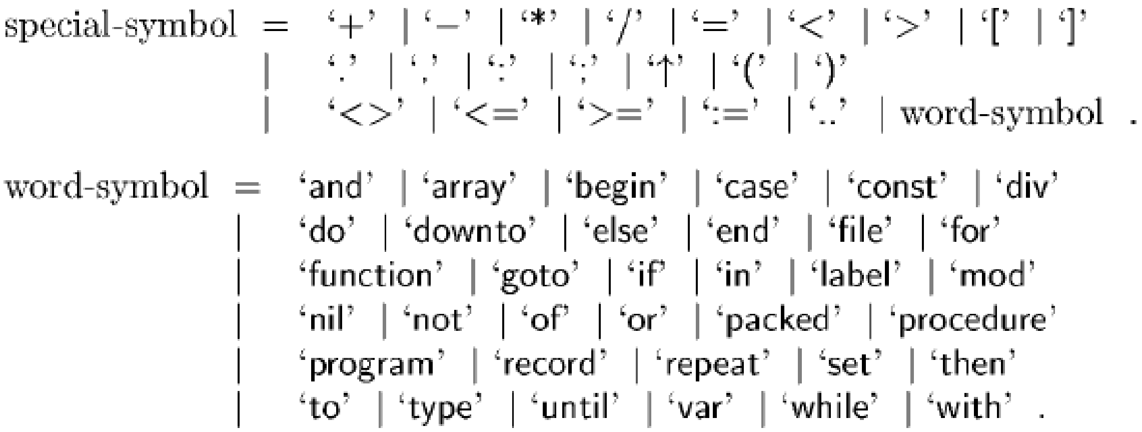
BorlandPascal 7.0

```
╔═[■]════════════════
║var real:integer;
║
║begin
║  real:=5;
║  write(real)
║end.
```



```
special-symbol  =  '+'  | '−'  | '*'  | '/'  | '='  | '<'  | '>'  | '['  | ']'
                |  '.'  | ','  | ':'  | ';'  | '↑'  | '('  | ')'
                |  '<>'  | '<='  | '>='  | ':='  | '..'  | word-symbol  .

word-symbol  =  'and'  | 'array'  | 'begin'  | 'case'  | 'const'  | 'div'
             |  'do'  | 'downto'  | 'else'  | 'end'  | 'file'  | 'for'
             |  'function'  | 'goto'  | 'if'  | 'in'  | 'label'  | 'mod'
             |  'nil'  | 'not'  | 'of'  | 'or'  | 'packed'  | 'procedure'
             |  'program'  | 'record'  | 'repeat'  | 'set'  | 'then'
             |  'to'  | 'type'  | 'until'  | 'var'  | 'while'  | 'with'  .
```

```
╔═[■]════════════════════
║type real=integer;
║var r:real;
║
║begin
║  r:=7.7;                    // 7
║  write(r)
║end.
```

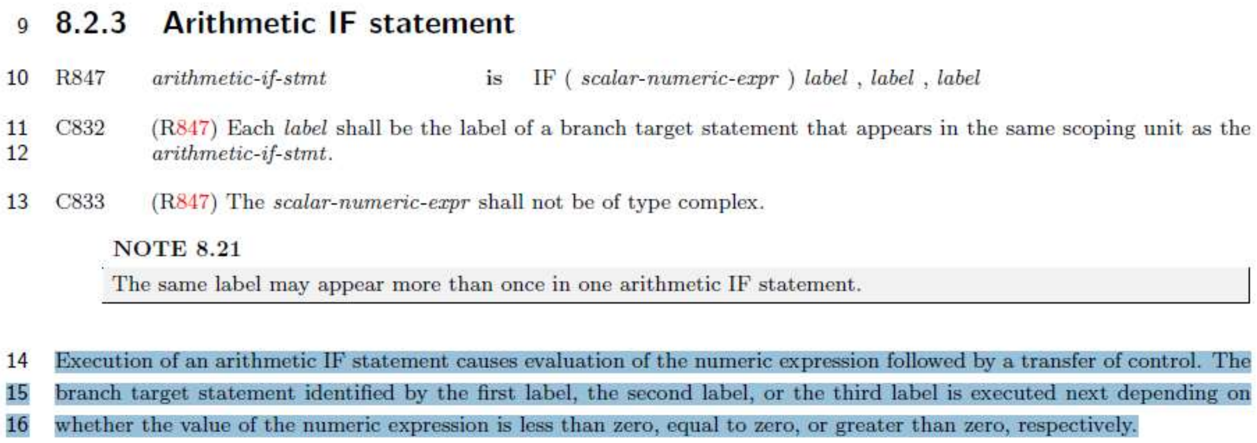
Very old FORTRAN code (using arithmetic if statement):

```
         DIMENSION IF(20)
         …

         J=1
         I=10-J
         IF(I) 3,4,5
3        A=B+C
         GO TO 10
4        A=B-C
         GO TO 10
5        A=B*C
10       IF(J)=A
```

Arithmetic IF Statement

This form of IF statement is obscure and its use is strongly discouraged.



8.2.3 Arithmetic IF statement

R847 *arithmetic-if-stmt* is IF (*scalar-numeric-expr*) *label* , *label* , *label*

C832 (R847) Each *label* shall be the label of a branch target statement that appears in the same scoping unit as the *arithmetic-if-stmt*.

C833 (R847) The *scalar-numeric-expr* shall not be of type complex.

NOTE 8.21

The same label may appear more than once in one arithmetic IF statement.

Execution of an arithmetic IF statement causes evaluation of the numeric expression followed by a transfer of control. The branch target statement identified by the first label, the second label, or the third label is executed next depending on whether the value of the numeric expression is less than zero, equal to zero, or greater than zero, respectively.

(source: https://fortran-lang.discourse.group)

http://en.cppreference.com/w/cpp/keyword C++ keywords

C++ keywords

This is a list of **reserved keywords** in C++. Since they are used by the language, these keywords are not available for re-definition or overloading.

alignas (since C++11)
alignof (since C++11)
and
and_eq
asm
atomic_cancel (TM TS)
atomic_commit (TM TS)
atomic_noexcept (TM TS)
auto(1)
bitand
bitor
bool
break
case
catch
char
char16_t (since C++11)
char32_t (since C++11)
class(1)
compl
concept (since C++20)
const
constexpr (since C++11)
const_cast
continue
decltype (since C++11)
default(1)
delete(1)
do
double
dynamic_cast
else
enum
explicit
export(1)
extern(1)
false
float
for
friend
goto
if
import (modules TS)
inline(1)
int
long
module (modules TS)
mutable(1)
namespace
new
noexcept (since C++11)
not
not_eq
nullptr (since C++11)
operator
or
or_eq
private
protected
public
register(2)
reinterpret_cast
requires (since C++20)
return
short
signed
sizeof(1)
static
static_assert (since C++11)
static_cast
struct(1)
switch
synchronized (TM TS)
template
this
thread_local (since C++11)
throw
true
try
typedef
typeid
typename
union
unsigned
using(1)
virtual
void
volatile
wchar_t
while
xor
xor_eq

- some changes in C++ newer versions.

...

The name `posix` is **reserved** for a future top-level namespace. The behavior is undefined if a program declares or defines anything in that namespace.	(since C++11)

Java

http://java.sun.com/docs/books/jls/second_edition/html/lexical.doc.html

Keywords

sequences **reserved** for use as keywords and cannot be used as identifiers

- abstract
- boolean
- break
- byte
- case
- catch
- char
- class
- **const**
- continue
- default
- do
- double
- else
- extends
- final
- finally
- float
- for
- **goto**
- if
- implements
- import
- instanceof
- int
- interface
- long
- native
- new
- package
- private
- protected
- public
- return
- short
- static
- strictfp
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- try
- void
- volatile
- while

Reserved words

`true, false, null`

While `true` and `false` might appear to be keywords, they are technically Boolean literals. Similarly, while `null` might appear to be a keyword, it is technically the null literal.

The keywords `const` and `goto` are reserved, even though they are not currently used. This may allow a Java compiler to produce better error messages if these C++ keywords incorrectly appear in programs.