

Introdução ao Tiny Cobol

Jonathan Riddell, jr@jridell.org

Uma breve introdução do Tiny Cobol para aqueles que são novos ao Tiny Cobol ou Cobol em geral(tradução de Hudson Reis, <<hudsonreis@softhome.net>>).

1. Este Documento

1.1. Objetivo

Uma breve introdução ao Tiny Cobol para aqueles que são novos ao Tiny Cobol ou Cobol em geral, levando o leitor a olhar através do Cobol, o primeiro e um programa mais complicado usando o make.

1.2. Novas versões deste documento

A última versão desse documento pode ser encontrada no website do Tiny Cobol em <http://tiny-cobol.sourceforge.net>

1.3. Copyrights (em lingua original)

This document is Copyright Jonathan Riddell 2001.

You may copy and distribute it only under the terms of the GNU Free Documentation License, available at <http://www.gnu.org/copyleft/fdl.html>

1.4. Copyrights (em lingua portuguesa)

Este documento é copyright de Jonathan Riddell 2001.

Você pode copiá-lo e distribuí-lo somente sobre os termos da licença de documentação livre GNU, disponível em <http://www.gnu.org/copyleft/fdl.html>

1.5. Agradecimentos

Agradecimentos para a equipe do Tiny Cobol e especialmente para Rildo Pragana, (<http://members.zoom.com/rpragana> (members.zoom.com/rpragana)) nosso líder fiel.

Agradecimentos a Richard Bland, meu instrutor e professor do all things Cobol.

Aplausos também para as pessoas da Debian por incluírem um programa sgmltools que funcione. Ele é muito apreciado.

1.6. Traduções

Não há traduções deste documento ainda(exceto a italiana e brasileira), mas alguns voluntários são bem-vindos.

E sim, este documento é em Inglês britânico. Parênteses são como (), e períodos são alguma coisa que as meninas tem todo mês. Considere-se sortudo... Eu não pus em escocês (<http://scots.jriddell.org>).

2. Introdução ao Cobol e ao Tiny Cobol

2.1. Sobre Cobol

Cobol é uma das últimas linguagens de terceira geração. Ela foi desenvolvida em 1959 (em torno do mesmo tempo como Fortran) para substituir linguagens de processamento específico assembler (linguagens de segunda geração). O primeiro padrão ANSI foi o padrão 68(embora ele tivesse sido usado comumente desde 1961). Padrões posteriores eram o padrão 74 e o padrão 85. A maioria dos programas Cobol usados hoje são de acordo com o padrão 85 (e então faz-se o Tiny Cobol).

Linguagens que competiam com as principais vantagens do Cobol (tais como C) eram pela rapidez e velocidade na leitura/gravação que ele pode entregar, a aritimética de ponto fixo para a contabilidade exata e o inglês como sintaxe para documentação e legibilidade. Isto fê-lo muito útil para trabalhos de grande processamento de dados tais como manter-se informado sobre um milhão de contas bancárias cada noite ou processar pagamentos combinados. A sintaxe em inglês tornaram os programas fáceis para entendimento, então programas comerciais podem ser facilmente modificados em seu tempo de vida assim como as regras de negócio são modificadas.

Cobol é significativamente diferente das linguagens estruturadas de bloco, tais como Pascal, C e descendentes. Ele não tem uma estrutura de blocos e consequentemente sem nenhuma maneira de esconder variáveis; há mais liberdade na escrita do programa; números são próximos a aritimética humana, por exemplo: ponto fixo ou números decimais são geralmente usados ao invés do ponto flutuante; O I/O é orientado por registro, não por classificação; recursividade não é permitido e a lingua própria é muito extensa pois não usa bibliotecas.

Mais informações sobre Cobol podem ser encontradas em um FAQ Cobol em <http://www.cobolreport.com/faqs/cobolfaq.htm>

2.2. Sobre Tiny Cobol

Tradicionalmente a sorte das pessoas que usam software livre não é a sorte das pessoas que usam COBOL. Este é o que o arquivo jargão tem a dizer sobre o assunto:

COBOL /koh'bol/ n. [COMmon Business-Oriented Language] (Sinônimo de maldade.) Uma fraca, verbosa, e uma flácida linguagem usada pelos perfuradores de cartão fazendo coisas monótonas nos mainframes dinossauros. Hackers acreditam que todos

os programadores COBOL são fracos ou moedores de código, e nenhum hacker que tenha amor próprio irá jamais admitir ter aprendido a linguagem. Ela é um nome muito raramente pronunciado sem rituais de aversão ou horror. Uma expressão popular é a famosa observação de Edsger W. Dijkstra que "o uso de COBOL aleija a mente", o seu ensino, deve, portanto, ser considerado uma ofensa criminal".

Crítica realmente bem forte. Mas para opor-se a isto há boas razões por que o Cobol é relevante hoje.

- Há aproximadamente 100 bilhões de linhas de Cobol em uso hoje.
- A grande parte do desenvolvimento é ainda feito em Cobol - há contudo para ser 1 milhão de desenvolvedores ao redor do mundo e 2 bilhões de linhas de Cobol escritas a cada ano.
- Há uma demanda considerável de programadores Cobol para trabalharem com sistemas legados
- Muitos cursos universitários ensinam Cobol
- Você pode discordar do autor do arquivo Jargão.

Assim se Cobol é ainda relevante hoje, significa que será relevante para muitos bons usuários do software livre e significa que serão ambas usadas e importantes para haver a criação de um compilador Cobol livre.

É para isso que o Tiny Cobol veio. Sua criação é encabeçada por Rildo Pragana e foi criado primeiramente para o restritivo ambiente do DOS. O Tiny Cobol é desenvolvido agora por programadores muito mais amigáveis do ambiente do Linux e avança em um longo caminho.

2.3. A Competição

- Cobol for GCC (<http://cobolforgcc.sourceforge.net/>), é um projeto para desenvolver um compilador Cobol que será integrado com a coleção de compiladores GNU. Ele não está ainda em um estado usável.

3. Instalação

3.1. Obtendo Tiny Cobol

O Tiny Cobol atualmente funciona somente com Linux ou FreeBSD em uma máquina i386. Ele está sendo completamente compilado na plataforma win32 com as ferramentas do cygwin, tente olhar os arquivos da mailing list, se você interessar em fazê-lo (Binários também podem estar disponíveis). O mais provável é que não compile, embora você seja bem-vindo a tentar.

Você pode obter o Tiny Cobol do seu website no Sourceforge:
<http://tiny-cobol.sourceforge.net> (<http://tiny-cobol.sourceforge.net/>). Há algumas versões de RPM disponíveis, mas a maioria das pessoas quer obter as versões GZIPadas. O tamanho do arquivo é menor que metade de um megabyte.

Copie este arquivo para algum lugar semelhante a /usr/local/src e descompacte-o (ele irá criar um diretório próprio). Acesse o diretório, tire um parecer do arquivo README e leia completamente o arquivo INSTALL. O arquivo INSTALL irá dizer a você quais bibliotecas você precisa; se você não tem alguma destas bibliotecas então instale-as do CD de sua distribuição ou obtenha-as dos sites sugeridos. Ele irá dizer também que você precisa do GCC instalado, se este não estiver instalado, você não obterá muito, assim, instale-o do CD de sua distribuição ou por um servidor de FTP ou por download da GNU.

3.2. Compilando e Instalando

Para preparar para a compilação, vá para o diretório do Tiny Cobol e digite ./configure (há várias opções para customizar o configure, veja o arquivo INSTALL para detalhes). Então digite "make" para compilar. Você não necessita ser root para isso.

Agora mude de diretório, 'cd test_suite' e digite 'make tests'. Você irá ver um rastro de resultados quando novamente o compilador compilado poderá funcionar com os programas de teste. Nem todos eles irão passar, lembrando que isto é um software alpha, mas a maioria deve.

Agora vá para o diretório original e digite 'make install', que copiará as bibliotecas, o compilador e o pré-processador e alguns outros arquivos assim eles estão prontos para usar. Você precisará de ser root para isto. Os binários estão inseridos em /usr/local/bin, as bibliotecas em /usr/local/lib e todos os outros arquivos em /usr/local/share/htcobol, a não ser que seja especificado com o programa configure.

3.3. Ajuda, se não funcionar!

Se alguma coisa foi errada e você não sabe como corrigi-la, leia este documento outra vez. Tenha certeza que você tem todos as pacotes importantes como o GCC e as bibliotecas necessárias instaladas(você deve ser capaz obtê-los de qualquer CD de distribuição baseado em linux que você usa). Depois leia a webpage do Tiny Cobol em <http://tiny-cobol.sourceforge.net>.

Ainda não está resolvido? vá e cadastre-se na mailing list tiny-cobol-users e pergunte as simpáticas pessoas de lá. Você pode assiná-la em <http://lists.sourceforge.net/mailman/listinfo/tiny-cobol-users> ou por enviando um e-mail para tiny-cobol-users-request@lists.sourceforge.net (<mailto:tiny-cobol-users-request@lists.sourceforge.net>) com subscribe no corpo da mensagem. O endereço para postagem é tiny-cobol-users@lists.sourceforge.net (<mailto:tiny-cobol-users@lists.sourceforge.net>). Você deve também olhar através dos arquivos no caso da sua pergunta ter sido respondida antes. Ela está em <http://lists.sourceforge.net/archives/tiny-cobol-users/>.

4. Primeiro Programa

4.1. Editores

Como a maioria das linguagens de programação, um programa Cobol é exatamente um arquivo texto, que você pode editar em seu editor de textos favorito. Felizmente, para

evitar algumas guerras de religião, ambos vim e Emacs funcionam com Cobol.

4.1.1. Emacs

Não há um modo Cobol que venha como padrão para o GNU Emacs. Ao invés disso, obtenha uma cópia do Cobol.el do projeto Cobol for GCC. Você pode obter o arquivo diretamente do CVS deles em <http://cvs.sourceforge.net/cgi-bin/cvsweb.cgi/gcc/cobol/cobol.el?cvsroot=CobolForGCC>.

Abra o emacs e compile o arquivo lisp com M-X byte-compile-file. Movimente o arquivo Cobol.elc resultante para /usr/share/emacs/20.7/lisp/textmodes/ ou o equivalente para sua instalação. Finalmente adicione este código para o arquivo .emacs ou .gnu-emacs em algumas distribuições) em seu diretório pessoal:

```
(autoload 'cobol-mode "cobol" "COBOL Editing mode" t)
(setq auto-mode-alist
      (append '(("\\.cpp$" . c++-mode)
                ("\\.hpp$" . c++-mode)
                ("\\.lsp$" . lisp-mode)
                ("\\.scm$" . scheme-mode)
                ("\\.pl$" . perl-mode)
                ("\\.cbl$" . cobol-mode)
                ("\\.CBL$" . cobol-mode)
                ("\\.COB$" . cobol-mode)
                ("\\.cob$" . cobol-mode)
                ("\\.CPY$" . cobol-mode)
                ("\\.cpy$" . cobol-mode)
                ) auto-mode-alist))

;; Auto font lock mode
(defvar font-lock-auto-mode-list
  (list 'c-mode 'c++-mode 'c++-c-mode 'emacs-lisp-mode 'lisp-mode
        'perl-mode 'scheme-mode 'scribe-mode 'shell-script-mode 'cobol-mode
        'dired-mode))
```

```
"List of modes to always start in font-lock-mode")

(defvar font-lock-mode-keyword-alist
  '((c++-c-mode . c-font-lock-keywords)
    (perl-mode . perl-font-lock-keywords)
    (cobol-mode . cobol-font-lock-keywords)
    (dired-mode . dired-font-lock-keywords))
  "Associations between modes and keywords")

(add-hook 'cobol-mode-hook
  '(lambda ()
    (set (make-local-variable 'dabbrev-case-fold-
search) t)
    (set (make-local-variable 'dabbrev-case-
replace) t)))
```

Abra o arquivo Cobol no emacs e digite 'M-x cobol-mode' e 'M-x font-lock-mode' para recolher belas cores. Agradecimentos para a equipe do Cobol for GCC por fazer um modo para o emacs.

4.1.2. VIM

Para um guia para usar Cobol no VIM vá para <http://dimensional.com/~sitaram/cobol/>.

4.1.3. Outros

Há muitos outros editores lá fora. Um interessante é o THE, um editor que imita editores de mainframe semelhante ao Xedit. Ele está disponível em <http://www.lightlink.com/hessling/>

4.2. Primeiro Programa

Aqui está um programa Hello World em Cobol:

```
* Hello World Program
* GPL Copyleft Jonathan Riddell 2001
  IDENTIFICATION DIVISION.
  PROGRAM-ID.      hello.
  ENVIRONMENT DIVISION.
  DATA DIVISION.

  PROCEDURE DIVISION.
    DISPLAY "hello ," WITH NO ADVANCING
    DISPLAY "world!"
    STOP RUN.
```

A estrutura do programa é bastante simples, quatro seções, duas que tem conteúdo. PROGRAM-ID é um nome simples para o programa, você pode opcionalmente ter outras informações na IDENTIFICATION DIVISION. A ENVIRONMENT DIVISION pode(opcionalmente agora) conter informações sobre sua configuração, semelhante ao do compilador que você está usando. A DATA DIVISION contém muitas declarações de variáveis que nós podemos precisar. Finalmente a PROCEDURE DIVISION são as atuais instruções do programa.

Você pode notar que há uma série de maiúsculas no código. Usar maiúsculas para palavras reservadas do COBOL mantém o programa mais compatível com outros compiladores e assegura às palavras reservadas uma posição contra as literais e variáveis. Com o Tiny Cobol você é perfeitamente livre para não usar maiúsculas que irão parar o seu Caps Lock desgastando a rapidez.

A outra coisa a notar é o espaçamento. Voltando aos dias dos cartões perfurados, cobol mantém um controle severo das partes onde o código deve estar:

```
column 1-6    : numeração de linhas.
column 7      : área de indicação,
                asterisco para comentar linhas,
                menos para continuação de linhas,
                barra para pular páginas na listagem
```

```
de compilação caso contrário espaços.  
column 8-11 : Margem A, aqui inicia divisões, seções,  
              identificadores de parágrafo e muitas  
              numerações de nível.  
column 12-72 : Margem B, para tudo que não pertencer  
              a margem A  
column 73-80 : área de identificação do programa.
```

O Tiny Cobol não força você a manter este regime mas ele pode manter seus programas mais bem arrumados, ele também para estardalhaços de alguns outros compiladores sobre espaçamento e ele mantém o modo Cobol no emacs felizmente. Tiny Cobol, entretanto, insiste em ter um asterisco que denota um comentário na primeira ou segunda coluna.

Agora compilaremos o programa. Salve o código acima como hello.cob e digite o seguinte comando:

```
htcobol hello
```

Há chances que o htcobol avise que \$COBDIR não esteja setado, enquanto você instalar no diretório default não deve ser um problema mas se você mudar o diretório ou se você recebeu mensagens estranhas de aviso então adicione 'export \$COBDIR=/usr/local/share/htcobol' ou similar para seu arquivo .bashrc ou .profile. Ele irá dizer isso também 'Processing compiler parameters', isto é uma boa coisa.

O Htcobol irá produzir os arquivos hello.lis e hello.s. hello.lis é a conversão do código Cobol em alguma coisa mais amigável para o compilador. hello.s é um código assembly, à leitura humana equivalente ao código de máquina.

Se seu shell não pode encontrar um programa chamado htcobol, primeiramente cheque sua variável \$PATH apontando para /usr/local/bin/htcobol ou onde quer que você instalou-o (digite 'echo \$PATH'). Se ele existir, então, cheque duplamente se ele está configurado, compilado e instalado corretamente.

Se você se embaraçou com o código abaixo, você vai notar que o Tiny Cobol não é ainda muito bom para reportar erros. De fato tudo que você receberá será uma "Falha de

Segmentação". Cheque duplamente se você soletrou tudo corretamente, e se você não esqueceu de nenhuma pontuação e os asteriscos dos comentários da primeira coluna.

O próximo passo é compilar o arquivo assembly hello.s para código de máquina. O comando para isto é:

```
as -o hello.o hello.s
```

o AS é parte da coleção de compiladores GNU, GCC, e se você não o tem instalado, você tem uma distribuição de fato muito esquisita (dado que nós já compilamos o Tiny Cobol hoje). Cheque a documentação da sua distribuição para instalá-lo.

hello.o é um arquivo objeto, mas não é um executável que funciona por que ele não está sendo linkado com as bibliotecas necessárias. Este é o comando final:

```
gcc -o hello hello.o -lhtcobol -lm
```

Se você achou isso difícil, você provavelmente não tem tido nenhum problema com isso, mas se o gcc não encontrar as bibliotecas libhtcobol ou libm então você pode ter que atualizar a base de dados das bibliotecas com o comando ldconfig.

Finalmente há um programa funcional que pode ser chamado com:

```
./hello
```

E você deve ver na tela:

```
Hello, world!
```

Que será praticamente previsível. Se você olhar o código do programa, ele deve ser óbvio que DISPLAY mostra e que WITH NO ADVANCING mostra(sem quebrar a linha).

5. Programação Adicional

5.1. Outro Programa

Aqui está um programa um pouco mais complicado, digite-o em um arquivo e salve-o como addition.cob.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.      addition.  
ENVIRONMENT DIVISION.  
  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
77  myvar PIC 999 .  
77  a  PIC 99V999 VALUE 1.777 .  
77  b  PIC 99V99  VALUE ZERO .  
77  c  PIC 9V9    VALUE 5.5 .  
77  d  PIC 9v9    VALUE 5.5 .  
  
PROCEDURE DIVISION.  
    DISPLAY "Por favor insira um número"  
    ACCEPT myvar  
    DISPLAY "O número é ", myvar  
    ADD a TO b ROUNDED  
    DISPLAY "b is ", b  
    ADD c TO d  
    DISPLAY "d is ", d  
    STOP RUN.
```

Você pode compilar este com a mesma rotina, a qual você usou com hello.cob, se você encontrou alguma falha de segmentação, cheque duplamente seu programa antes de se queixar.

Se você é novo com o Cobol, há um ponto para se tomar aqui. Primeiramente nós temos variáveis declaradas na **WORKING-STORAGE SECTION** na **DATA**

DIVISION. O Cobol requer que todas as variáveis sejam declaradas desta maneira. As outras seções da DATA DIVISION lidam com arquivos e linkage.

As variáveis declaradas aqui iniciam com '77'. Isso significa que a variável está no nível 77: ele não tem estrutura interna e num relacionamento definido com nenhuma outra variável. Ele não é um parte de um registro(ou qual as estruturas de chamadas C). O Nome da primeira variável é myvar, nome das variáveis podem ser letras maiúsculas ou minúsculas, mas com o TinyCobol você deve ser consistente em todas as partes do seu programa.

A cláusula PIC especifica o tipo da variável. Usando "9s", significa que ela é uma variável decimal e usando 3"9s", cria uma variável que recebe os números 0 a 999. Você pode ter também um S antes do "9s" para criar uma variável com sinal. Um 'V' na declaração PIC significa uma variável com um ponto decimal assumido na posição V. Para variáveis grandes, você pode escrever 9(15), que é uma variável que tem 15 espaços. Usando "As" ao invés de "9s", criamos uma variável alfabética e usando "Xs" para criar uma variável alfanumérica. Finalmente um VALUE inicial pode ser indicado para a variável.

Programadores Cobol inexperientes podem estar preocupados como o final de um parâmetro é marcado. O final dos blocos são marcados com uma parada completa, mas os parâmetros não são marcados definitivamente. Ao invés disso o compilador Cobol reconhece quando um novo parâmetro se inicia ao notar um grande número de verbos Cobol. O final das linhas não é importante no Cobol. Outra coisa para notar no código acima é as vírgulas adicionadas. Vírgulas são completamente opcionais, você pode inserí-las no lugar que você preferir ou em lugar nenhum que não fará diferença. Use-as com moderação e elas farão seu programa ficar mais caprichado.

Quando executar o programa, ele deve perguntar a você por um número, então irá mostrar este número ao longo de dois outros cálculos que ele acabou de realizar. O número dado não pode ser maior que 3 espaços alocados para ele, senão somente os três caracteres finais que são dados, se ele são dados menos que 3 caracteres, então mostrará zeros primeiro. Caracteres não numéricos irão mostrar algum código de debug.

A adição das variáveis a e b devem mostrar um bom funcionamento, embora você note que a resposta é arredondada, como nós pedimos. Adicionando c e d não fará coisa alguma, desde que a resposta seja muito grande para adequar-se na variável d. Tente

mudar c para, digo, 1.5 e tudo irá funcionar muito bem.

5.2. Make

Por agora você pode obter um pequeno aborrecimento com as rotinas htcobol, as e gcc. Bom há um caminho melhor, make. O make exige um arquivo chamado Makefile e usa as instruções para funcionar que necessitamos compilar em um simples passo. Ele também checa se o arquivo necessita de compilação antes de ele fazer qualquer coisa, se nenhuma das dependências dos programas foram modificados desde a última compilação, nenhum sendo feito.

As regras do makefile são da seguinte forma:

```
alvo:dependências
    comando
```

O alvo(que deve iniciar na primeira coluna) é o arquivo que você está tentando obter. As dependências são os arquivos dos quais ele depende. O comando(que deve ser a segunda linha e precedida por um tab) é o comando que você normalmente, digita no shell para fazer seu arquivo. Aqui está um makefile para addition:

```
#um simples makefile para um programa cobol

addition:addition.o
gcc -o addition addition.o -lhtcobol -lm

addition.o:addition.s
as -o addition addition.s

addition.s:addition.cob
htcobol addition
```

Salve o como makefile neste mesmo diretório como addition.cob e digite make. Ele deve compilar, assemblar e linkar para você.

Por default, make irá compilar a primeira regra no arquivo. Este é para criar o binário addition, mas este depende do addition.o que para fazer addition.o depende do addition.s. Então este é o primeiro comando que o make executa. Agora ele cria addition.o e finalmente addition.o está pronto para criar addition. Se você tentar digitar make novamente, não irá fazer coisa alguma, por que nenhum dos arquivos foi modificado então não há necessidade (o make olha o arquivo timestamp para determinar isso), mude addition.cob bem de leve e execute make e todo o processo executará novamente.

5.3. Make Avançado

Mas o makefile anterior tem um problema: e se você quiser adaptá-lo para trabalhar com um outro programa? há bastante mudanças em addition para fazê-lo e com um programa mais complicado que irá pedir tempos. Felizmente make tem muitas complexidades na manga. Aqui está um makefile avançado:

```
#makefile para um programa cobol composto de um único arquivo.
```

```
PROGRAM := addition
LIBS := -lhtcobol -lm
OBJECTS := $(PROGRAM).o
```

```
$(PROGRAM):$(OBJECTS)
gcc -o $(PROGRAM) $(OBJECTS) $(LIBS)
```

```
#regras para os mesmos arquivos terminados em .o
#depende do arquivo com mesmo prefixo com .s
# $@ é o alvo
%.o:%.s
as -o $@ $<
```

```
%.s:%.cob
htcobol $(PROGRAM)
```

O makefile inicia com algumas variáveis. PROGRAM é o nome do programa que nós estamos criando. Você pode ver se ele está sendo usado na primeira regra (precedido por um sinal de dólar delimitado com parênteses). LIBS é justamente as bibliotecas que nós precisamos para compilar o programa. OBJECTS é alguns arquivos objetos necessários, neste caso é exatamente um, e que possui o mesmo nome com um .o no final, que é justamente como ele é declarado.

A segunda regra combina os alvos terminados em .o (%.o) e ele requer o mesmo arquivo com um sufixo .s (%.s como uma dependência). A variável automática \$@ iguala o nome do alvo e \$< iguala a primeira dependência. Por agora você deve estar também trabalhando o seu caminho até o alvo final.

Se você quiser adaptar este makefile para outro programa COBOL, tudo que você tem que fazer é mudar o valor da variável \$PROGRAM.

6. O Final

Isto é tudo por agora. Deve possivelmente haver um outro capítulo aqui com um programa que usa mais de um arquivo e o pré-processador, mas isto é além do meu conhecimento por hoje. Talvez também alguns links para recursos do COBOL.

Jonathan Riddell, jr@jridell.org

