGLVL(ANSI83|EXTENDED)	—	LANGLVL(EXTENDED)	Controls whether the compiler accepts Standard Pascal or full VS Pascal
LANGUAGE(*ccc*)	—	LANGUAGE(UEN)	Specifies the language (*ccc*) in which messages, report headings, and other textual information is presented
LINECOUNT(*n*)	LC(*n*)	LINECOUNT(60)	Specifies the number of lines to appear on each page of the output listing
LIST | NOLIST	—	NOLIST	Controls the generation of the pseudo-assembler listing
MARGINS(*m*,*n*)	MAR(*m*,*n*)	MARGINS(1,72)	Sets the left and right margins of the input program
OPTIMIZE | NOOPTIMIZE	OPT | NOOPT	OPTIMIZE	Controls whether the compiler generates optimized code
PAGEWIDTH(*n*)	PW(*n*)	PAGEWIDTH(128)	Specifies the maximum number of characters that can appear on a single line of the output listing

Compile-Time Option	Abbreviated Name	IBM-Supplied Default	Description
PXREF | NOPXREF	—	PXREF	Specifies that the right margin of the output listing is to contain cross-reference entries
SEQUENCE(*m*,*n*) | NOSEQUENCE	SEQ(*m*,*n*) | NOSEQ	SEQUENCE(73,80)	Specifies which columns within the program being compiled are reserved for a sequence field
SOURCE | NOSOURCE	S|NOS	SOURCE	Controls the generation of the compiler source listing
STDFLAG(I|W|E|S)	—	STDFLAG(E) for LANGLVL(ANSI83)	Controls how most standard extensions are flagged (informational, warning, error, or severe error message) when LANGLVL(EXTENDED) is not in use
WRITE | NOWRITE	—	NOWRITE	Controls whether messages in a %WRITE compiler directive are written to the terminal during compilation
XREF(SHORT|LONG) | NOXREF	X | NOX	XREF(SHORT)	Controls the generation of the cross-reference portion of the source listing

Link-Edit Options

Link-Edit Option	Abbreviated Name	IBM-Supplied Default	Description
DEBUG | NODEBUG	—	DEBUG	Prepares the module for debugging with the interactive debugging tool
TRANLIB | NOTRANLIB	—	NOTRANLIB	Controls whether generated code will be fully link-edited before execution, or whether run-time library routines will be loaded dynamically during execution
XA | NOXA	—	NOXA	Controls whether the load module can use extended addressing on MVS/XA, MVS/ESA, and VM/XA

Run-Time Options

Run-Time Option	IBM-Supplied Default	Description
COUNT	—	Causes statement frequency information to be collected during program execution and written to file OUTPUT; (effective only if the program was compiled with the DEBUG option and link-edited with the debugging library)
DEBUG(PROMPT| NOPROMPT)	DEBUG(PROMPT)	Activates the interactive debugging tool, either with or without an initial command prompt
ERRCOUNT(*n*)	ERRCOUNT(20)	Specifies the number of nonfatal run-time errors that can occur before the program terminates
ERRFILE(*ddname*)	ERRFILE(terminal)	Specifies the file to which error diagnostics, debugging output, and counting statistics are written
HEAP([*initsize*,] *incrsize*)	HEAP(12,12)	Specifies, in kilobytes, the initial size of the heap (*initsize*) and how much the heap is to be extended on overflow (*incrsize*)
LANGUAGE(*ccc*)	LANGUAGE(UEN)	Specifies the language (*ccc*) in which messages, report headings, and other textual information is presented
MAINT	—	Includes system run-time routines in any routine traces
NOCHECK	—	Causes all checking errors to be ignored
NOSPIE	—	Suppresses the interception of program exceptions
SETMEM	—	Initializes a routine's local storage to a specific value on each entry
STACK(*n*)	STACK(12)	Specifies the number of kilobytes (*n*) the stack is to be extended on overflow

IBM®

Program Number
5668-767
5668-717

File Number
S370-40

SX26-3760-1

Printed in U.S.A.