INSTRUMENTADOR DE PROGRAMAS "C"

Marcelo S. Pimenta Verner Heidrich Ana M. A. Price

Universidade Federal do Rio Grande do Sul
Porto Alegre - Brasil

Resumo

Descreve-se um instrumentador para a linguagem "C", que possibilita o uso de comandos especiais visando facilitar a depuracao de programas. A ferramenta foi implementada com o objetivo de proporcionar ao programador um recurso que permita o melhor acompanhamento da execucao de programas, facilitando a localizacao de erros e identificacao de suas causas.

Os comandos de depuracao são especificados como comentários no programa fonte, de modo que, concluída a fase de teste, o programa pode ser imediatamente compilado e colocado em operacao, dispensando a remocao daqueles comandos.

O instrumentador é um pré-processador, que recebendo um programa em "C" instrumentado, analisa declaracoes-fonte e comandos de depuracao, e traduz os últimos em instrucoes "C", entregando ao compilador o programa original com as adicoes para depuracao.

Sao apresentados, tambem, detalhes da implementacao do instrumentador proposto.

1. Introducao

O teste de programas pode absorver acima de 50% do custo de desenvolvimento de sistemas automatizados[Tai 80]. Uma das razoes do alto custo da fase de testes é o caráter imprevisível da depuracao. Depuracao é a atividade que inicia com a descoberta de erros no programa e compreende, basicamente, a localizacao e correcao destes erros. A identificacao de um erro pode demorar minutos, horas e até dias. Existem dados que mostram que a maioria dos erros de programas sao encontrados e corrigidos em aproximadamente uma hora. Entretanto, existe um conjunto de erros (em torno de 20%) onde cada elemento toma da ordem de um dia para ser identificado e corrigido [Johnson 83].

A depuracao de um programa é uma atividade difícil e dispendiosa, particularmente, por dois motivos :

- porque o programador desenvolve mentalmente um modelo de comportamento do programa que nao corresponde exatamente ao seu funcionamento real; e
- pela falta de ferramentas adequadas a este tipo de atividade.

A ferramenta ideal para depuracao deve ser interativa, facil de usar e oferecer facilidades tais como [Ward 86]:

- possibilidade de executar o programa passo-a-passo, mostrando os comandos sendo executados e os valores das variáveis envolvidas;
- suspender a execucao do programa em pontos especificos ou na ocorrencia de eventos pré-determinados;
- permitir ao usuario acesso ao codigo fonte para modificacao de instrucoes e alteracao de dados;
- permitir compilacao incremental e registrar os comandos de depuracao executados de modo a possibilitar a repeticao do processo.

Entretanto, a implementacao de tais facilidades nao constitui um procedimento trivial e pode sofrer sérias restricoes quando desenvolvida para microcomputadores,pois exige espaco de memória apreciável, considerando que devem permanecer em memória varias estruturas geradas durante a compilacao do programa juntamente com os interpretadores para o programa fonte e para os comandos de depuracao requeridos pelo usuario.

Uma tecnica de depuracao mais acessivel é a instrumentacao de programas fonte antes da compilacao. A instrumentacao de programa permite determinar certas propriedades (por exemplo, identificacao de caminhos inacessiveis, frequencia de execucao de instrucoes, anomalias no fluxo de dados, etc) que possivelmente só seriam identificadas a um custo elevado por outras técnicas de verificacao.

As formas mais frequentes de instrumentacao de programas incluem :

- insercao de contadores para medir niveis de profundidade de teste;
- adicao de assercoes para verificar a validade de valores de variaveis; e
- exibicao de valores de variaveis em determinados pontos do programa.

Entretanto, muitas das funcoes pertinentes a depuradores simbolicos sao passiveis de serem implementadas atraves de instrumentacao, como por exemplo execucao passo-a-passo, exibicao dos comandos sendo executados, suspensao da execucao do programa em pontos predeterminados e entrada de valores para variáveis durante a execucao do programa.

A maioria das implementacoes de linguagens existentes no mercado nao possui ambiente de depuracao simbolica. Fica a cargo dos proprios programadores o desenvolvimento de procedimentos de depuracao, que acabam desviando sua atencao do problema a que se propunham resolver. Um ambiente desejável seria aquele que tornaria os detalhes de depuracao transparentes ao programador. A necessidade de uma ferramenta de depuracao com tal requisito é que deu origem a este trabalho.

O protótipo descrito neste artigo resultou de um trabalho proposto na Disciplina de Compiladores ministrada no Bacharelado em Ciencias da Computacao da UFRGS. Os projetos propostos pela Disciplina visam a implementacao de processadores que envolvam tecnicas de compilacao. Os estudantes têm aproximadamente tres meses para a realizacao do trabalho, sendo este periodo partilhado com o desenvolvimento de projetos propostos em outras disciplinas. O prototipo em questao foi desenvolvido e testado, em tempo hábil, por dois dos autores deste relatório.

O assunto do trabalho foi escolhido por condizer com o tema apresentado na Disciplina, o prazo cabível de desenvolvimento e por vir ao encontro da comunidade usuária local, como instrumento de aplicacao prática imediata. Os implementadores pretendem extende-lo com novas funcoes e aplicá-lo no desenvolvimento de programas na indústria onde trabalham.

A linguagem "C" foi escolhida como linguagem hospedeira devido a :

- suas características de portabilidade, o que faz com que o instrumentador possa migrar para vários equipamentos, e
- expressiva comunidade de usuarios (pesquisadores, estudantes) atuantes na Universidade.

2. Descricao dos Comandos de Depuracao

Os comandos de depuracao, aqui descritos, foram projetados para obterem informacoes sobre o programa, através de uma linguagem simples e facilmente memorizável pelo programador.

A fim de que o programa "C" instrumentado nao precise ser modificado para ser submetido ao compilador, os comandos de depuracao devem ser escritos na forma de comentarios, isto é, encerrados entre os pares '/*' e '*/'. Assim, qualquer erro na escrita ou formulacao dos comandos de depuracao fará com que sejam interpretados como comentários.

A sintaxe, a semantica e a interpretacao dos comandos é apresentada a seguir.

⟨comando_inicio_depuracao⟩ ::= /* DEPURAR */

O comando DEPURAR delimita o início do segmento de programa a ser depurado. Se tal comando não estiver presente no programa, os comandos de depuracao posteriores serao desprezados. Dois ou mais comandos DEPURAR consecutivos têm o valor de um único.

⟨comando_exibicao⟩ ::= /* EXIBIR ⟨lista_de_variaveis⟩ */

Mostra, no dispositivo de exibicao,o valor corrente das variaveis indicadas em ⟨lista_de_variaveis⟩, no formato "⟨var⟩ = ⟨valor⟩".

As variaveis podem ser quaisquer variaveis validas em "C", que tenham sido declaradas previamente. Se nao forem reconhecidas é gerada, a tempo de pré-processamento, uma mensagem de alerta.

⟨comando_assercao⟩ ::= /* VERIFICAR ⟨expressao⟩ */

Realiza o teste da assercao indicada quando da execucao daquele segmento do programa. Se a assercao for violada, isto é, se a expressao avaliada resultar em FALSO, será exibida a mensagem "ASSERCAO ⟨expressao⟩ VIOLADA".

⟨expressao⟩ deve ser uma expressao valida na linguagem "C", impondo-se as seguintes restricoes :

(1) expressoes nao podem conter variaveis do tipo 'structure', 'union' e 'typedef',

(2) expressoes nao devem modificar o valor das variaveis envolvidas, por exemplo :

++x, x= a--, --x,

(3) expressoes nao podem conter : conversao de tipo, a funcao pré-definida 'sizeof' e expressoes condicionais envolvendo o operador ternario '?'.

(4) lista de expressoes nao sao permitidas, por exemplo :

x || 4, w & 0x2f.

Expressoes do tipo (2) nao podem aparecer em comandos de depuracao, pois quando avaliadas, modificam o valor das variaveis envolvidas. Tais expressoes serao avaliadas se o instrumentador for ativado para gerar o codigo correspondente ao comando VERIFICAR. Caso contrario, elas serao vistas como comentarios. Desta maneira, se o programa for instrumentado, as variaveis envolvidas terao um valor, senao, terao outro. O que deseja-se evitar com a restricao é o reflexo indesejado da depuracao na execucao do programa. As demais restricoes apenas reduzem a dificuldade de implementacao do instrumentador.

A opcao de aceitar a mesma sintaxe das expressoes da linguagem "C" em <expressao> na assercao facilita ao programador, pois ele nao necessita memorizar novos formatos, operadores e regras de sintaxe.

<comando_rastreamento> ::= /* RASTREAR */

O comando RASTREAR delimita inicio do segmento de programa que será rastreado. Rastrear significa exibir a imagem do comando <sequencial> sendo executado. Isto é, a partir do ponto marcado pelo <comando_rastreamento> todo Comando Significativo executado é mostrado no dispositivo de exibicao. Comando Significativo é todo aquele que nao é Comando de Controle de Fluxo de Programa (CCFP). Em "C", os CCFP sao : 'if', 'if - else', 'switch - case - default', 'while', 'do - while' e 'for'. Os comandos que compoem os campos dos CCFP sao Comandos Significativos se nao forem CCFP.

Dado o segmento de programa abaixo, somente os comandos especificados nas linhas 3, 4, 6, 8 e 10 sao passiveis de exibicao quando rastreados :

```
1.   while (x) {
2.         if (w) {
3.              c++;
4.              break;
5.              }
6.         a = proc(x,c);
8.         b = a >> w & 0x80;
9.         }
10.  return (a);
```

<comando_fim_rastreamento> ::= /* NAO RASTREAR */

O comando NAO RASTREAR delimita o fim do segmento de programa que estava sendo rastreado. Se omitido, é assumido como fim do segmento o final do programa. Entre pares de comandos de rastreamento, podem ser especificados os comandos EXIBIR, VERIFICAR, PARAR E PASSO A PASSO.

<comando_parada> ::= /* PARAR */

O comando PARAR define um ponto de parada ("breakpoint") no programa. No momento de sua execucao, é solicitado do programador uma diretiva dentre as possíveis : continuar execucao sequencial ou encerrar o programa. Isto é feito quando da emissao da mensagem "PARADA : Continuar / Encerrar (C/E) ? _".

<comando_inicio_passo_a_passo> ::= /* PASSO A PASSO */

O comando PASSO A PASSO delimita o início do segmento de programa no qual a execucao de cada Comando Significativo é condicionada à intervencao do programador, através de mensagens semelhantes às do comando PARAR.

<comando_fim_passo_a_passo> ::= /* FIM PASSO A PASSO */

O comando FIM PASSO A PASSO delimita o final do segmento iniciado com PASSO A PASSO. Se omitido, é assumido o final do programa.

<comando_fim_depuracao> ::= /* NAO DEPURAR */

O comando NAO DEPURAR delimita o final do segmento de depuracao. A partir do ponto demarcado por este comando, qualquer comando de depuracao posterior será considerado comentario. Se omitido, é assumido o final do programa.

Podem existir vários pares de comandos de início e fim de depuracao num mesmo programa fonte.

3. Implementacao

O instrumentador foi implementado como um pré-processador que analisa codigo fonte "C" contendo comandos de depuracao e gera um programa "C" instrumentado. Este constitui-se em um programa com modificacoes e adicoes feitas pelo pré-processador para implementar as facilidades dos comandos de depuracao.

Esta opcao nos pareceu mais imediata pelos seguintes motivos : a implementacao era viável no prazo estabelecido, considerando-se ser um trabalho de cunho academico; os comandos poderiam ser implementados independentemente um dos outros, de modo que se poderia implementar uma versao básica e, depois, incrementá-la com novos comandos. Além disso, se implementado desta forma, o instrumentador se tornaria facilmente adaptável a outras linguagens de programacao, bem como outros microcomputadores devido à portabilidade da linguagem "C" e ao método tabular de análise implementado pelo instrumentador.

O pré-processador implementado foi desenvolvido em linguagem "C", sob o Sistema Operacional MS-DOS em microcomputador PC-compatível nacional. "C" foi a linguagem escolhida devido a suas características de modularidade, portabilidade e sua expressiva utilizacao pela comunidade academica local no desenvolvimento de software de suporte.

3.1. Estrutura do Sistema

O pré-processador é constituído pelos seguintes módulos
- Analisador Léxico,
- Analisador Sintático e
- Gerador de Comandos,

os quais são descritos a seguir.

Analisador Léxico

O Analisador Léxico reconhece basicamente seis tipos de tokens: identificadores, palavra-chave, constantes, operadores, separadores e cadeias de caracteres. Interage com o Analisador Sintático, cada vez que esse o solicitar, devolvendo o próximo 'token' válido ou uma indicacao de erro.

Análisador Sintático

O Analisador Sintático trabalha em duas etapas :
- Pré-análise, que identifica as figuras sintáticas chave (que interessam à depuracao) e só entao desencadeia o processo de análise sintática. Evita-se assim a análise completa do código fonte, que é desnecessária segundo os objetivos do instrumentador.
- Análise, implementada através de um analisador 'top-down' nao-recursivo, baseado na técnica do grafo sintático de Wirth com funcionamento determinístico [Setzer 83], adequado aos objetivos do instrumentador.

Gerador de Comandos

O Gerador de Comandos tem por funcao acrescer ao programa fonte trechos de código que implementam as tarefas de depuracao especificadas pelo programador. O Gerador realiza sua funcao a partir de esqueletos de comandos, que são completados de acordo com as características específicas de cada programa (nomes de variáveis, funcoes declaradas e outros).

3.2. Funcionamento do Instrumentador

Uma linha do fonte é lida a cada vez, sendo percorrida pelo Analisador Léxico, que a separa em elementos sintáticos ('tokens'). Como apenas os elementos, declaracao de variáveis e comandos de depuracao do programa fonte, precisam ser analisados para a depuracao, deve haver um processo de identificacao destes elementos antes da análise, o qual é feito na fase de pré-análise do Analisador Sintático.

Sao identificadas apenas as figuras sintáticas chaves de estruturas que interessam à depuracao para entao desencadear o processo de análise sintática de cada estrutura. Se a estrutura é significativa para a depuracao, o Analisador Sintático é acionado.

Quando uma declaracao de variável, ou de funcao, é reconhecida, sao feitas inclusoes e alteracoes na tabela de símbolos.

Quando é reconhecido um comando de depuracao, o gerador de comandos é chamado.

Por fim, as linhas lidas, alteradas ou nao pelo gerador de comandos sao gravadas no arquivo destino da instrumentacao, gerando o programa instrumentado que poderá ser submetido a um processo compilativo normal. Na figura 1 é apresentado o diagrama funcional do sistema.

3.3. Dispositivo de Exibicao

O programador pode especificar qual o dispositivo que pretende utilizar para a exibicao dos resultados da depuracao. Um dispositivo pode ser um arquivo do tipo texto ou um arquivo especial do sistema operacional, como por exemplo, a impressora ou o vídeo. Isto é indicado na chamada do instrumentador, passando-se como parâmetro o nome do arquivo a conter os resultados da depuracao.

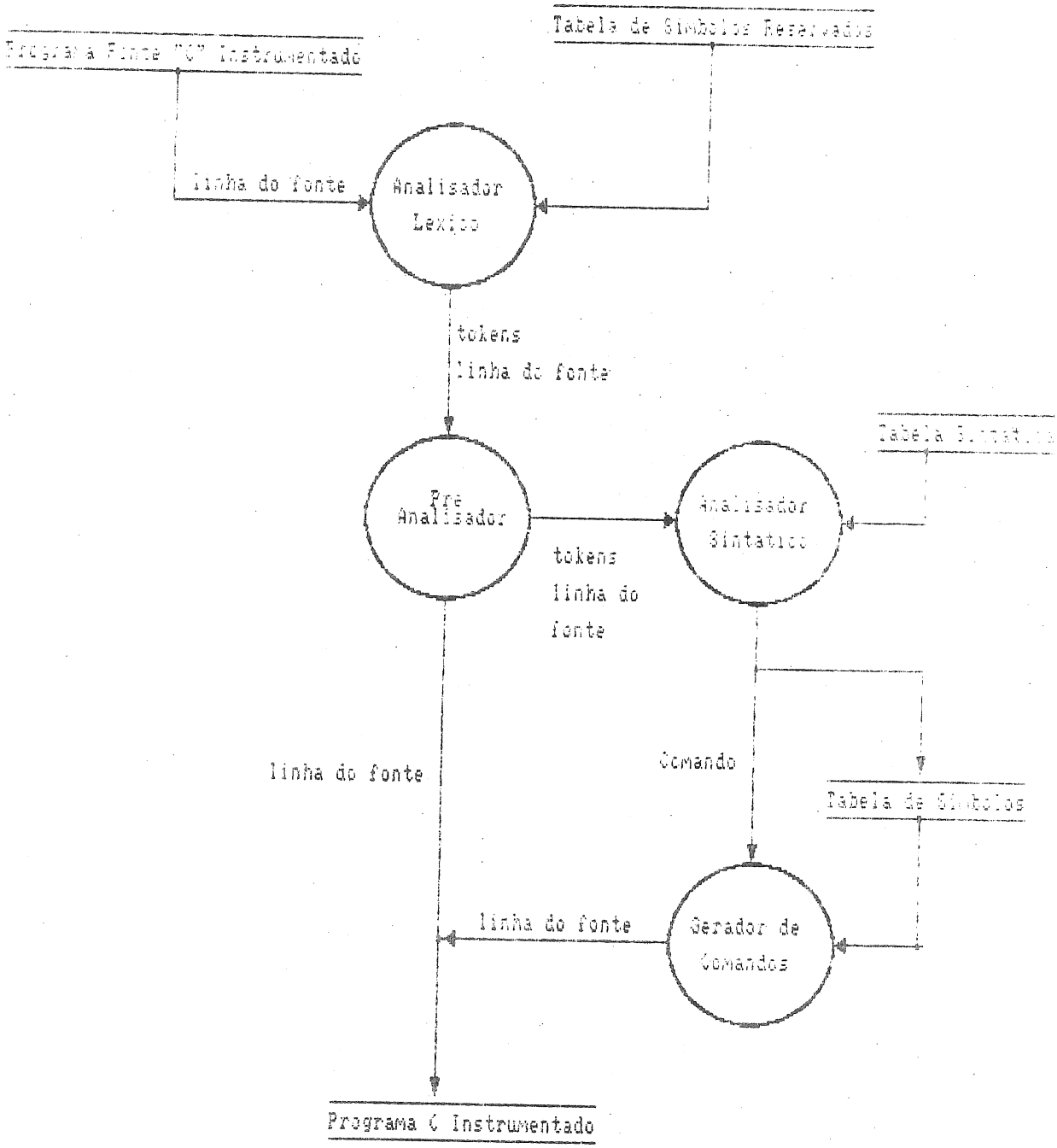

Figura 1. Diagrama Funcional do Sistema.

De forma geral, a execucao do instrumentador é efetuada como segue :

- para disparar a instrumentacao sobre o programa gerarq.c, redirecionar o dispositivo das mensagens de instrumentacao para o arquivo depura.err e gerar os futuros resultados da depuracao no arquivo depura.sai :

```
A) depura -igera_arq.c -ogera_arq.dep -edepura.sai > depura.err
```

- para verificar possíveis mensagens de erros da instrumentacao :

```
A) type depura.err
```

- para compilar o programa instrumentado :

```
A) compila gera_arq.dep
```

- para executar o programa :

```
A) gera_arq
```

- para verificar o resultado da depuracao :

```
A) type depura.sai
```

4. Conclusoes

Instrumentacao de programas é um meio economico de obter informacoes adicionais sobre a execucao de um programa como um produto suplementar do teste de programas. É razoavelmente simples de ser implementada e fácil de ser usada por programadores pois, ao contrario de outras tecnicas de verificacao e validacao de programas, nao requer conhecimento profundo de matematica ou logica formal [Huang 80].

Foi apresentada a descricao de um instrumentador para programas em "C" visando facilidades para depuracao. Este trabalho foi desenvolvido com o objetivo de propiciar ao programador comandos especiais de depuracao, de fácil aprendizagem, que permitam obter informacoes sobre a execucao de seus programas.

Os comandos de depuracao sao especificados como comentários, de modo que, finda a fase de depuracao, o programa em teste é compilado e liberado para operacao sem exigir quaisquer alteracoes.

O instrumentador foi implementado como um pré-processador que analisa declaracoes de dados , funcoes do programa fonte e comandos de depuracao entre delimitadores de comentários, e traduz estes últimos em codigo fonte "C".

Os comandos de depuracao implementados incluem marcadores de início e fim de depuracao, exibicao de valores de variáveis, verificacao de assercoes, delimitadores de início e fim de execucao passo-a-passo, e parada de programa. A implementacao, tal como foi desenvolvida, permite, com razoável facilidade, a insercao de novos comandos de depuracao e, também, a adaptacao do instrumentador a outras linguagens de programacao (isto deve-se ao fato de que a análise sintática foi implementada através de um método tabular).

Este trabalho resultou de um projeto proposto na Disciplina de Compiladores do Bacharelado em Ciências de Computacao da UFRGS. Como programadores experientes em "C", os implementadores acreditam na praticidade da ferramenta proposta.

Referências Bibliográficas.

A. Aho & J. Ullman, Principles of Compiler Design, Adison Wesley, 1977.

D. Gries, Compiler Construction for Digital Computers. John Wiley, New York, 1971.

[Huang 80]
J. C. Huang, Program Instrumentation : A Tool for Software Testing : 2, in INFOTECH State of The Art Report, U.K., 1979.

[Johnson 83]
M.S. Johnson(ed.),Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on High-Level Debugging,
ACM Software Engineering Notes,Vol.8,No.4,Aug 1983.

B. Kernighan & D. Ritchie, The C Programming Language, Prentice-Hall, 1978.

[Setzer 83]
Setzer, Valdemar W., A Construcao de um Compilador, Editora Campus, 1983.

L. G. Stucki & G. L. Foshee, New Assertion Concepts for Selfmetric Software Validation, Proc 1975 International Conference on Reliable Software (April 75) in SIGPLAN Notices Vol.10,No.6,Jun 1975.

[Tai 80]
K. Tai, Program Test Complexity and Test Criteria, IEEE Transactions on Software Engineering, Vol.6, No. 6, 1980.

[Ward 86]
R. Ward, Debugging C, Indianapolis, QUE Corporation, c 1986.

R. T. Yeh, Current Trends im Program Methodology vol II: Program Validation, Prentice Hall, Englewoods Cliffs, NJ , 1977.