Compatible Fortran

A. Colin Day

5886

Compatible Fortran

A. Colin Day

Computer Consultant, University College London

Cambridge University Press

Cambridge

London New York Melbourne

Preface

One of the chief advantages of a standardised language is its compatibility, i.e. its independence of the computer on which it is run. After some experience in writing portable Fortran programs, it soon became apparent to me that conformity to the Fortran Standard is not enough. The subset which constitutes Compatible Fortran is sufficiently different from American National Standards Institute (ANSI) Standard Fortran to require a separate description.

The need for such a description of Compatible Fortran was brought home to me by working in a University which possesses several quite different computers, and where programs may be developed on one of these and sent to another for production running. The need became more urgent when the local computer was to be replaced by equipment made by a different manufacturer.

This book was written with the hope that experience gained in trying to run programs on more than one machine might be useful to others. That my experience is limited, and therefore that my advice suffers from many shortcomings, is readily admitted. However, my experience is enough to show me that limited advice in this area is nevertheless much to be preferred to no advice at all.

I am greatly indebted to a number of people for the help they have afforded me (by their published writings or by their comments) in the production of this book. First and foremost among these is Brian Shearing, who was to have shared in the writing of the book. When this unfortunately proved impossible, he graciously made over his material to me, for which I am deeply grateful. Mention must also be made here of David Muxworthy, and of Andrew D. Hall Jr, whose help and advice have been greatly appreciated.

A. COLIN DAY

October 1977

Contents

1 Introduction

There is no computer language as potentially compatible as Fortran. Despite this, few programmers are able to make use of this potential. Compiler writers constantly produce compilers with new 'goodies' and manufacturers seldom point out that using these features can prevent a program from running on a competitor's machine.

This book describes a dialect of Fortran ('Compatible Fortran', or 'CF' for short) which is acceptable to most compilers, so that programs written in it will run on many different machines.

Lest some basic terms be misunderstood, it is necessary to clarify them at the outset. A language which is acceptable to more than one type of computer is said to be *compatible*. A program written in such a compatible language is described as *portable* (or *mobile*).

It is assumed that the reader is already fully conversant with Fortran on at least one machine. It is not assumed that he is an expert on Fortran standardisation or Fortran dialects.

Advantages of portability

Often programs are written using the facilities available on one machine with no thought of transferring them. However, news of good programs travels fast. You may receive a request for one of them from someone who uses a quite different machine. Even if you are careful to keep the virtues of your program secret, the day may well come when you have to move from your present machine (because it is closing down, being replaced etc.), and the new machine may not be the same as the old one.

In these days of computer networks, you may find it advantageous to debug a program on one computer, then send it down the line to a larger mainframe for production running. Obviously, this is only possible if it is portable.

The main advantage of portability is that it enables you (or others — or both) to bypass the problems of adapting a program in order to transfer it to another machine. If it seems to you to be a nuisance learning about a new Fortran dialect and conforming to its rules, be well aware of the immense nuisance that adaptation can be. The program must be adapted for every new computer on which it is to be run. If incompatible features have been deeply embedded in the code, it may be necessary to restructure the basic logic. Countless people have discovered that without portable programs there are times when you have to run very fast indeed, like Alice, in order to stay where you are.

Fortran standards and Fortran 77

Standards for the Fortran language have been drawn up 'for the purpose of promoting a high degree of interchangeability of . . . programs for use on a variety of automatic data processing systems' (ANSI 1966*a*, p.7). The main reason why Fortran has such potential compatibility is the existence of complete and well-defined standards. Much of this book will be taken up in describing what the standards have laid down, since unfortunately many Fortran programmers appear to be unaware of them.

For a history of Fortran standardisation, see Muxworthy (1972). The American National Standards Institute (ANSI) are responsible for the best known standard for Fortran, American Standard Fortran, defined in their document X3.9–1966 (ANSI 1966*a*). This is the language commonly known as ANSI Fortran, or (in loose parlance) Fortran IV. Unfortunately, ANSI have had trouble in standardising their own name. They were at one time the American Standards Association (ASA), and then the United States of America Standards Institute (USASI), so one finds references to 'ASA Fortran' or to 'USASI Fortran', meaning the same as 'ANSI Fortran', i.e. the language defined in X3.9–1966.

ANSI also published another standard alongside X3.9. This is called American Standard Basic Fortran, and is defined in ANSI document X3.10–1966 (ANSI 1966*b*). The language is a subset of X3.9, and corresponds in power to the old Fortran II.

ANSI produced two articles clarifying the Fortran standards at certain points which had been open to various interpretations (ANSI 1969, 1971).

The European Computer Manufacturers Association (ECMA) have produced an ECMA standard for Fortran which is intermediate between the two American standards (ECMA 1965). The International Organisation for Standardisation (ISO) produced a recommendation which incorporated these three levels of Fortran (ISO 1972).

Technically the two ANSI standards lapsed in 1971, since they were not revised or reaffirmed within five years. An ANSI subcommittee, X3J3, has been at work on a new ANSI standard, called Fortran 77, which has now been adopted. Fortran 77 has two levels. The higher of these retains almost all of X3.9 as a subset, but has considerable extensions. (See the next section.)

As far as compatibility is concerned, the standards are a great help, but are not enough. Some parts of the standards remain open to interpretation, and many compilers, while purporting to accept ANSI Fortran, are in fact more restrictive. Later extensions, whether allowed by a compiler or introduced by a new standard such as Fortran 77, must be avoided if compatibility is to be maintained.

There are also areas where compatibility is possible even though the standards (in their most literal interpretation) would deny it. One such area is character manipulation (chapter 14). In such a case Compatible Fortran is less restrictive than the standards.

The standard which comes nearest to Compatible Fortran is (and will continue to be for some time to come) American National Standard Fortran. In the remainder of this book, whenever 'Standard Fortran' or 'the Standard' is mentioned, it will refer to this standard. As far as possible, the terms used in this book will be those used in the Standard. Those places where Compatible Fortran differs from the Standard will be pointed out.

Many problems in the area of compatibility have their origin in introductory textbooks and courses for Fortran where either the Standard was not known to the teacher, or the temptation to introduce 'useful' extensions could not be resisted. The teaching of Standard Fortran in programming courses is strongly recommended. (See, for instance, Day (1972*a*) and the videotape course which those notes accompany.)

Because Compatible Fortran lies so closely alongside Standard Fortran, much of this book will be occupied with teaching what constitutes the Standard. For this reason, it is much more convenient to adopt the nomenclature which the Standard uses, even though this is often at variance with commonly used terms. It is not possible to alert the reader whenever such terms occur. Nor is it possible to paraphrase them on each occurrence. However, the index does give explanations of these terms, and should be consulted frequently until the reader acquires familiarity with the Standard's terminology.

Although this book seeks to clarify the differences between common (incompatible) Fortran usage, the Fortran Standard and Compatible Fortran, it does not repeat all of the elementary rules of Fortran, which every introductory textbook describes and which every compiler checks.

Fortran 77

Some may consider that the recent emergence of a new standard for Fortran largely undercuts the need for this book. A little thought will show that this is not so. Now that the new standard has been approved, some compilers will be written for it, but not every compiler will be changed. The compatible subset will still be the lowest common denominator, and since Fortran 77 will be a larger hoop, programs which get through the smaller hoops should get through that one also.

There are, however, the problem areas where Fortran 77 does not retain full compatibility with X3.9. These have been noted as areas where Compatible Fortran is more restrictive than the 1966 Standard.

The environment

It must be emphasised that the portability of a program does not consist solely in conforming to certain grammatical rules. A program may make certain demands on its environment which can only be satisfied by very few machines, and this may severely limit its portability.

A program requiring a large number of peripheral devices, or unusual and specialised equipment, obviously may not be very portable. However, a program which needs real values to be held to great precision is also making demands from its environment which not many machines can fulfil. A very large program may also be non-portable for the same reason, however well it is written.

Fortran dialects

Different compilers accept slightly different kinds of Fortran, and so different 'dialects' have sprung up. A dialect causes problems in two ways. It may be more permissive than the Standard, and so may encourage programmers to use features which are not available with more than one compiler. Or, on the other hand, it may be more restrictive than the Standard, and so the compiler concerned will not accept programs which ought to be portable. Therefore Compatible Fortran will need to be different from the language defined in the Standard.

Mention should be made here of another attempt to provide a compatible Fortran dialect. This is PFORT, described in Ryder (1974). In general, CF is more restrictive than PFORT, and therefore compatible over a wider range of compilers.

It is rarely possible to give details here of specific compilers which cause trouble in particular areas. There are many reasons for this. Compilers change, and to try to name the culprits is to try to hit a host of moving targets. It is not sufficient to pin down the problems by referring to a particular type of computer, because the compilers available on that computer exhibit differences among themselves. Not the least of the reasons for not naming particular compilers is, without doubt, my own ignorance. It will certainly be discovered that, because of the shortcomings of my knowledge, Compatible Fortran is not as compatible as it could be. Nevertheless, such an attempt, though incomplete, is better than no attempt at all.

Some comparisons have been made between various Fortran compilers. Those interested should consult Stuart (1969), and Muxworthy and Shearing (1970).

Although it has not been possible in general to name the compilers, I have attempted to give reasons for restrictions and words of caution. These have usually been phrased in terms of machine architecture, or in terms of the habits of certain unnamed compilers.

Some may object that CF is itself just another dialect, helping to swell the number of non-standard varieties of Fortran. One might as well object that a master-key is just another key. So it is, but it may drastically reduce the number of other keys which one has to carry around.

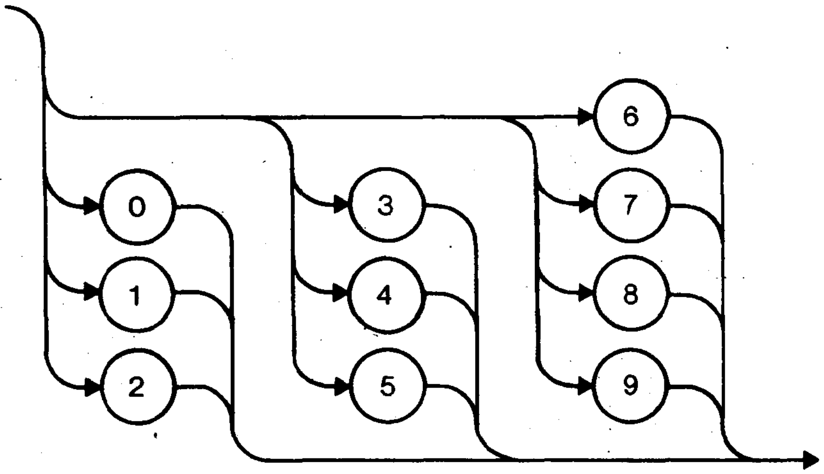
2 Building blocks

This chapter deals with some of the basic components of Compatible Fortran, namely the character set, symbolic names, constants and array elements (including subscript expressions). Other expressions are dealt with in chapter 11.

Character set

The character set consists of digits, letters and special characters. At this point it is necessary to introduce the special flowcharts which are used here to indicate permitted forms. The first flowchart shows that a digit is any of the ten characters 0, 1, 2, 3, 4, 5, 6, 7, 8 or 9.

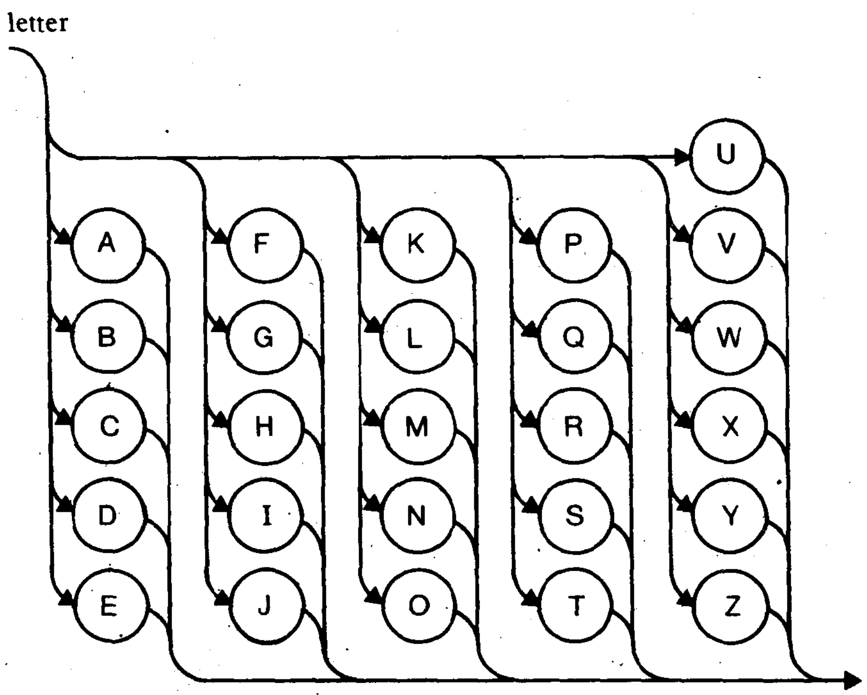
digit



This kind of flowchart is similar to a maze. You start at the top left-hand corner, and come out at the bottom right. Boxes are like turnstiles. You may only pass through them in one direction (indicated by the arrow), and then only on 'payment' of the required item. Rounded boxes require the literal characters shown on them. Rectangular boxes require an entity which has been defined elsewhere. The paths should be treated like railway lines. Shunting backwards past a sharp corner is not permitted. Sometimes a chart must be modified by verbal restrictions placed underneath it. I believe that M. D. McIlroy of Bell Laboratories was the first to use such charts for Fortran, in McIlroy (1974).

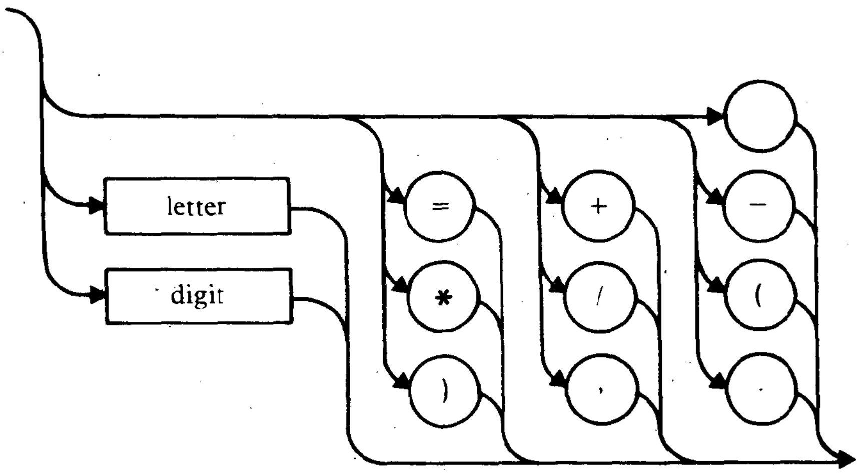
The name above the entrance shows the entity which the flowchart defines. You can get through the flowchart of digits above only if you have one of the ten digits to give in 'payment'.

The letters may be defined in a similar way.



Now that we have defined digits and letters, the full character set may be defined using these, together with the ten special characters.

character



The box containing no character is to be traversed on payment of a blank.

Standard Fortran also includes the currency symbol ($), but the Standard does not give any indication of how this symbol is to be used in the language. Some compilers interpret it as an extra alphabetic character. Others use it to separate two statements on the same line. The character itself is hard to pin down in terms of a punched-card code. For these reasons it should be avoided.

The Standard allows non-Fortran characters in three places:

(1) In comment lines. These will cause problems on transfer to other card codes. Use only characters from the Compatible Fortran character set in comments.

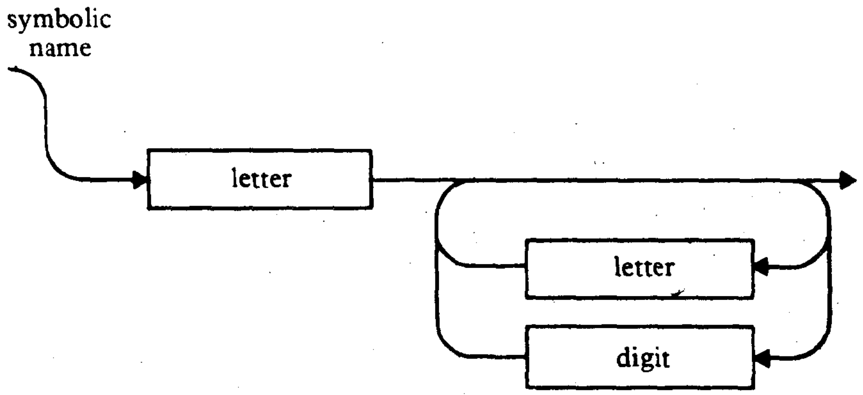
(2) In a Hollerith datum (e.g. in a DATA statement or a FORMAT statement). Once again, problems will be caused on transfer, and characters other than those in the CF character set should be avoided here also.

(3) On data cards. If the data is to be transferred along with the

program, then non-Fortran characters should be avoided. If the data cards are solely used for running the program on one machine and no other, no harm results from having non-Fortran characters in the data.

Symbolic names

A *symbolic name* is the name of a variable, array, common block, statement function, function, subroutine or external subprogram. Such a name is constructed as shown here.



Restriction: No more than six characters in the name

Examples: A I X2 Y58902 K2SO4

Note that symbolic names may not be longer than six characters, even though some compilers allow eight, or even 31 characters.

The Standard allows a name to be used for more than one purpose within the same program unit, within certain limits. For instance, a variable may have the same name as a COMMON block. Also, according to the Standard, variables may have the same name as intrinsic functions which are not referred to in that program unit. Some compilers do not permit this, however.

The Standard does not restrict one from using symbolic names which are identical to Fortran keywords, e.g. END or REWIND. Some compilers cannot cope with this. In particular, compilers have been known to treat all statements beginning with the characters FORMAT(as a FORMAT statement, so forbidding a sequence of statements like

```
DIMENSION FORMAT (10)
FORMAT (1)=0.0
```

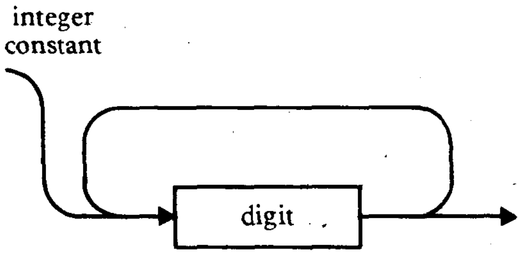
Safer rules for symbolic names are as follows:

(1) A name for a COMMON block should not be used for any other purpose in the same program unit.
(2) The name of any intrinsic or basic external function should not be used for any other purpose in any program unit.
(3) A symbolic name should not be the same as any Fortran keyword.

Constants

This section will be concerned merely with the form of constants. For a discussion of the limits on the size of their values, see chapter 13.

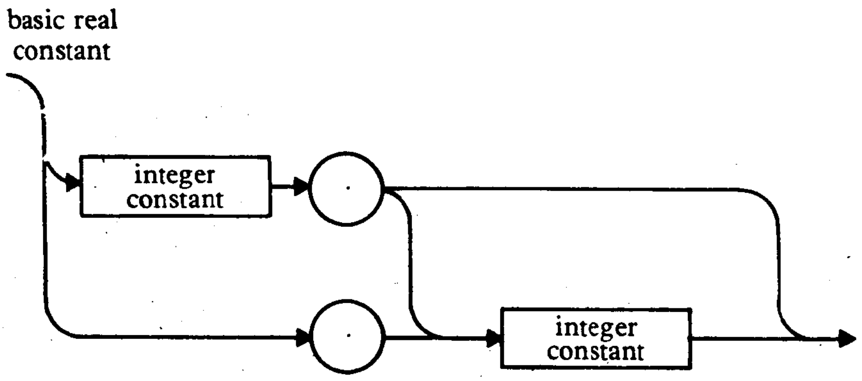
Integer constants are defined as follows.



Examples: 1 0 17 68458

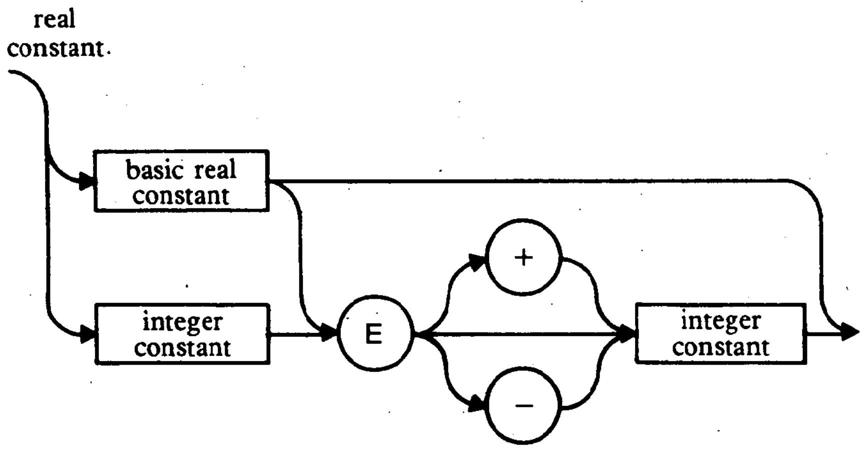
Note that the term 'integer constant' (as used by the Standard) refers to an unsigned constant. This is also true for real and double precision constants. Where a sign is permitted, this will be specifically shown in later flowcharts.

Real and double precision constants are defined in terms of an intermediate construct which the Standard calls a *basic real constant*.



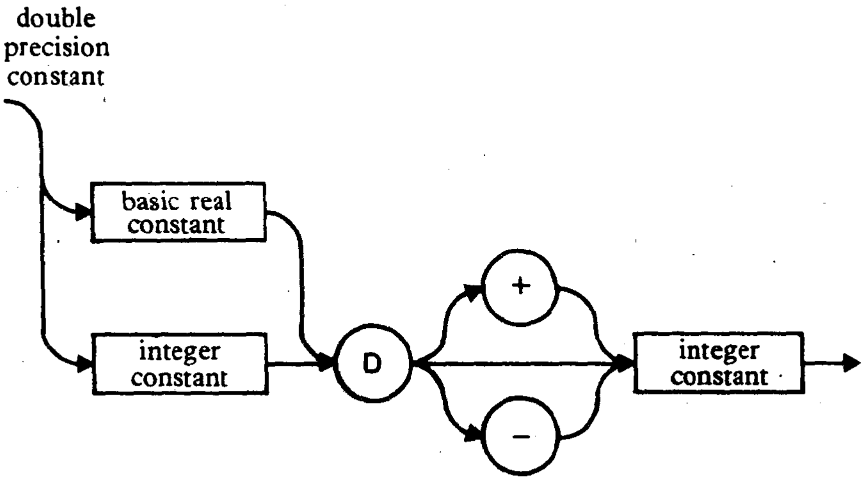
Examples: .002487 5. .0 7.2 865.489

Now we can define a real constant in terms of this.



Examples: 5.E−32 2E5 0. 7.2 7.2E+8

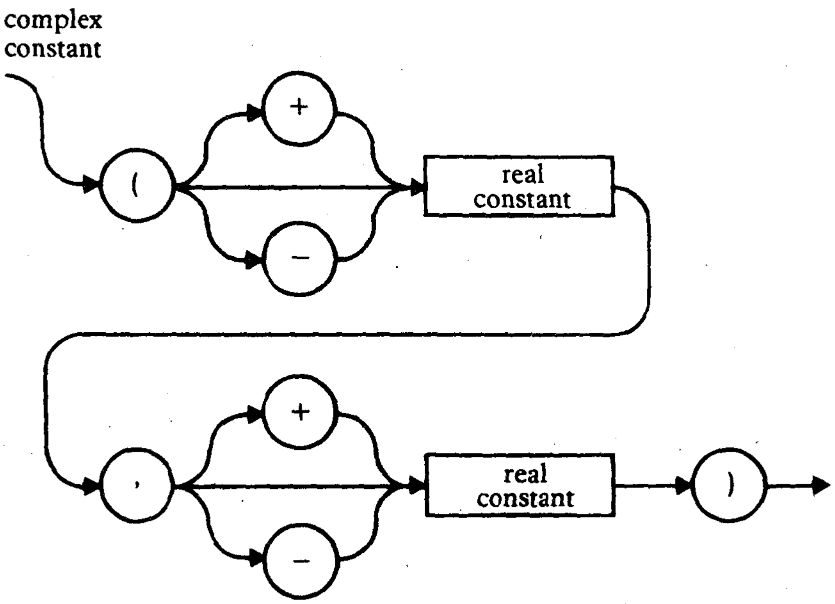
A double precision constant has a similar definition.



Examples: 5.D−32   2D5   7.2D+8   0D0

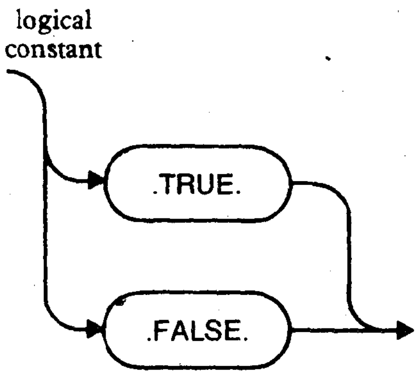
Note that a double precision constant must have a D exponent. Some compilers also consider a basic real constant with more than a certain number of digits to be double precision, but this is neither compatible nor Standard.

Complex constants are defined in terms of real constants.



Examples: (0.,0.)   (−5.6E3,.25)   (−1.0,−2.0E−1)

Double precision complex constants are not Standard.
There are only two logical constants.


logical
constant
.TRUE.
.FALSE.