Real-Time languages

The requirements of real time software place heavy demands on programming languages. It should be now be obvious that it is essential that real-time software is reliable, the failure of a real-time system can be expensive both in terms of lost production, or in some cases, in the loss of human life (on aircraft control system).

User Requirements

The user requirement is divided into six general areas:

(1) Security (2) Readability (3) Flexibility

(4) Simplicity (5) Portability (6) Efficiency

(1) Security

Security can be considered to be a measure of extent to which a language to detect errors automatically either at compile time or through the run time.

Economically it is important to detect errors at the compilation stage than at run-time since the earlier the error is detected the less it costs to connect.

In real-time system development the compilation is often perform on a computer than the one used in actual system, whereas run-time testing has done on the actual hardware, and in the later stages, on the hardware connect plant.

(2) Readability

The readability of a program is a measure of the ease with which its operation can be understood without resort to supplementary documentation such as flowcharts or natural language descriptions.

The emphasis is on ease of reading because a particular segment of code will only be written once but will be read many times.

The benefits of a good readability are:

(1) Reduction in documentation costs.

(2) Easy error detection.

(3) Easy maintenance.

(3) Flexibility

For a language to be described as a general purpose language there is a requirement that the programmer should be able to express all the operations required in a program without the need to use assembly coding. The flexibility of language is a measure of this facility.

It is particularly important in real-time system, in that frequently non-standard I/O device will have to be controlled. The achievement of high flexibility can conflict with achieving high security.

(4) Simplicity

In language design, as in other areas of design, the simple is to be preferred to the complex. Simplicity contribute to security. It reduces the cost of training. It reduces the probability of programming errors arising from misinterpretation of the language features, it reduce compiler size and leads to more efficient object code.

(5) Portability

The achievement of portability, while very desirable as a means of speeding up developments, reducing costs, and increasing security, is difficult to achieve in practice. Surface portability has improved with the standardization agreements on many languages, i.e. it is now often possible to transfer a program from one computer to another and find that it will compile and run on the computer to which it has been transferred.

There are however, still problems when the word lengths of the two machines differ; there may also be problems with precision with which numbers are represented even on computers with the same word-length.

(6) Efficiency

In the early computer control systems great emphasis was placed on efficiency of the coding _ both in term of the size of the object code and the speed of operation _ as computers were both expensive and very slow (by today's standards) .

As a consequence programming was carried out using assembly languages and frequently tricks were used to keep the code small and fast. The desire for the generation of efficient object code was carried over into the designs of the early real time languages and in these languages the emphasis was on efficiency rather than security and readability.

Language Requirements and features

The major features which must be considered are listed below:

1. Declarations
2. Types
3. Initialization
4. Constants
5. Control structures
6. Scope and visibilities
7. Modularity
8. Exception handling
9. Independent/Separate compilation
10. Multi-tasking
11. Low level constructs

1. Declarations

The purpose of declaring an object used in a program is to provide the compiler with information on the storage requirements and to inform the system explicitly of the names being used.

Languages such as Pascal require all objects to be specifically declared and for a type to be associated with the object at declaration.

The provision of type information allows the compiler to check that the object is used only in operations associated with that type. If, for example, an object is declared as being of type real and then is used as an operand in logical operation, the compiler should detect the type incompatibility and flag the statement as being incorrect.

2. type

As we have seen above, the allocation of types is closely associated with the declaration of objects. The allocation of a type defines the set of values that can be taken by an object of that type and the set of operations that can be performed on the object.

The richness of types supported by a language and the degree of rigor with which type compatibility is enforced by the language are important influences on the security of programs written in the language.

Languages which rigorously enforce type compatibility are said to be strongly typed, languages which do not enforce type compatibility are said to be weakly typed.

3. Initialization

It is useful if, at the time of declaration of variables, it can be given initial value. This is not, of course, strictly necessary as a value can always be assigned to a variable.

In terms of the security of a language it is important that the compiler checks that a variable is not used before it has had a value assigned to it. It is bad practice to rely on the compiler to initialize variables to some zero or null value.

The security of languages such as Pascal is enhanced by the compiler checking that all variables have been given an initial value.

4. Constants

Some of the objects referenced in a program will have constant values either because they are physical or mathematical entities such as the speed of light or pi (π) or because they are a parameter which is fixed for that particular implementation of the program.

It is always possible to provide constants by means of initializing a variable to the appropriate quantity.

5. control structures

There has been extensive argument over the past few years about the use of both conditional and unconditional GOTO statements in high-level languages. It is argued that the use of GOTOs makes a program difficult to read and it has been shown that any program can be expressed without the use of GOTOs as long as the language supports the WHILE statement, the IF -- THEN -- ELSE conditional and BOOLEAN variables.

6. scope and visibility

The scope of a variable is defined as the region of a program in which the variable potentially accessible or modifiable. The regions in which it may actually be accessed or modified are the regions in which it is said to be visible.

Thus in a FORTRAN program the scope of a variable declared in the main program extends over the whole program, but such a variable, unless named in a COMMON statement, will not visible in any sub-program. Scope and visibility are closely related to where in a program a variable is declared.

Choice of programming language

The development and maintenance costs can be considerably reduced if a language is well supported by a range of development tools. Development tools would include all, or some, of the following:

1. Editor
2. Library manager
3. Linker/ loader
4. Debugger
5. Version control
6. Database manager
7. Pretty printer
8. Cross-reference generator