Safe and Secure Software

An Invitation to **Ada 2005**

Conclusion

Courtesy of AdaCore
The GNAT Pro Company

John Barnes

It is hoped that this booklet will have proved interesting. It has covered a number of aspects of writing reliable software and hopefully has shown that Ada is a good language and source of inspiration to use for programs that matter. We conclude with some background notes on the development of languages.

The balance between hardware and software is interesting. Hardware has evolved in an amazing way in the last half century. The hardware of today bears no resemblance whatever to the hardware of 1960. By contrast, software has progressed but little. The languages of today are in many ways little different to those of 1960. I suspect that the ultimate problem is that we know little about software although we probably think we know rather a lot. Moreover, society has made huge investments in badly written software and finds it hard to move forward at all. But hardware changes so rapidly that it inevitably gets discarded. And of course it is very easy for anyone to learn to write a bit of software but massive know-how is required to build any hardware.

Mainstream languages have two main origins, Algol 60 and CPL. These are the ancestors of the languages mentioned most in this booklet. Another group of languages, Fortran, COBOL and PL/I, live on but seem to be somewhat isolated.

Algol 60 was perhaps the most important step forward ever made. (There was a lesser known precursor called Algol 58 from which the US military language Jovial was derived but that is a minor detail.) Algol gave the feeling that writing software was more than just coding.

Algol made two big steps. It recognized that assignment was not equality by using := for assignment. It also introduced English words for control purposes and thereby eliminated most of the gotos, jumps and labels that made early Fortran and autocode programs so hard to understand. This second point is worth looking at in some detail.

Consider first the following two statements in Algol 60

```
if X > 0 then
  Action( ... );
Otherstuff( ... );
```

The effect is that if X is indeed greater than zero then the subroutine Action is called. Whether Action is called or not we then always go on to call Otherstuff. The interesting thing is that the conditional only governs the first statement following **then**. If we need to govern several statements such as call subroutines This and That then we have to combine the two statements into a single compound statement thus

```
if X > 0 then
begin
  This( ... );
  That( ... );
```

```
end;
Otherstuff( ... );
```

There are two dangers here. One is simply that we might forget to insert **begin** and **end**. It would still compile of course but That would always get called whatever the value of X. But a bigger hazard is the danger of stray semicolons. Algol 60 was perhaps the first language to use semicolons to terminate or separate statements. Now consider what happens if a programmer inadvertently adds a semicolon immediately after **then**. We get

```
if X > 0 then ;
begin
  This( ... );
  That( ... );
end;
Otherstuff( ... );
```

Unfortunately, in Algol 60 the semicolon is deemed to be separating a null statement from the compound statement (a null statement does nothing – it is invisible too!) And so the conditional does nothing and the subroutines This and That are always called. There were other related problems in Algol 60 concerning the syntax of loops.

The designers of Algol 68 recognized this problem and introduced a bracketed form thus

```
if X > 0 then
  This( ... );
  That( ... );
fi;
Otherstuff( ... );
```

Other similar structures were used for loops with **do** being matched by **od** and **case** being matched by **esac**. This structure completely solves the problem. It is now crystal clear that the conditional governs the two statements. Moreover, adding a spurious semicolon after **then** is a syntax error and so is instantly detected by the compiler. Of course many thought that the reversed words **fi**, **od** and **esac** indicated that the language was bizarre and not to be taken seriously.

Whatever the reason, the designers of Pascal ignored this sensible approach and continued to use the flawed structure of Algol 60. Eventually however they did realize their error when it came to Modula 2 but this was long after Ada.

Ada was probably the first successful language to use the bracketed structure but it does sensibly avoid the peculiar backward words. Thus in Ada we write

```
if X > 0 then
  This( ... );
  That( ... );
```

```
end if;
Otherstuff( ... );
```

Many other languages have taken this safe route including even the macro language in the elegant Microsoft Word for DOS and Visual Basic which is the corresponding macro language for Word for Windows.

The other important background language was CPL. It was devised in about 1962 as the language to be used by two powerful new computers at Cambridge and London universities.

CPL (like Algol 60) used := for assignment and = for equality. Here is a small fragment of CPL

```
§ let t, s, n = 1, 0, 1
  let x be real
  Read[x]
    t, s, n := tx/n, s + t, n + 1
    repeat until t << 1
  Write[s] §|
```

An interesting feature of CPL is that it used = rather than := when setting initial values on the grounds that no change was involved. CPL had many novel features such as parallel assignments and list processing. However, CPL was never implemented but remained an academic design.

CPL used essentially the same structure as Algol 60 for grouping statements. Thus we would have written

```
if X > 0 then do
  § This( ... )
    That( ... ) §|
Otherstuff( ... )
```

Note that the items grouped together are surrounded by the strange brackets § and §| (actually the closing bracket was the section sign with the vertical bar through it but this word processor does not allow me to do that so I have put them side by side).

Although CPL was never implemented, the simple language BCPL (Basic CPL) was a simple successor devised at Cambridge. The major difference was that whereas CPL was a strongly typed language, BCPL really had no types at all and arrays were just treated as arithmetic on addresses. BCPL is the origin of the buffer overflow problem which plagues the world today.

From BCPL came B and then C, C++ and so on. BCPL used := for assignment but somewhere along the way someone missed the point and C ended up with = for assignment. Having hijacked = for assignment C uses a

double equals (==) to mean equality and this gives rise to a number of problems as we saw in the chapter on Safe Syntax.

C inherited the same compound statement style from CPL but replaced the strange brackets by the braces { and }and thus in C we write

```
if (x > 0)
{
   this( ... );
   that( ... );
};
otherstuff( ... );
```

There is little of the original CPL left in C. In fact the only thing really left is the brackets.

And finally, we conclude by noting that the use of the equals sign for assignment is an example of the use of puns so hated by the late Christopher Strachey. Strachey was one of the designers of CPL. At a NATO lecture many years ago he said *"The way in which people are taught to program is abominable. They are over and over again taught to make puns; to do shifts when they mean multiplying; to confuse bit patterns and numbers and generally to say one thing when they mean something quite different. I think we will not make it possible to have a subject of software engineering until we can have some proper professional standards about how to write programs; and this has to be done by teaching people right at the beginning how to write programs properly. I'm sure that one of the first things to do about this is to say what you mean, and not to say something quite different."*.

That about sums it up. We need to learn to say what we mean. Ada enables us to say what we mean clearly and that ultimately is its strength.

North American Headquarters
104 Fifth Avenue, 15th floor
New York, NY 10011-6901, USA
tel +1 212 620 7300
fax +1 212 807 0162
sales@adacore.com
www.adacore.com

European Headquarters
46 rue d'Amsterdam
75009 Paris, France
tel +33 1 49 70 67 16
fax +33 1 49 70 05 52
sales@adacore.com
www.adacore.com