April,22, 2002

CKMGEANT Developement, " 10 Easy Bugs" and simple ways to avoid them

C. Milsténe

1 Introduction

Parallel code developement can hide unexpected traps. Part of them will be described below, from light to heavy, with actual examples encountered during the CKMGEANT developement with simple solutions and most of it might be useful, even while going from Fortran to C^{++}.

2 The Way of Various Bugs

2.1 Variable Names in the code

When writing code in each individual package, obvious variable names, should be avoided, for example.

PHI, $IPHI$, $NPHI$, $PHIMIN$, $PHIMAX$, ...

There is a high likelihood that in the future it will clash with another variable having the same name and transported through a common.

Example : IPHI, belongs to geant321 common /GCSCAN/. Therefore phi angle bins and counters cannot be called iphi without creating a conflict with the iphi in common /GCSCAN/. Same thing for PHIMIN, PHIMAX, etc

2.2 The name of a Statement

It is better not to assign to a variable the name of a statement, e.g. one should avoid:

real data(84), common(12), implicit(10)

Example : data(84) was used in the versions of CKM_read_user.F prior to V4_0, however data is also a statement, it is not a good practice to use it as a variable.

2.3 Include File Variables

When writing include files for a detector package, a check for obvious or not so obvious variable names is an healthy practice.

Example : NPRT in umscomm.inc, is also a variable of the geant321 common /PRNTFL/ as NPRT(10).

A solution to (1), (2), and (3) is to check the variables used in the include files against those in the directories geant321/include and $CKMGEANT_DIR/include, and to use a flag detector dependant variable, e.g., vvs_iphi, bivs_iphi.

2.4 Same dimensions Variables

While dealing with same dimension variables, which are defined in different routines, it is good practice to have them defined in the same include file and to setup their dimension there.

Example: In the CKM_read package the versions prior to v4_0, the dimension of the Ntuple variables ,(ntag=84) in CKM_read_init.F and data(84) in CKM_read_user.F were defined separatly. To make sure that one does not get changed without the other, it is good practice to have them both in the same include file.

2.5 Dealing with commons

A good practice is to have them defined in include files, to avoid obvious errors (e.g. change of variables in only part of the routines).

e.g. the common/ckmtrack/ is defined separatly in different routines CKM_read.F, CKM_user_read.F. e.g. the common /mycom/ is defined separatly in CKM_read.F, CKM_read_final.F, CKM_read_init.F, CKM_read_user.F, CKM_display.F, dis_event.F, dis_spec.F.

A solution to(4) and (5) is to define an include file containing the common, e.g. mycom.inc, ckmtrack.inc

2.6 Sharing Include file with other packages

When adding a new material in one detector either through GSMAT or for a mixture through GSMIXT and calling CKMGSTMED, one should change the include file ckm_media.inc.

This include file is also loaded in each of the routines which are totally independents of the detector updated, e.g.

- bst_geom.F
- cerenkov.F
- ckmgeom.F
- ckmgstmed.F:
- fvs_geom.F
- hvs_geom.F
- keat_geom.F
- mvs_geom.F
- straw_geom.F
- ums_geom.F

- vacu_geom.F

Solution to (6) It is advisable when compiling and linking the new code of the detector, to include also those routines.

2.7 Indirect action

Changing the code in part of the programs might affect the code of other developers. Example 1: In an attempt to check the effect of a non-homogeneous

VVS magnetic field ,e.g. IFIELD=3, I caused the DMS to be destabilized. At that time, changes were directly done on the source code (CVS was not yet used). The DMS people took immediate action and changed completly their code. I understood that my hands were tied and there was no coming back to my previous code with the homogeneous magnetic field and this ended my study at the very first step. I could only let the DMS people optimize the field and I understood that the magnet is part of the DMS. I did push to make the Magnet code part of it. Example 2: When the Interactions were

introduced in CKMGEANT, by D. Vavilov, the 2γ tracks of the π^0 became one track only,with a double number of hits per track. This was above the maximum number of hits allowed per track (of 100) and part of the hits got lost. This factor which did also affect the rates of the widely spread e.m. showers of the VVS

A Solution: Action should be taken by both the developer making the changes and the developers who keep the code static. a) From the developer who makes the changes to the general code an e-mail summarizing what is attempted in its general lines and, as soon as it is known, indicating which routines have been altered could be sent to the developers. In those routine clear and dated comments of the changes are very useful. It might allow the other developers to understand which indirect effects it has on their software.

As a general rule, it would be very useful that the steering routines e.g. Gustep.F, Gukine.F, Guout.F, etc which call the different detectors would be commented very clearly as well. This should allow the rest of the developers to get a rough understanding of what is being done, where and how, in a very short time. The example below illustrates this point for Gustep.F

- CALL SPJ_GUSTEP(SPJ_BEG, do_return)
- IF (do_return) RETURN

Is it clear ? Is it well commented ? This is true for all of the general routines as well and it would be nice to have at the beginning of the general routines a commented summary of the mode of action adopted and the principal moves. b) From the developers, who have kept their code static:

Sometimes the effect of the change might appear immediatly, e.g. the change of the properties of the magnetic field did translate into "no hits" in the DMS. Most of the time, however, the effect is subtle and is manifested by unstable results. Characteristic parameters such as multiplicites or hits, for that detector, should be recorded in histograms or such. For example the distribution of the energy deposited per Photomultiplier/track/event as a function of the module number was histogrammed twice, at collection (vvs_gustep) and before output (vvs_guout). If the vector output got truncated there are discripancies between the histograms in size and shape.

2.8 Include file not found

Changes in our code may trigger a strange request when we re-compile and link the changed code with the regular makefile commands used in CKM . The common might not be found even though it was unchanged and appears in its usual directory $CKMGEANT_DIR/include.

Example: During the rate studies V Molchanov, R. Coleman and myself were getting at compilation and link the following diagnostics.

- f77 -o vvs_gukine.o -c -O -OPT:Olimit=2000 -I/include -I/includes
- -I/src/geant321 -DCERNLIB_SGI -DCERNLIB_UNIX F"vvs_gukine.F",
- line 10: error(2006): could not open source file "vvscom.inc"
- #include "vvscom.inc"
- 1 catastrophic error detected in the compilation of "vvs_gukine.F".

- Compilation terminated.
- gmake: *** [vvs_gukine.o] Error 2

I succeeded to RE-CREATE that condition in one case so far. I did create my own "repository" and libraries for the detector with a DIFFERENT VERSION of the code than the one in use, I got that same diagnostics.

This kind of situation will be handled by Unix machines like the silicon graphics with just that warning and no error message. it will run and give results varying from run to run. And the fact that those errors suddenly started to appear in my code resulted, from messing in the repository with the code used to create libraries, by the code and repository management.

2.9 Invisible differences

Sometimes corrections to our detector which are CVS recorded do not show up, neither using CVS update nor CVS diff

Example: A CVS update showed only 2 differences which I have extracted from the file that I keep also on my PC.

SEE NEXT PAGE

- <<<<<<< bivs_guout.F
- C $Id: bivs_guout.F,v 1.14 2001/06/25 04:44:59 caroline Exp $
- C $Author: caroline $
- =======
- C
- C $Id: bivs_guout.F,v 1.15 2001/06/28 14:48:37 ramberg Exp $
- C $Author: ramberg $
- >>>>>>> 1.15
- <<<<<<< bivs_guout.F
- 169 format(1x, I6,1x,i3,2x,E13.7,3(1x,f12.6))
- =======
- 169 format(1x, I6,1x,i3,2x,E13.7,3(1x,f12.6))
- >>>>>>> 1.15

Namely, the only apparent difference is the author's name. There must be an explanation for this I am sure !

A few months later making a file difference between BIVS version 14 and version 15, I was surprised to discover that the version number in the repository is now attributed to 2 of my own development versions during which I did change the format. I cant understand how Development Versions became Numbered Versions with the Wrong Number and how, my changes, got attributed to E. Ramberg who did not have anything to do with it ?

- retrieving revision 1.15
- retrieving revision 1.14

- diff -r1.15 -r1.14
- 2,3c2,3
- < C $Id : bivs_guout.F, v1.152001/06/2814 : 48 : 37rambergExp$
- < C $Author : ramberg$
- —
- > C $Id : bivs_guout.F, v1.142001/06/2504 : 44 : 59carolineExp$
- > C $Author : caroline$
- 153,154c153,154
- < 168 format(I4,1x,I3,1x,f9.5,3(1x,f12.6),2(1x,e14.7),1x,f9.5,
- < & 1x,E14.7,1x,i2,1x,f6.3,1x,f9.5, 3(1x,f12.6),
- —
- > 168 format(I4,1x,I3,1x,f6.3,2(1x,f8.4),1x,f8.1,3(1x,f8.5),
- > & 1x,E13.7,1x,i2,1x,f6.3,1x,f8.5, 3(1x,f12.6),
- 156c156
- < 169 format(1x, I6,1x,i3,2x,E13.7,3(1x,f12.6)) —
- > 169 format(1x, I6,1x,i3,2x,E13.7,3(1x,f8.5))
- 162,164d161
- < C Revision 1.15 2001/06/28 14:48:37 ramberg
- < C Change format to match BIVS_read.F
- < C

2.10 Collegue's Code

Finally, when finding a mistake in the code of a collegue developer, just telling him would be extremely helpful, no point to rewrite the code in the exact same logic (its easy to rewrite Hamlet once Shakespeare did it).

Example: When in vvs, vect(5)/vect(4) instead of vect(2)/vect(1) was used to define the angle.

3 Conclusion

The above list of easy bugs is not exhaustive. Additional improvments could be added if the decision to create new versions would be taken with the active participation of all the developers.

The decision to make a new version when developers are in the middle of a big change is not a good idea and no need of a manager who is not a coordinator. In other words, it is bad practice and bad management to create new versions constantly out of synchronization with the code of one of the developers. The SGI-Unix fortran compiler was too forgiving, the Linux and AIX compiler doing a cleaner job. Going to C^{++} will certainly open new options, a better management of the code might also bring further improvements.