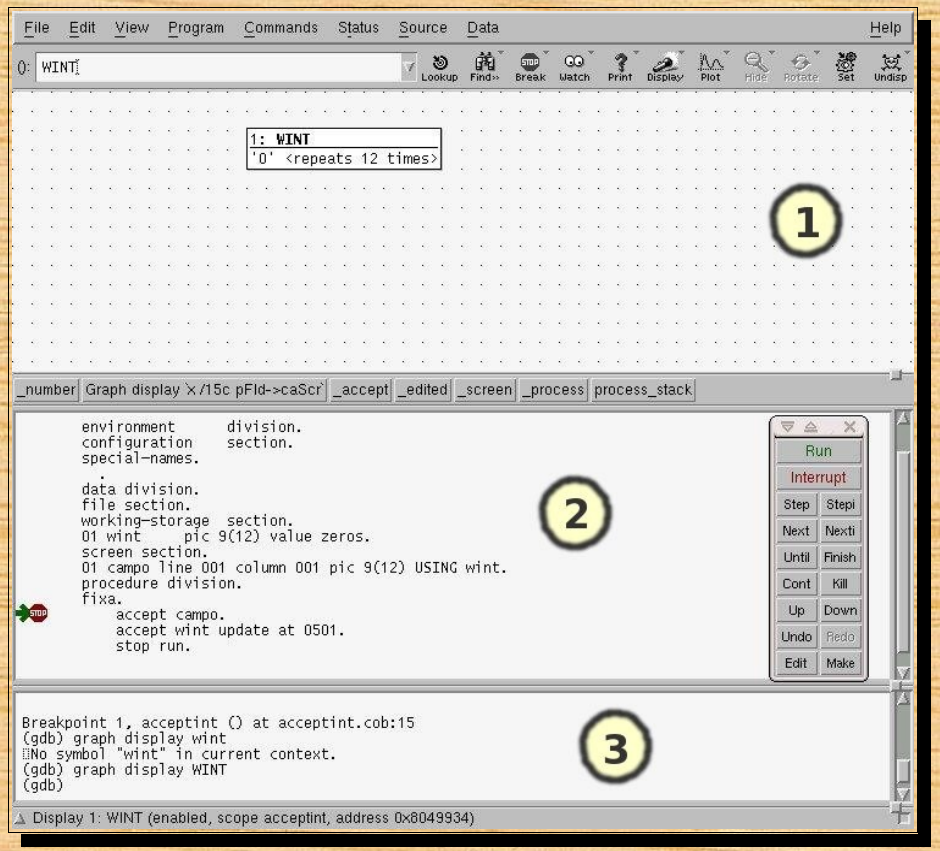
TinyCobol: Depurando com o DDD

http://www.pragana.net/

Introdução

O DDD, **Data Display Debugger,** é um *frontend* (interface visual) para vários depuradores, em especial o GDB, onde ambos são disponíveis na forma de software livre, sob licença GPL.

Um programa compilado pelo TinyCobol com a opção **-g** habilitada contém a tabela de símbolos (stabs) que o gdb, e consequentemente o DDD, podem usar para visualizar o código fonte durante a execução controlada do programa, para fins de depuração. A biblioteca *runtime* do TinyCobol também pode ser compilada com a opção **-g**, para termos a possibilidade de navegar passo-a-passo pela execução das funções desta biblioteca, geralmente escritas em C. Isso permitiria, por exemplo, um desenvolvedor encontrar falhas na implementação das funções dessa biblioteca.



A figura nos mostra uma sessão típica do uso do DDD com um programa cobol. São mostrados tres painéis: **(1)** dados, com visualização de variáveis; **(2)** fonte, mostrando o programa que está sendo depurado; **(3)** interface com o gdb, que nos permite entrarmos comandos diretamente para o gdb.

No painel (2) vemos também uma seta verde, que indica onde o programa está atualmente parado, e logo à sua direita encontramos uma figura vermelha que indica um *breakpoint*, ou ponto-de-parada que foi introduzido pelo operador. Se rodarmos o programa sem nenhum *breakpoint,* ele irá proceder normalmente, sem parar em qualquer lugar.

Botões de controle

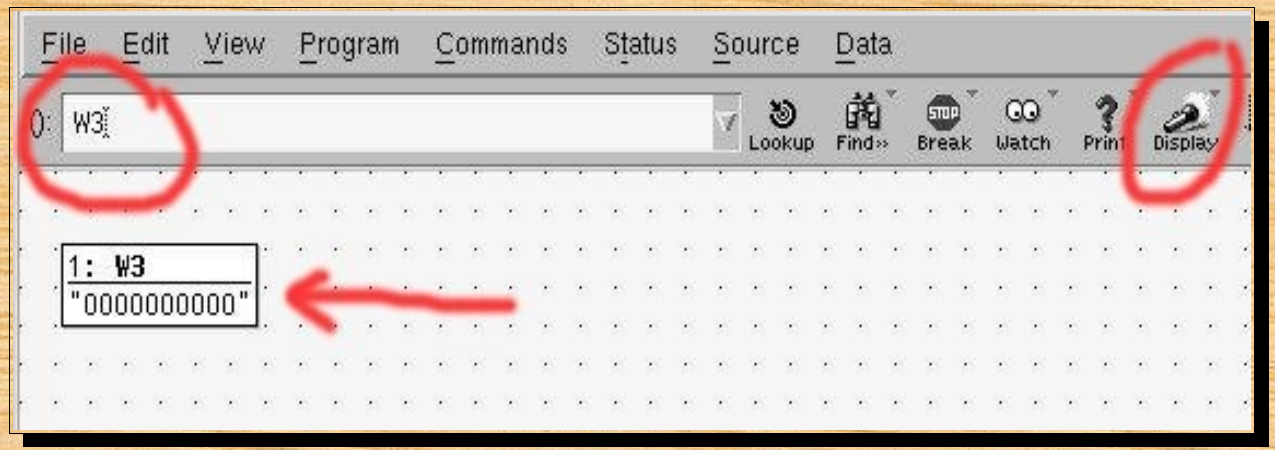
A janela com botões de controle do DDD (figura ao lado) pode ser usada convenientemente para comandar a execução do programa. Essencialmente, podemos abrir o programa fonte e efetura um **duplo-clique** no canto esquerdo de uma das linhas do texto (programa fonte) para adicionarmos um *breakpoint* na posição indicada. Em seguida poderemos usar o botão Run dessa janela (botão na cor verde) para executar o programa até o *breakpoint* ser alcançado. Se o programa está em *loop* (execuando sem parar), podemos usar o botão Interrupt para forçar sua parada. Nesse caso, nem sempre o resultado sera o esperado, pois normalmente ele irá estacionar numa biblioteca do TinyCobol ou na glibc (*runtime* C).

Obs: Também podemos analisar as instruções em linguagem *assembly* (máquina), visualizando-as num quarto painel, que pode ser mostrado pelo menu *Source->Display Machine Code* do DDD.

Os botões **Step** e **Next** servem para avançarmos uma instrução no programa. O primeiro trata uma chamada de rotina como uma instrução normal, enquanto que o segundo (Next) avança sobre um *call* como se fosse uma só instrução.

Como nossas bibliotecas estão escritas em C e há falhas na nossa implementação de *stabs* (tabela de símbolos para o gdb), algumas vezes tanto um Step como um Next nos deixará numa função da biblioteca libhtcobol. Para avançarmos no programa cobol exclusivamente, nesse caso, podemos por *breakpoints* adiante e usarmos o botão **Cont** se não estamos interessados em visualizar o próprio *runtime*. A maioria das instruções, felizmente, se comportam como esperado.

Podemos visualizar o conteúdo de variáveis do programa, colocando *displays* no painel de dados. Para isso, introduzimos o nome da variável em letras maiúsculas na caixa de entrada e pressionamos no *toolbar* o botão **Display**. Variáveis contendo '-' (hífens), devem ser colocados entre apóstrofes para serem reconhecidas, exemplo: **'WS-VAR'**.



Compilando com a depuração habilitada

Para podermos visualizar o código fonte, precisamos ter a tabela de símbols adicionada na compilação. Veja como compilamos um programa **sample.cob**, e o link-editamos para deixá-lo preparado com essa finalidade.

```
htcobol -c -v -g -e sample -D sample.cob
gcc -g -o sample sample.o -L/usr/local/lib \
        -lhtcobol -ldb -lncurses -ldl -lm
```

Vejamos quais as opções usadas e seu significado:

- A opção **-c** indica que queremos compilar o fonte cobol, traduzindo-o para *assembly*, e em seguida montar o código *assembly* (máquina), transformando-o em objeto, mas não queremos link-editar com as bibliotecas.
- A opção **-v** (verbose) seleciona a saida de mais informações sobre as etapas da compilação.
- A opção **-g** é a que produz a tabela de símbols, indispensável para a compilação.
- A opção **-e** seleciona o nome (PROGRAM-ID) que será o ponto de entrada do programa.
- A opção -D escolhe a adição do fonte cobol ao código gerado (opcional).

O resultado dessa compilação será o arquivo sample.o. Na segunda etapa, novamente usamos **-g** para manter a tabela de símbols no executável final. Note que estamos usando o **gcc** e não propriamente o **htcobol** (compilador TinyCobol) para produzir o executável. As opções que se seguem são relativas ao **gcc**, portanto:

A opção **-o** batiza o nome do executável resultante.

A opção **-L<caminho-para-bibliotecas>** adiciona um caminho à lista de caminhos (path) procurados para resolver as chamadas pendentes com as bibliotecas *runtime*.

A opção **-lhtcobol**, e de modo semelhante as outras, **-ldb -lncurses -ldl -lm**, indica quais as bibliotecas que deverão ser pesquisadas, nessa ordem, para resoluçso de símbolos no programa link-editado. A biblioteca lib**htcobol** deverá sempre ser usada, bem como a lib**db**. A biblioteca **ncurses**, somente quando a SCREEN SECTION é declarada ou alguma das opções do ACCEPT em tela (posicionamento) são usadas no programa.

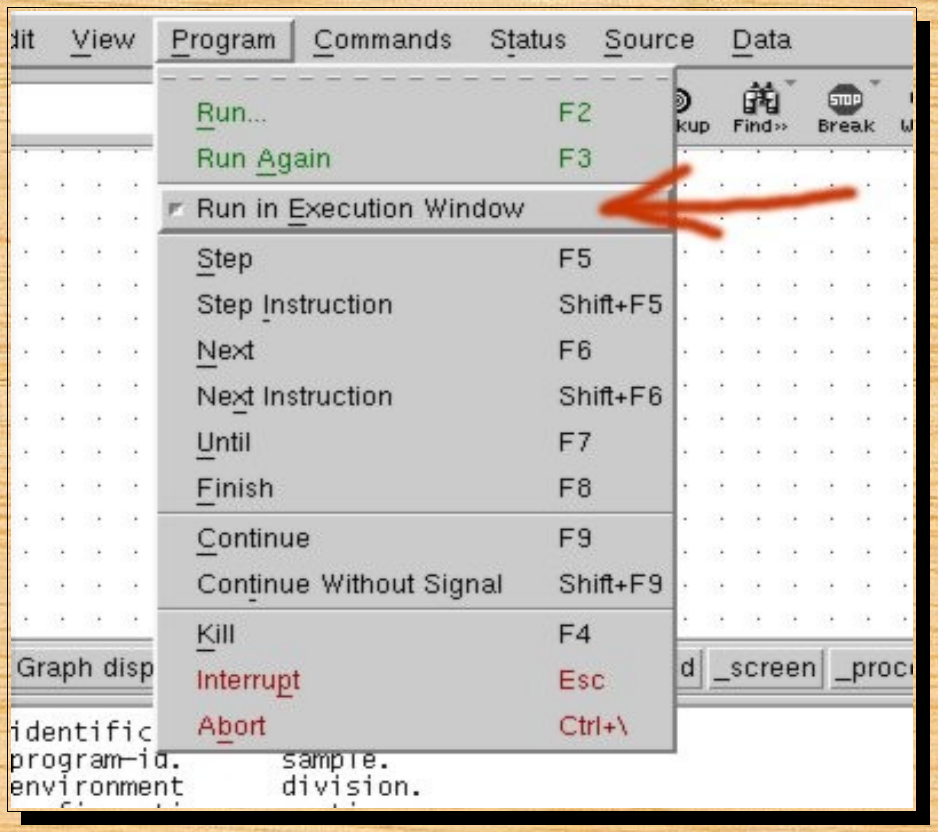
Exemplo de programa

Podemos fazer um simples experimento com o seguinte programa:

```
identification division.
program-id.       sample.
environment       division.
configuration     section.
special-names.
data division.
file section.
working-storage   section.
01 W1         PIC 9(10) VALUE 1234.
01 W2         PIC 9(10) VALUE 5678.
01 WS-TOT     PIC 9(10) VALUE ZEROS.
screen section.
procedure division.
display "W1=" W1
display "W2=" W2
display "WS-TOT (antes)=" WS-TOT
compute WS-TOT = W1 + W2.
display "WS-TOT (depois)=" WS-TOT
stop run.
```

Note que estamos declarando uma SCREEN SECTION, o que irá fazer o programa executar os displays numa tela **ncurses**.

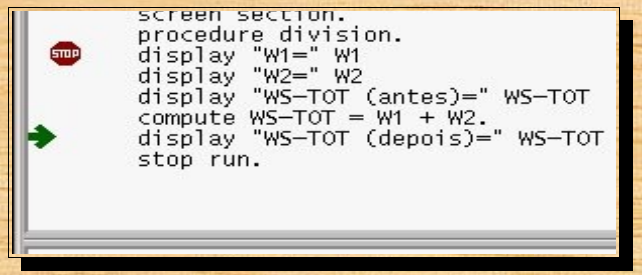
Uma forma de visualizarmos essa saida no DDD é escolher o menu *Program->Run in Execution Window,* deixando o marcador ativado como mostra a figura abaixo.



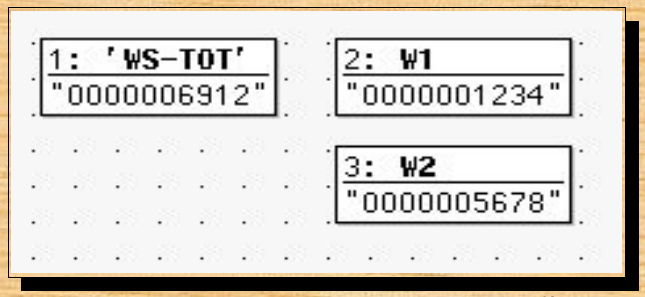
Agora devemos definir um *breakpoint* para que o nosso programa pare logo após ser carregado.

Para adicionarmos o *breakpoint*, com o mouse efetuamos um **duplo-clique** no início da linha onde queremos parar. (podemos adicionar múltiplos *breakpoints, se assim o deesejarmos)*

Em seguida pressionamos o botão **Run**. O programa irá rodar, prando neste local. Pressionando sucessivamente **Next**, teremos o programa sendo executado passo a passo, mostrando uma seta verde à esquerda de cada linha, logo antes do seu código ser executado. Na figura seguinte mostramos o cursor logo antes de executarmos o segundo display de WS-TOT:



Nesse mesmo ponto, teremos a visualização das variáveis contendo o seguinte:



Finalmente, observe que estes displays não acontecem automaticamente. Podemos introduzi-los como já mostrado anteriormente, com o cuidado de por o nome WS-TOT entre apóstrofes, para ser reconhecido como uma variável (e não uma diferença de duas variáveis).

Podem ser criados também displays arbitrários contendo expressões e até mesmo *dumps* de áreas de memória, mas não os trataremos aqui (consulte o manual oficial do DDD).

Uma janela separada mostra os displays (e também ACCEPTs) relativos ao programa, como na figura abaixo.

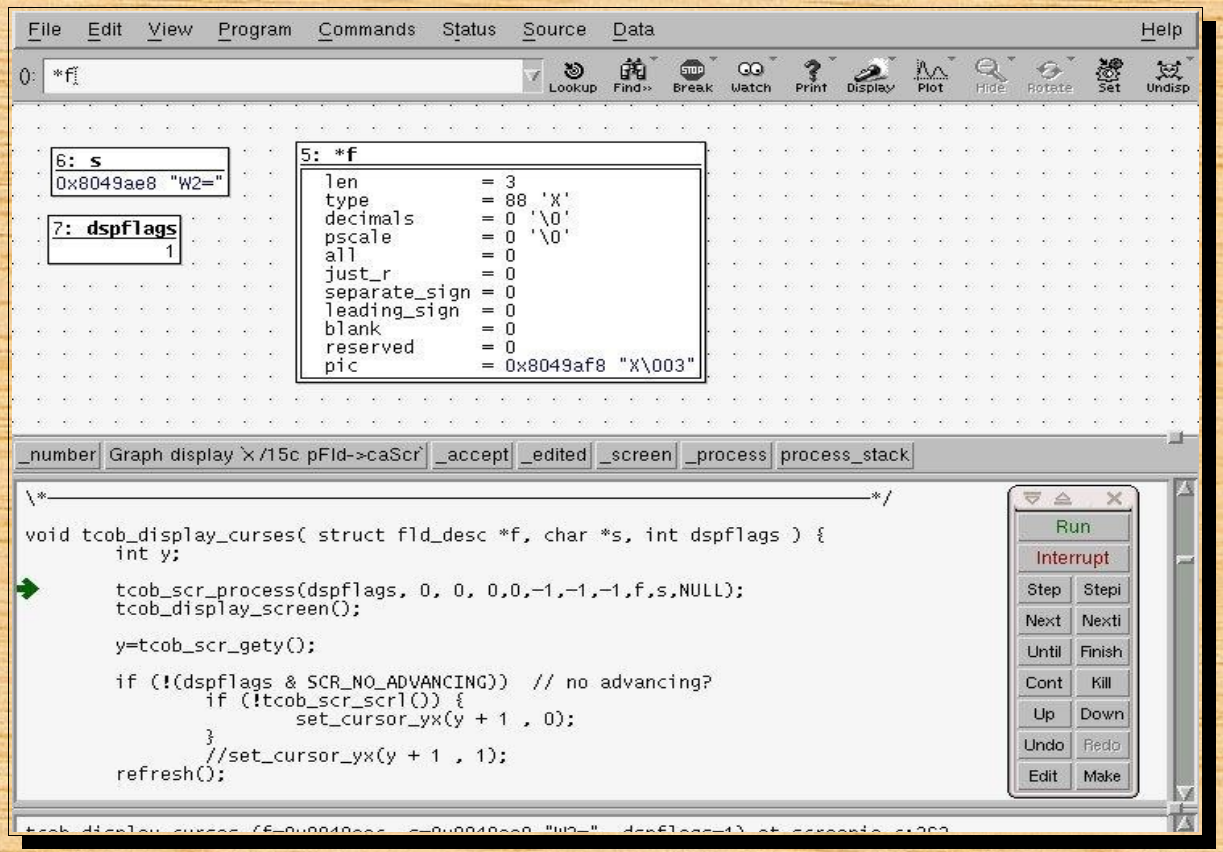
Examinando uma chamada da biblioteca

Suponhamos que queremos verificar o funcionamento de uma função da biblioteca, correspondente a uma instrução cobol. No nosso exemplo, queremos ver como se comporta a instrução

```
DISPLAY "w2="  w2
```

conforme a listagem do programa dado.

Primeiramente, colocamos um *breakpoint* nessa linha e executamos o programa. Em seguida, pressionamos o botão **Step**, que irá nos colocar na seguinte situação:



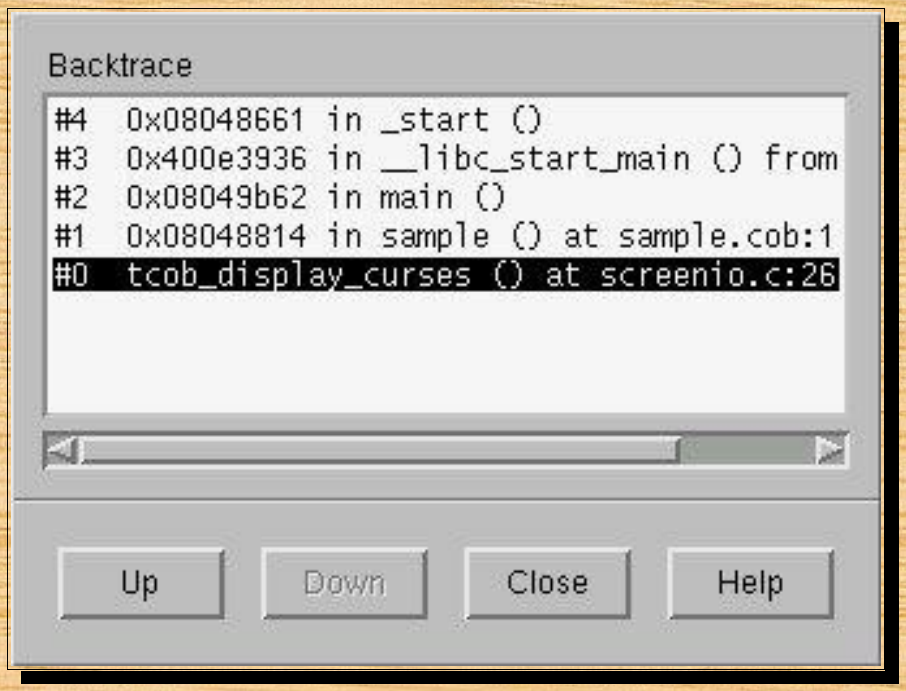
Observe na parte superior (painel de dados) os displays que adicionamos, mostrando os argumentos recebidos pela função, no caso a funçso em C da **libhtcobol,** `tcob_display_curses`.

O display **s** nos mostra o conteúdo da variável, neste caso o literal "W2=". Isso porque uma simles instrução no cobol é geralmente transformada em múltiplas chamadas de funções C. O display **dspflags** indica entre outras, se o comando irá avançar uma linha no terminal após sua conclusão. E o display ***f** mostra propriedades (de uso interno do compilador/runtime) sobre a variável em questão. Por exemplo, podemos ver pelo campo **len** que o tamanho da variável é de 3 caracteres, e vemos também que sua picture (campo **pic**) é "X(03)".

Evidentemente, para um bom aproveitamento dessas informações, o programador deverá ter um conhecimento razoável de C, pelo menos no que se refere à depuração de funções da biblioteca **htcobol**. Para funções escritas no próprio cobol, esse requisito não se aplica.

Examinando a pilha de chamadas

Em qualquer ponto da execução, podemos ver a pilha de chamadas de subrotinas (ou funções) com todos os seus argumentos, invocando o menu *Status->Backtrace*, ou podemos navegar por essa pilha com os botões de controle **Up** e **Down**. No exemplo anterior, se usarmos esse comando do menu, teremos a seguinte janela apresentada:



Na janela temos um cursor na forma de campo reverso (fundo preto com letras brancas) mostrando o nome da função, seguido do **programa fonte:linha**, indicando precisamente onde estamos. Selecionando com o mouse as linhas acima, podemos ver a "função que chamou esta função", ou a "função que chamou...que chamou a função" e teremos o respectivo código-fonte mostrado no painel central do DDD.

Uma forma de encontrarmos rapidamente um problema em um programa, especialmente aplicável na depuração da **libhtcobol**, é deixar o programa rodar livremente, sob o controle do gdb/DDD, e em seguida selecionar a primeira linha que esteja em um programa fonte cobol. No nosso exemplo, seria o *stack frame #1,* que se refere ao fonte **sample.cob,** pois os demais estão na própria biblioteca. Assim isolaríamos um erro associado à execução de determinada instrução cobol.

Poderemos então localizar a linha faltosa, colocar um *breakpoint* nela, e re-executar o programa (com o botso de controle **Run**, respondendo em seguida ao diálogo que aparecerá que o programa seja executado do início).

Webliografia

1. Homepage do programa DDD
   http://www.gnu.org/software/ddd/
2. Homepage do programa gdb
   http://www.gnu.org/software/gdb/gdb.html
3. Um tutorial interessante sobre o gdb
   http://www.dirac.org/linux/gdb/
4. Artigo sobre o DDD na revista LinuxFocus
   http://www.linuxfocus.org/English/January1998/article20.html
5. Revista DDJ - Dr.Dobb's Journal contendo artigo sobre o DDD (necessita assinatrua para acesso ao artigo completo)
   http://www.ddj.com/articles/2001/0103/
6. Site oficial TinyCobol (no SourceForge)
   http://tinycobol.org/
   http://tiny-cobol.sourceforge.net
7. Site TinyCobol do desenvolvedor Rildo Pragana (provisório)
   http://pragana.net/cobol.html
8. TinyCobol Wiki – um wiki sobre o TinyCobol
   http://wiki.tinycobol.org/