Fortran And The Genesis Of Project Intercept

by John Van Gardner

The Fortran compiler had been released by IBM in April 1957. Fortran stood for Formula Translation. A compiler takes symbolic representation of and equation and generates a machine language program to solve that equation.

My first encounter with Fortran was the day they called me to the machine room and the 716 Printer had printed two words on a page. It said, "Machine error". I turned to the operator and said, "I wonder which machine it's referring to." I always though that if a programmer was smart enough know when a machine had made and error he should be smart enough to tell you which machine. At this time there were no source code listings for Fortran in the field. IBM had spent a lot of money and time developing Fortran and they did not want to make it easy for competitors to copy it. The branch manager was finally able to get a listing on 35 mm microfilm to be under IBM control at all times. After many hours working with our Applied Science Representatives we found the problem. The Lockheed Fortran programmer had coded a routine with no name and a GO TO statement prior to the routine. There was no way to get to the routine. The compiler got confused and printed machine error message.

This was the first of six problems where Fortran printed machine error when nothing was wrong with the machine. As a matter of fact I never saw a machine problem where Fortran printed the message. This kind of problem was bad enough but worse came later. We started getting complaints of the object program not running correctly and the programmer not being able to find anything wrong with the source. They would re-compile the source and this time the object program was ok. We used a 519 Card Reproducer to compare the two object decks and there was a difference. The big problem to overcome was to find a way to determine when and where, in the compilation, the error occurred that caused the decks to be different. I found a programmer that had a source deck that had failed to compile the first time but worked the second time. He gave me a copy of his source deck so I could try to find out why.

I had learned that the Fortran Compiler was too big to reside in core memory all the time. The 704 had two 737 Memory units with 4096 words each and a 733 drum unit (actually two physical drums in one frame) that had a total of 8192 words. The compiler program was broken into sections that resided on magnetic tape. When the tape was loaded a loader program was brought into low core and it loaded the first section of the compiler. When that section was finished it transferred control back to the loader program to load the next section. This sequence was repeated until the compile was complete.

My first goal was to try and find a way to determine which section the failure was in. Since I had a program that would sometimes compile and sometimes not I would use it to determine the difference. I added some code to the loader that would add up all the words in memory except the loader locations and write each total out on a tape. After several runs I got a compile that the object deck checked good in the 519. I now had a tape that had a correct check sum for memory at the end of each section of the compile for that particular program.

Next I added a few more instructions to the loader that would compute the check sum at the end of each section, but instead of writing the check sum to tape it read one from the tape that had the good check sums, and compared it to the one just computed. The very next compile failed with a check sum mismatch and the machine stopped. I now knew which section failed but where in the 8000 memory locations were the words that were different? I modified the loader to write all of memory to a tape unit after it computed the check sum. It took several runs to get a good compile but then I had a dump of memory with the correct data after each section. Using a tape

compare program I found four words in memory to be different. The bad dump had four words of all zeros.

The next morning I met with the Lockheed System Programmer and the IBM Applied Science Representative and told them what I had found. They determined that the area that had the four all zero words was a buffer where data had been read from the 733 Drum. The drum did not have a parity checking circuits but the software used a check sum for each block of data. The first word read would be the check sum of the rest of the words read. The drum data in the good dump looked like a library program routine. We suspected the drum had read from the wrong drum address and got into an unused area containing all zeros. The problem with the check sum method was that if you read a check sum of zero then read in a bunch of zero words and added them up you got zero. Then when you compared to the checksum they were equal so no error was detected.

The 733 Drum had a timing track written on the drum surface and a counter that counted the timing pulses to keep up with the addresses. This timing track would deteriorate with age and had to be checked during a scheduled PM period. When the amplitude got below a certain level it had to be re-written. The heads on the 733 Drum did not actually touch the surface of the drum. They had a very light spring loading. When you installed a head you connected an ohmmeter lead to the head and the other to the drum. You turned an adjusting sleeve until the head contacted the surface then backed off until contact was broken.

Later on we found out that if you lowered the timing track head until it contacted the drum then wrote a new timing track, it would be so large it never deteriorate below the required level.

We resolved our problem with this compile by changing the vacuum tubes in the drum counter circuits. We kept the source deck, check sum tape, dump tape and object deck and used them as a diagnostic test when someone thought there was something wrong with the machine. Later on as the programmers wrote more and more complex programs others failed during compile. I used the same technique to solve them.

All during this problem I had been in contact with Poughkeepsie and they were very interested in my technique. I submitted my ideas to the Suggestion Department and received an initial award of \$100.00 on September 29, 1961 pending further investigation. This type of problem was showing up all over the country and the plant had started running Fortran on machines before they shipped them. Later on during one of my schools in Poughkeepsie I met with R. E. Clark and gave him copies of all my material. He told me they had a 7090 on the production floor that was supposed to ship to London, England in a few days, but it was failing on Fortran Compilations.

We went over to the plant and ran my package on the 7090 and it showed the machine was picking a bit memory in the address field of an instruction. They called for their 7302 Memory expert and we told him what bit was picked and what the address was. He scoped a few driver outputs and pulled out a delay card. He moved some jumpers around on the card and plugged it in. The machine ran the compile for the first time and they were able to ship it. IBM started a project called CEMON (CE Monitor) but later changed it to INTERCEPT and released it to the field for the 7090 in 1963. On February 3, 1964 I received an additional award for my original suggestion of an additional \$375.00.