COBOL compiler written in COBOL (too old to reply)

DSlash 18 years ago

Way back in 1974 I was the systems programmer in an all COBOL DP department, using an ICL 1902A mainframe, with punch card input and tape and disk storage.
That was a small slow machine, and to compile a COBOL program could take up to half an hour.

As usual there were many changes and file fixups to do, and these could be done only by writing a COBOL program especially for that purpose.
Even for a small program it could take half a day to code, punch, list, check, compile, fix errors, recompile, and test.
COBOL being a somewhat verbose language, the source was punched into cards by expert data-entry operators who were however unfamiliar with COBOL format! and often mispunched zeroes and Ohs, and didn't think that missed full-stops were all that important.

One day in 1974 I wondered if I could have a table in a COBOL program which had a list of various operations to be performed, such as READ, MOVE, WRITE etc, and just drop into the top of the table with a GO TO DEPENDING ON instruction.
I thought the resulting program would be extremely inefficient, since it would be an interpreter written in COBOL, but I decided to see if it worked.

My chief programmer (ChiefP) allowed me to secretly work on the project in my own time, since his boss, the development manager (DevM), liked to think up all the ideas, and would never have allowed this one to proceed.

I quickly coded a compiler which read cards with the desired instructions and converted them to a code starting from 1 upwards, and loaded them into the table, and dropped into a loop that executed the GO TO DEPENDING ON for each code in the table.
To my surprise it worked like a charm, and the speed no slower than any other program.
The compile time was only a few seconds.
Of course I should have realized that the execution time would be insignificant compared with file reads and writes, even though my MOVE was done by a loop one character at a time.
This initial version had the basic COBOL verbs, including nested IF and PERFORM. I didn't include GO TO at that stage but was eventually asked to include it.

Soon the DevM needed an urgent file fix, and the ChiefP suggested that I use my new compiler for the job. Soon afterwards the job was done, and we showed the DevM a printout that proved it had worked. We were grinning like hyenas, and he knew something was up since we'd been impossibly quick.
When I explained how I'd done it he was somewhat angry but he could see the potential of it.
After stewing for a while he allowed the development to continue, and allocated the next available utility name of UT06 to my compiler/interpreter.

I ended up with a language that looked very much like a COBOL procedure division, with most of the COBOL verbs allowable.
The compile time was a few seconds, and the object could be stored together with the interpreter as a permanent program.
There was also a decompiler built in so the original source could be examined if needed.

Thousands of programs were written in UT06.
Eventually the DP department installed an IBM mainframe, and it was an easy matter to recompile UT06 for IBM instead of ICL.
Those thousands of UT06 programs continued to work with no changes needed.

I still have the source of UT06. In theory it could be completed so that it could compile itself indefinitely.
Of course there is now no point in doing that. (Who uses COBOL these days? Don't answer that!)
However I have since written compilers that do compile themselves, and there are some advantages in that (but those languages are unlike COBOL).

OK then, any questions? Or shall I go back to my rest home :)

Arnold Trembley 18 years ago

> (snip)
> I still have the source of UT06. In theory it could be completed so that
> it could compile itself indefinitely.
> Of course there is now no point in doing that. (Who uses COBOL these
> days? Don't answer that!)
> However I have since written compilers that do compile themselves, and
> there are some advantages in that (but those languages are unlike COBOL).
> OK then, any questions? Or shall I go back to my rest home :)

Is UT06 available for download? How big is the source file?

--
http://arnold.trembley.home.att.net/

DSlash 18 years ago

> *Post by Arnold Trembley*
> > (snip)
> > I still have the source of UT06.
>
> Is UT06 available for download? How big is the source file?

I'd have to type it in off the printout that I have, which I will do if anyone is really interested. First I'll have to find the printout, which is stored among thousands of other things I have. I saw it a few weeks ago so there is hope.

But first, it's based on pre-1974 COBOL, and as it stands, there are three predefined areas, Record, Print and Work areas.
Fields within an area are defined as they are used, e.g. you can say
IF R12 = "TRANS" MOVE "LONDON" to P20.
-- which works with a 5 character field in position 12 of the Record area, and a 6 char field in position 20 of the Print line.

You could call them R_Code_12 and P_City_20 if you wanted to, since only the first character and the numeric part define the field.

With UT06 I never implemented symbolic names. I did that later with other languages, which I should probably discuss elsewhere.

Arnold Trembley 18 years ago

> *Post by DSlash*
> > *Post by Arnold Trembley*
> > > (snip)

> > > I still have the source of UT06.
> >
> > Is UT06 available for download? How big is the source file?
>
> I'd have to type it in off the printout that I have, which I will do if
> anyone is really interested. First I'll have to find the printout, which
> is stored among thousands of other things I have. I saw it a few weeks
> ago so there is hope.

Well, I don't want to give you a huge typing assignment, but I am curious as to how UT06 worked.

> *Post by DSlash*
> But first, it's based on pre-1974 COBOL, and as it stands, there are
> three predefined areas, Record, Print and Work areas.
> Fields within an area are defined as they are used, e.g. you can say
> IF R12 = "TRANS" MOVE "LONDON" to P20.
> -- which works with a 5 character field in position 12 of the Record
> area, and a 6 char field in position 20 of the Print line.
> You could call them R_Code_12 and P_City_20 if you wanted to, since only
> the first character and the numeric part define the field.

So, did UT06 take statements like 'IF R12 = "TRANS" MOVE "LONDON" to P20.' and compile/translate them into some code of tokenized code that was then processed by a run time executor? Perhaps I am assuming too much, but that sounds similar to IBM Series/1 EDL (Event Driven Language), used on an extinct 16-bit minicomputer. Technically, it was an interpreter, but in practice the performance was very close to native object code.

Rather than typing in the original UT06 source code (if that's too much of a chore), perhaps you could just describe the general approach in a little more detail. For example, how did UT06 handle arithmetic? Besides the IF and MOVE statements from the previous example, what other kinds of statements did UT06 handle? READ? WRITE? GOTO? Did UT06 do both compilation and execution?

> *Post by DSlash*
> With UT06 I never implemented symbolic names. I did that later with other
> languages, which I should probably discuss elsewhere.

Please, tell me more!

--
http://arnold.trembley.home.att.net/

DSlash 18 years ago

On Wed, 01 Oct 2003 04:46:23 GMT, Arnold Trembley

> *Post by Arnold Trembley*
> > *Post by DSlash*
> > I'd have to type it in off the printout that I have, which I
> > will do if anyone is really interested.
> > First I'll have to find the printout, which is stored among
> > thousands of other things I have. I saw it a few weeks
> > ago so there is hope.

I have found it! How many programmers can find 17 compiler printouts of a program they wrote in 1974/75? It was my last COBOL program, incidentally. I even found some punch cards with the original source on. The company who employed me does not exist any more.

> *Post by Arnold Trembley*
> Well, I don't want to give you a huge typing assignment, but I am
> curious as to how UT06 worked.

UT06 has 1545 lines, or 25 pages of lineflo.
I even found the manual for it, which has 21 pages.

> *Post by Arnold Trembley*
> > *Post by DSlash*
> > But first, it's based on pre-1974 COBOL, and as it stands, there are
> > three predefined areas, Record, Print and Work areas.

Actually in addition there is a Card area (for the card reader), three printer Heading areas (H, I, J) and a printer control area A.

...

Yes for that the compiler would generate two instructions in its object table, each with an op-code and (up to) four parameters.

The op code for IF-equal-character is 7, and the following parameters

would be the location of R12, the location in a literal table of "TRANS",
a length of 5, and the instruction step to go to if the test was false.

The op code for MOVE-character is 15, and the following parameters would
be the location of "LONDON" in the literal table, the location of P20,
and a length of 6.

> *Post by Arnold Trembley*
> Did UT06 do both compilation and execution?

UT06 was a single COBOL program of 13056 24-bit words consisting
of a compiler compiling pseudo-COBOL source to an internal table, and an
interpreter that executed the instructions in the table, plus a debugging
decompiler that translated the table back to a form vaguely like the
original source.

> *Post by Arnold Trembley*
> Perhaps I am assuming too
> much, but that sounds similar to IBM Series/1 EDL (Event Driven
> Language), used on an extinct 16-bit minicomputer. Technically, it
> was an interpreter, but in practice the performance was very close to
> native object code.

With UT06, most instructions used subscripted data, and there was the
slight overhead of selecting the next instruction in the table.
But efficiency was never a problem, and machines so much faster now.

> *Post by Arnold Trembley*
> Rather than typing in the original UT06 source code (if that's too
> much of a chore), perhaps you could just describe the general approach
> in a little more detail. For example, how did UT06 handle
> arithmetic? Besides the IF and MOVE statements from the previous
> example, what other kinds of statements did UT06 handle? READ?
> WRITE? GOTO?

UT06 was originally written for ICL who had binary numbers held in 8, 16,
24, 32, 40 or 48 bit fields, and display numbers of up to 13 digits, so
UT06 had to handle all of those, and convert between them.
Possible editing was: leading zeroes, zero suppress, floating minus,
decimal point.

IF-tests worked with all of those, and also with alpha literals of up to
79 characters (to fit on a punch card!) and even with single bit fields,
which were useful as Y/N flags in records.

UT06 keywords were:

File definition: IN OUT [MT ED DA GENERATION BLOCK] (up to 15 files)
Input/Output: READ [AT END] WRITE PRINT
Instructions: MOVE [TO] GO TO PERFORM STOP CALL
ADD [TO] SUBTRACT [FROM] MULTIPLY [BY] DIVIDE [INTO]
IF [< = > LESS THAN EQUAL TO GREATER THAN NOT] .

> *Post by Arnold Trembley*
>> *Post by DSlash*
>> With UT06 I never implemented symbolic names. I did that later with
>> other languages, which I should probably discuss elsewhere.
>
> Please, tell me more!

I've posted some details elsewhere in this thread.
The other languages departed more and more from COBOL so maybe there is a
more suitable newsgroup for a discussion.
On the other hand some might be interested in how readable code is when
you reduce COBOL keywords to the absolute minimum.
Also, by having instructions in uppercase and fieldnames in lower case
there are no reserved words, so very old programs will continue to
compile for ever, no matter what new instructions are implemented.

James J. Gavan 18 years ago

...

Good. I'm sure there's more than one of us intrigued by you description of

UT06.

And as for going down Memory Lane, 'By GEORGE he's got it'. I didn't program back then (systems analyst) but you rang a bell mentioning the GEORGE O/S. (Unigate - '63 - '67, ICT 1500 with MCF; Debenhams - '67 - '75 - ICL 1900 series, hard disks, OMR document Reader (UDT)).

Jimmy, Calgary, AB

DSlash 18 years ago

> *Post by James J. Gavan*
> Good. I'm sure there's more than one of us intrigued by you description of
> UT06.
> And as for going down Memory Lane, 'By GEORGE he's got it'. I didn't program
> back then (systems analyst) but you rang a bell mentioning the GEORGE O/S.
> (Unigate - '63 - '67, ICT 1500 with MCF; Debenhams - '67 - '75 - ICL 1900
> series, hard disks, OMR document Reader (UDT)).

There's lots of work been done creating those old systems. It seems a pity for nothing much to be kept. But is there any point? I think so.
I once wrote BEST programs for NCR, and that was a truly amazing program generator designed in the 1960s.
I still have printouts of the generated code :)

Here's the basics of the UT06 program generator source code.
There's a 50 line routine called READ-WORD that gets the next word of source off a card. It handles anything including alpha and numeric literals, end of card etc.
The code for each instruction calls READ-WORD as needed and generates code into CELL which is then moved with STORE-CELL to the big table of object code CELL-STORE, then it goes to NEXT-WORD to find another source instruction.
Not very structured I admit, but a GO TO is much more efficient than some kind of loop and test, and this is a compiler right?, and needs to be efficient.

The RUN-GENERATED-PROGRAM SECTION is really the guts of UT06.
The GO TO ... DEPENDING ON code is what the whole thing is about.
It just cycles around until it hits a STOP instruction.
There, see it's easy - anyone could write a compiler in COBOL :)

```
______UT06 Program Generator__Created Oct 1974_________

[snip 40 line setup
NEXT-WORD.
PERFORM READ-WORD.
CHECK-WORD.
[snip 8 line setup
IF WORD-2-CHAR = "AD" GO TO COMPILE-ADD.
IF WORD-2-CHAR = "DI" GO TO COMPILE-DIVIDE.
IF WORD-2-CHAR = "EL" GO TO COMPILE-ELSE.
IF WORD-2-CHAR = "GO" GO TO COMPILE-GO.
IF WORD-2-CHAR = "IF" GO TO COMPILE-IF.
IF WORD-2-CHAR = "MO" GO TO COMPILE-MOVE.
IF WORD-2-CHAR = "PE" GO TO COMPILE-PERFORM.
[snip 20 similar lines
COMPILE-ADD.
MOVE TYPE-ADD TO CELL-TYPE.
GO TO COMPILE-ARITHMETIC.
```

```
COMPILE-GO.
MOVE TYPE-GO TO CELL-TYPE.
PERFORM READ-WORD.
IF WORD-2-CHAR = "TO" PERFORM READ-WORD.
MOVE WORD-16-CHAR TO CELL-LABEL.
PERFORM STORE-CELL.
GO TO NEXT-WORD.
[snip 720 similar lines

RUN-GENERATED-PROGRAM SECTION.
[snip 20 line setup
NEXT-CELL.
ADD 1 TO CELL-POINT.
ACTION-CELL.
MOVE CELL-STORE(CELL-POINT) TO CELL.
BRANCH-ON-CELL-TYPE.
GO TO
RUN-ADD
RUN-DIVIDE
RUN-GO
RUN-IF-EQUAL-CHARS
RUN-IF-NOT-EQUAL-CHARS
RUN-IF-GREAT-CHARS
RUN-IF-NOT-GREAT-CHARS
[snip 14 similar lines
RUN-MULTIPLY
RUN-PERFORM
RUN-PRINT
RUN-READ
RUN-STOP
RUN-SUBTRACT
RUN-WRITE
DEPENDING ON CELL-TYPE.

RUN-ADD.
ADD BINARY-WORD(CELL-LOC1) TO BINARY-WORD(CELL-LOC2).
GO TO NEXT-CELL.
RUN-GO.
MOVE CELL-EXIT TO CELL-POINT.
GO TO ACTION-CELL.
[snip 330 similar lines
```

RKRayhawk 18 years ago

DSlash ***@hotmail.moc
Date: 10/1/03 11:52 PM EST
Message-id: <blgb3h$u22$***@lust.ihug.co.nz>

posts

<<
Not very structured I admit, but a GO TO is much more efficient than some kind of loop and test, and this is a compiler right?, and needs to be efficient.

The RUN-GENERATED-PROGRAM SECTION is really the guts of UT06.
The GO TO ... DEPENDING ON code is what the whole thing is about.
It just cycles around until it hits a STOP instruction.

There, see it's easy - anyone could write a compiler in COBOL :)
Your posts are most interesting.

I do disagree with your comment
<<
... Not very structured ...
That is totally wrong, IMHO. Like, totally!

A computed GOTO is what modern COBOL INVOKE does! A computed GO TO is what 'message handing' is in SmallTalk and early efforts at object programming in C. (In many early efforts 'message' were just data reduced to an index to use old geezer terminology. Intermediate geezers would call this 'transaction processing'.)

It is entirely structured.

When Kristen Nygaard and Ole-Johan Dahl extended ALGOL to be a SIMULAtot they had to build something into the compiler that selected methods (they too called them PROCEDURES) from a list. You can bet the references to PROCEDURES of a CLASS were in an array, and that index data items inside of the compiler were used to select them.

A computed GO TO is a wonderful thing.

It is excess GO TOs that are a problem. In COBOL simple GO TOs, computed GO TOs and ALTER GO TOs decrease productivity of technical workers as they accumulate in a program.

This can be translated to modern times for modern business applications managers. The deeper we nest the inheritance, the harder it is going to be to maintain these systems, because of decreasing understanding.

Best Wishes,
Bob Rayhawk
***@aol.com

DSlash 18 years ago

> *Post by DSlash*
> Not very structured I admit, but a GO TO is much more efficient than some
> kind of loop and test, and this is a compiler right?, and needs to be
> efficient.
> I do disagree with your comment
> <<
> ... Not very structured ...
> That is totally wrong, IMHO. Like, totally!
> A computed GO TO is a wonderful thing.

Yes I agree it's a wonderful thing, in fact it's the first piece of code that I wrote for UT06.
I was not criticizing the computed GO TO as unstructured.

I didn't make it clear that I was referring to other GO TOs in UT06, where the compiler code has a number of routines at the top which are branched to depending on whether another keyword needs to be read or not, and other similar pieces of code.

Some instructions have optional parameters, so UT06 doesn't know that they are omitted until it has read another keyword.
For instructions that don't have parameters, the compiler branches to the place that reads another keyword.

In theory there is a structured way of doing that, but I was worried
about branching out of a PERFORM when unexpected things happened, e.g. a
compile error, or end of source with a missing parameter.

RKRayhawk 18 years ago

DSlash ***@hotmail.moc
Date: 10/1/03 11:52 PM EST
Message-id: <blgb3h$u22$***@lust.ihug.co.nz>

posted a front end if/elif as

```
<<
CHECK-WORD.
[snip 8 line setup
IF WORD-2-CHAR = "AD" GO TO COMPILE-ADD.
IF WORD-2-CHAR = "DI" GO TO COMPILE-DIVIDE.
IF WORD-2-CHAR = "EL" GO TO COMPILE-ELSE.
IF WORD-2-CHAR = "GO" GO TO COMPILE-GO.
IF WORD-2-CHAR = "IF" GO TO COMPILE-IF.
IF WORD-2-CHAR = "MO" GO TO COMPILE-MOVE.
IF WORD-2-CHAR = "PE" GO TO COMPILE-PERFORM.
```

Anyone can answer this simple question.

Wasn't there a thing with early COBOL that all the reserved words had a unique
pair of first two letters? Did this apply to all reserved words or just the
verbs? Does it still apply?

Richard 18 years ago

> *Post by RKRayhawk*
> Anyone can answer this simple question.
> Wasn't there a thing with early COBOL that all the reserved words had a unique
> pair of first two letters?

I very much doubt it - DISPLAY DIVIDE.

You are probaly thinking of BASIC.

> *Post by RKRayhawk*
> Did this apply to all reserved words or just the
> verbs? Does it still apply?

It never did.

I do recall an ICL product RAPIDWRITE that used 2 character operation
codes and fixed format cards that would generate Cobol, or possibly
just used the Cobol compiler to generate machine code.

Richard 18 years ago

> *Post by DSlash*
> The company who employed me does not exist any more.

Motor Specs became Repco (N.Z.) Ltd. I recall doing some support work
at the Anzac Ave computer site, I worked for ICL. They had designed a
daily stock control update system and it was going to take > 24 hours
for each run.

Names Carol Price and Brian McKenna spring to mind.

DSlash 18 years ago

> *Post by Richard*
>> *Post by DSlash*
>> The company who employed me does not exist any more.
>
> Motor Specs became Repco (N.Z.) Ltd. I recall doing some support work
> at the Anzac Ave computer site, I worked for ICL. They had designed a
> daily stock control update system and it was going to take > 24 hours
> for each run.

Was that prior to May 1974? As it happens I have a printout of that
program, P512, first written in Jan 1974. I understand that it took
several hours to run.

My first job when I arrived at Motor Specs in May 1974 was to write a
stock picking program separate from the main stock update P512.
The requirement was that this picking program be as fast as possible, and
my (COBOL) version took half an hour to run each day.
I am still aghast that I used an ALTER GO TO to get the maximum speed
from it, but it did the trick. The main update was then run weekly
without the stock picking requirement.

> *Post by Richard*
> Names Carol Price and Brian McKenna spring to mind.

Yes I met Carol again years later. She was the only COBOL programmer I've
met who could code a COBOL program from the beginning to the end without
going back to add more data definitions etc :)

Motor Specs was certainly the friendliest place I've worked.

RKRayhawk 18 years ago

DSlash ***@hotmail.moc
Date: 10/1/03 6:33 PM EST
Message-id: <blfocc$i7l$***@lust.ihug.co.nz>

Speaking of an application program said

<<
I am still aghast that I used an ALTER GO TO to get the maximum speed
from it, but it did the trick.
In modern object oriented system overriding inherited class methods with
different methods is an ALTER GO SUB. Use of 'this' pointer and
'super' pointer are transitory ALTER GO SUBs.

The COBOL ALTER GO TO construct is entirely valid. Like a simple GO TO it can
be misused, but the concept is useful when used in moderation, even in
application code.

The meaning of ALTER GO TO is difficult to pass on to the next programmer. The
meaning of a program with many GOTOs is difficult to pass on to the next
programmer. The issue is the productivity of the laborers. It is crude, but
simple. We do grind them to a halt if we generate code with many of these, but
the constuct in isolation is not difficult.

Actually, if you follow one tenet of the more radical variations of structured
programming, you take a lot of pressure off of GO TOs.

Honestly, this is rare, but if you actually keep your program size to a page or
two of PROCEDURE DIVISION, then what is the problem with ALTER GO TO? It's not
like business code is a Rubik's Cube. What's the problem?

But more seriously, it is excess jumps that make it hard for a person to
understand. They just can't track it, especially under pressure.

The ALTER GO TO is a legitimate construct. You cannot override a class method
in modern technologies without do exactly that! And I am sure that deeply
enherited application classes take up many, many pages of code. In fact if you
are dealing with polymorphism you must read the pages backwards, scout's honor.

A polymorphed method is named at the bottom (or atleast a lower level) of the method name sequence, yet if it abends or offends you with its behavior, you may need to read all the way through the inheritance sequence to actually understand how 'that' method was invoked. Unquestionabley the modern graphical toolsets help greatly in this.

A modern programmer can not even succeed in school if they do not understand how things are altered when we inherit. The modern situation is much more cerebral than ALTER GO TO. Although, as I have posted elsewhere, the modern OOP tools automate the coder, so that they do not have to code the ALTER GO TO. The alteration is in the compiler, interpreter, JIT compiler, or the run-time. But you cannot master it unless you understand the method address alteration.

Just as your code was made more efficient with the ALTER GO TO, modern systems are efficient at run time, and the tools do make the technician efficient in the coding. But ya gotta understand the ALTER, even if you call it 'inheritance override', 'super' or 'this'; or you will be lost.

Best Wishes,
Bob Rayhawk
***@aol.com

DSlash 18 years ago

> *Post by DSlash*
> I am still aghast that I used an ALTER GO TO to get the maximum speed
> from it, but it did the trick.
> In modern object oriented system overriding inherited class methods with
> different methods is an ALTER GO SUB. Use of 'this' pointer and
> 'super' pointer are transitory ALTER GO SUBs.
> The COBOL ALTER GO TO construct is entirely valid.
> Like a simple GO TO it can be misused, but the concept is useful
> when used in moderation, even in application code.

Most DP departments that I worked for had standards that banned ALTER GO TO, among other things.
But in this case efficiency was paramount, as the only other option was to buy a much faster mainframe to run the stock picking program, which was the central program for the company.

It was impossible to write it in PLAN, the ICL assembler, with any certainty of it working properly. In fact there's a little story about what happened when the first system test of my program was done compared with the old program. Mine was then used unchanged!

Arnold Trembley 18 years ago

> (snip)
> UT06 has 1545 lines, or 25 pages of lineflo.
> I even found the manual for it, which has 21 pages.
> (snip)
> File definition: IN OUT [MT ED DA GENERATION BLOCK] (up to 15 files)
> Input/Output: READ [AT END] WRITE PRINT
> Instructions: MOVE [TO] GO TO PERFORM STOP CALL
> ADD [TO] SUBTRACT [FROM] MULTIPLY [BY] DIVIDE [INTO]
> IF [< = > LESS THAN EQUAL TO GREATER THAN NOT] .

How were files specified, in order to disambiguate one from another?
Did UT06 support fixed-length, variable-length, or line-sequential files?

How were labels or targets of GO TO and PERFORM written in UT06 source code? How were they handled in the execution table?

I think this is fascinating! I can almost see how to write it.

--
http://arnold.trembley.home.att.net/

DSlash 18 years ago

On Thu, 02 Oct 2003 08:15:05 GMT, Arnold Trembley

> *Post by Arnold Trembley*
> How were files specified, in order to disambiguate one from another?

From the UT06 manual:

The following files may be open at any one time:
Card Reader, Printer,
One MT or EDS (serial or sequential) input file
One MT or EDS (serial or sequential) output file

Mutiple input files:
If more than one IN statement is present, the files defined by IN statements will be read one at a time in order of definition. At the end of the last input file the AT END action will be taken.

File numbers may be differentiated by using file numbers in the range 0 to 15. When reading, all input files with the file number specified in the read statement will be read once, in the order of the IN statements, before the AT END statement takes effect.

A file may be read an unlimited number of times by issuing another read after end of file.

READ [Fn] [AT END statement-1[statement-2 ... statement-n]] .

> *Post by Arnold Trembley*
> Did UT06 support fixed-length, variable-length, or line-sequential files?

Being ICL, all records had a record length at the beginning, so there could be any mixture of different length records in a single file.

> *Post by Arnold Trembley*
> How were labels or targets of GO TO and PERFORM written in UT06 source code? How were they handled in the execution table?

A label had to start with a numeric character and be up to 16 characters long. That was so a label could not be confused as an instruction keyword. Because UT06 source could be entirely free-form to allow it to be passed as parameters from the GEORGE operating system, UT06 made no assumptions about where a label might be, unlike COBOL where a label must be at the start of a line.
Therefore in UT06 there could be many labels on one line.
You could of course make it look like COBOL if you want.

In the execution table there was an op code for a label that was just ignored at run-time. The label name was put immediately after the label op-code, to be checked at end of compile, and left there for the debugging decompiler.

When a GO TO or PERFORM was compiled, the label name was initially put just after the op-code. Then at end of compile, the execution table was searched for GO TO and PERFORM op-codes, and for each of those the table was searched for the label. The cell number of the cell after the label was then put as one of the parameters for the GO TO/PERFORM.

> *Post by Arnold Trembley*
> I think this is fascinating! I can almost see how to write it.

Well I can in time type the whole of UT06.
I doubt if I can scan jiggly old lineprint, but I'll see.
When I depart this life, someone will throw all this stuff away.
I've had cancer twice so it may not be long.

Apart from just historical curiosity, is there really a good use for UT06
now? As a fairly powerful utility in a COBOL shop it could be useful, but
I don't think there's any point in adding all modern COBOL features to
it.

Besides since 1975 I've worked on a number of self-compiling compilers
for other commercial languages, and made program generators which I think
are much better since they are non-procedural, take minutes to generate,
and have no bugs.

RKRayhawk 18 years ago

DSlash ***@hotmail.moc
Date: 10/2/03 5:32 AM EST
Message-id: <blgv0a$hln$***@lust.ihug.co.nz>

<<
Well I can in time type the whole of UT06.
I doubt if I can scan jiggly old lineprint, but I'll see.
When I depart this life, someone will throw all this stuff away.
If you can get your pages in machine transferable image, I should think that
you could find some volunteers to do a page or so of typing. FAX or GIF or
JPEG.

If you are placing the code in a public domain, then transfering the graphic
image is not less
dramatic.

There is no reason for you to do all of the typing.

You probably can get assistance on the proofing as well if all volunteers have
the graphical image.

If your equipment at home is not best for the task consider using a retail
joint that has good camera technology. Picking up old images on large printout
is not difficult if you have the right equipment. The camera at the core of the
machine is the issue. A request for graphics file output is not unheard of in
modern copy shops.

Did you say you have like 20 some pages?

Best Wishes,
Bob Rayhawk
***@aol.com

DSlash 18 years ago

> *Post by RKRayhawk*
> If your equipment at home is not best for the task consider
> using a retail joint that has good camera technology.
> Picking up old images on large printout is not difficult if
> you have the right equipment. The camera at the core of the
> machine is the issue. A request for graphics file output is
> not unheard of in modern copy shops.
> Did you say you have like 20 some pages?

UT06 has 1545 lines, or 24 pages.
By photocopying it first, I have managed to get a reasonable conversion
to text using TextBridge.
Some lines are even correct!
I'll persevere next week when I have access to a better photocopier.

When it's done shall I email it, or post it in the group in several
chunks? There's been nearly 1545 lines of discussion already.

Arnold Trembley 18 years ago

> (snip)
> UT06 has 1545 lines, or 24 pages.
> By photocopying it first, I have managed to get a reasonable conversion
> to text using TextBridge.
> Some lines are even correct!
> I'll persevere next week when I have access to a better photocopier.
> When it's done shall I email it, or post it in the group in several
> chunks? There's been nearly 1545 lines of discussion already.

I would like to get a copy by email. Would it be possible to put it
on a web page where interested parties could download it? Then you
would only need to post a URL.

With kindest regards,

--
http://arnold.trembley.home.att.net/

d*@panix.com** 18 years ago

In article <blnupi$o93$***@lust.ihug.co.nz>,
DSlash <***@hotmail.moc> wrote:

[snip]

> *Post by DSlash*
> When it's done shall I email it, or post it in the group in several
> chunks? There's been nearly 1545 lines of discussion already.

Please be so kind to email a copy to me; if you then express no objections
I will post it myself (with appropriate attributions) because I believe
that bandwidth is cheap enough and Google is a passable-enough archive.

DD

DSlash 18 years ago

> *Post by d***@panix.com*
> [snip]
>> *Post by DSlash*
>> When it's done shall I email it, or post it in the group in several
>> chunks? There's been nearly 1545 lines of discussion already.
>
> Please be so kind to email a copy to me; if you then express no

objections

> *Post by d***@panix.com*
> I will post it myself (with appropriate attributions) because I believe
> that bandwidth is cheap enough and Google is a passable-enough archive.

I'd like what is posted to be reasonably accurate. TextBridge has
converted the scanned text, and it likes to convert "I" (capital I) into
"1" (one) or "l" (small L), and many other errors.
So it would be nice if someone could run a COBOL compiler on it to at
least check the syntax , before it's posted as a whole.
I'm 25% of the way fixing up the scan, and have reached the Procedure
Division.

I have discovered that the latest printout that I have does not have all
the features specified in the manual. In particular, the code for
arithmetic with other than one word fields is not there, and nor is the
code for single bit fields (which is probably non-standard ICL extension
of COBOL anyway).

It does depend on what you guys want it for. If you actually want to use
it as a file utility, it might be worth someone getting it going as it is
before any improvements are made. Or as a basis for a total rewrite it's
OK as it is as a guide.

I have discovered today that UT06 is still in use after 29 years, by a different company, on a different mainframe (IBM) and in a different country from where it started. That shows the stability of COBOL!

d*@panix.com** 18 years ago

> *Post by DSlash*
>> *Post by d***@panix.com*
>> [snip]
>>> *Post by DSlash*
>>> When it's done shall I email it, or post it in the group in several
>>> chunks? There's been nearly 1545 lines of discussion already.
>>
>> Please be so kind to email a copy to me; if you then express no objections
>> I will post it myself (with appropriate attributions) because I believe
>> that bandwidth is cheap enough and Google is a passable-enough archive.

[snip]

> *Post by DSlash*
> It does depend on what you guys want it for.

[snip]

> *Post by DSlash*
> I have discovered today that UT06 is still in use after 29 years, by a
> different company, on a different mainframe (IBM) and in a different
> country from where it started. That shows the stability of COBOL!

I cannot speak of what others might want it for... but what you say here is sufficient reason for me. Flexibility, longevity and stability... who needs those from a computer language, eh?

DD

Arnold Trembley 18 years ago

> *Post by d***@panix.com*
> [snip]
>> *Post by DSlash*
>> When it's done shall I email it, or post it in the group in several
>> chunks? There's been nearly 1545 lines of discussion already.
>
> Please be so kind to email a copy to me; if you then express no objections
> I will post it myself (with appropriate attributions) because I believe
> that bandwidth is cheap enough and Google is a passable-enough archive.
> DD

Dslash sent me the source code for UT06 in two sections. I have reformatted it into standard 80-column COBOL, and zipped up several ms-dos text files. It's about 56K bytes.

You can download the zip file from:
http://home.att.net/~arnold.trembley/ut06.zip

I have it posted on my web page at:
http://home.att.net/~arnold.trembley/prog.htm

Here's a list of the files in the zip archive:

UT06 TXT 49,324 10-09-2003 1:21a UT06.txt
UT06-A01 COB 142,927 10-22-2003 12:46a ut06-a01.cob
UT06-A01 ERR 17,920 10-22-2003 12:46a ut06-a01.ERR
UT06-B01 COB 145,334 10-22-2003 12:46a ut06-b01.cob
UT06-B01 ERR 1,738 10-22-2003 12:47a ut06-b01.ERR
UT06-MAN TXT 15,059 10-22-2003 1:03a UT06-Man.txt

UTO6.TXT is a copy of the previous discussion in comp.lang.cobol, since it has some useful comments are how UT06 is intended to work.

UT06-A01.COB is the original ICL COBOL source, as close as I can make it.

UT06-A01.ERR is a compile error listing, but it's based on IBM VS

COBOL II syntax, which is why there are so many errors listed.

UT06-B01.COB is a version that I modified, attempting to get UT06 to compile cleanly with IBM VS COBOL II syntax. There's still a lot of work to do on the file definitions, building the print subroutine to replace the A510 subprogram, handling the runtime switches, etc.

UT06-B01.ERR is the error listing for the "IBM compatible" version. There's still some work to do on the file definitions.

UT06-MAN.TXT is the manual for UT06/5, but the original ICL COBOL source code was for an earlier version without most of the arithmetic functions in the manual.

I'm going to try to clean up the COBOL II version a little more, but I don't have samples of UT06 code for input into the compiler, so testing might take a while. If anyone wants to try it out, please let me know of any corrections or improvements that can be made.

Good luck!

--
http://arnold.trembley.home.att.net/

Brian W Spoor 15 years ago

...

A friend has just pointed me to this thread and sent me a copy of the source archive.

I've just compiled UT06-A01.COB using #XEKB/#XPCK under GEORGE 3 - it gave 3 minor compilation errors, now corrected in a version A02 and a lot of warnings (col 73-80 should be a fixed program identity or blank, not a repeat of the line number).

It #UT06 now compiles cleanly, but gives 3 missing segments on consolidation - I'll look at these another night.

Do you mind if I put #UT06 on my web site about the ICL 1900 series computers? See:-
http://www.icl1900.co.uk/

I am currently running G3 under an executive emulator (in beta test) on a W98 system. There is still a live site in Russia running G3 under DME.

d*@panix.com** 15 years ago

...

Better late than never, I guess.

DD

Arnold Trembley 15 years ago

Top Post - nothing below quoted text.

I have no objections to your posting #UT06 on your website.

I can't speak for Dslash, but if he didn't mind me posting a version on my website, I suppose he would not object either.

With kindest regards,

...

--
http://arnold.trembley.home.att.net/

Richard 18 years ago

> *Post by DSlash*
> Being ICL, all records had a record length at the beginning, so there
> could be any mixture of different length records in a single file.

I suspect that the program will be unusable on most systems because these require a specific FD to be compiled with specific record sizes or variable record range.

With the old XEKB (or XE11 to XE20) it ws the block size that was required to match the actual file.

With disk ISAM files the keys much match between the compile and the actual disk file. It may be necessary to have a CALL for a file interface module and then to generate a module per actual file to access the file correctly.

DSlash 18 years ago

> *Post by Richard*
>> *Post by DSlash*
>> Being ICL, all records had a record length at the beginning, so there
>> could be any mixture of different length records in a single file.
>
> I suspect that the program will be unusable on most systems because
> these require a specific FD to be compiled with specific record sizes
> or variable record range.
> With the old XEKB (or XE11 to XE20) it ws the block size that was
> required to match the actual file.

UT06 had a predefined output tape file for each block size in use in the installation, I.e. sizes 64, 128, 512 and 2048, and an input tape file of block size 2400.

You could say OUT MT "TEST FILE" BLOCK 512
and UT06 would select the correct tape file to do that.

Martin Jones 18 years ago

> *Post by DSlash*
> Way back in 1974 I was the systems programmer in an all COBOL DP
> department, using an ICL 1902A mainframe, with punch card input and tape
> and disk storage.

<snip>

Hi Dave

I hadn't read more than three paragraphs when I had the suspicion that the subject of the post was UT06, (DSlash was also a strong clue). As a junior prog at MSI Corp in '77 I was at first baffled but soon enamoured of the power and flexibility of UT06. My crowning achievement in UT06 was a program to edit the card deck for JF's timesheet and billing system to weed out the bugs before running the reports. Lots of fun at Anzac Ave. Loads of talent in the punchroom if you could get around the 'mother hen' supervisor who guarded her chicks quite zealously.

We later, of course, worked together at IDAPS with another interpretive language - TNT which syntactically was like no other language I've seen before or since.

Whilst I was working on Paxus Polisy in the mid-eighties on Concurrent minicomputers, (nee Perkin Elmer), I could have made good use of a port of UT06 to this platform. The PE toolchest was woefully bare. All a bit late

now
alas.

Good to hear of you although you can't be in you dotage yet :)

Regards
Martin

DSlash 18 years ago

On Tue, 30 Sep 2003 12:38:48 GMT, "Martin Jones"

> *Post by Martin Jones*
> I hadn't read more than three paragraphs when I had the suspicion
> that the subject of the post was UT06, (DSlash was also a strong
> clue).

Hi Martin, when I went to IDAPS they had nicknames there, and it amused them to call me DSlash which is an abbreviation of D/L or DataLength, and to give me operator number 69!

> *Post by Martin Jones*
> As a junior prog at MSI Corp in '77 I was at first baffled but soon
> enamoured of the power and flexibility of UT06. My crowning
> achievement in UT06 was a program to edit the card deck for JF's
> timesheet and billing system to weed out the bugs before running the
> reports.

That's probably different from the original billing system for the department which was written by the Operations manager entirely with UT06. He was refused funding for that system so wrote it himself. He was as pleased as Punch as he knew nothing of system design or programming. He was then quite friendly to me, which was useful to get access to the computer room, and you know what bastards the Op managers usually are towards those nuisance programmers.

The bit I liked best was being able to type an entire program on one card as a parameter for the George operating system.

> *Post by Martin Jones*
> We later, of course, worked together at IDAPS with another interpretive
> language - TNT which syntactically was like no other language I've seen
> before or since.

There's another huge story there. RT who designed TNT, hired me three times and TNT went on to become the language AMPLE into which I incorporated much of UT06.
See http://www.ams.co.nz/ and click on Products/AMPS

AMPLE had the left to right processing and the clarity of COBOL needed for huge business applications, except that keywords were reduced to a minimum, and I replaced those bloody full-stops with END statements!

AMPLE with its Opsys AMPS (with a C++ interpreter) was originally used by Prism Software as a generator and interpreter for their generated packages, hence their applications were double interpreted. The inefficiency of that was never a problem, in fact all TNT/AMPS systems are notable for their extreme speed and conciseness. Compiles still take only seconds (maybe because it uses translate tables rather than table searches, and garbage collects and virtual storage are unnecessary).

See http://www.prism.co.nz/site.asp?item=whyprism&file=whyprism.html

AMPS has gone on to become DB++ somewhere in Australia, and there are two further offshoots of AMPS that I've not heard of for years, written by a number of ex-employees (PW and TMJ where are you?)

> *Post by Martin Jones*
> Whilst I was working on Paxus Polisy in the mid-eighties on Concurrent
> minicomputers, (nee Perkin Elmer), I could have made good use of a port of
> UT06 to this platform. The PE toolchest was woefully bare. All a bit late

> now alas.

You may have noticed that the medical insurance company next door has lost about $40 million with computer processing problems. They should have stuck with TNT.

> *Post by Martin Jones*
> Good to hear of you although you can't be in you dotage yet :)

I'm retired and think I should write a book about all this.

Continue reading on *narkive*:

SEARCH RESULTS FOR *'COBOL COMPILER WRITTEN IN COBOL'* (NEWSGROUPS AND MAILING LISTS)

19 replies	[OT] Fortress Mentality - Doesn't Happen in a Vacuum started 18 years ago	comp.lang.cobol
178 replies	a history question started 17 years ago	alt.folklore.computers
51 replies	Aren't the fuddy-duddies in the group going to comment on Fortress started 15 years ago	comp.lang.fortran
120 replies	a history question started 17 years ago	comp.lang.fortran
105 replies	out in web comics land... started 13 years ago	talk.bizarre

about - legalese