# UM DEPURADOR MULTILINGUAGEM EM UM AMBIENTE DE DESENVOLVIMENTO

Marcelo Soares Pimenta *,+
Roberto Tom Price **,+

SUMARIO :

O artigo apresenta o depurador multilinguagem para o Ambiente de Desenvolvimento de Software (ADS) em construção na UFRGS. São propostos requisitos que, na percepção dos autores, devem ser atendidos por um depurador. É descrita a integração do depurador com os demais componentes do ADS (editor dirigido pela sintaxe e interpretador) bem como a interface com o usuário e a sintaxe dos comandos de depuração.

ABSTRACT :

This article presents the multilanguage debugger of the Software Development Environment (ADS) project currently beying developped at UFRGS. The requirements that must be met by a debugger, in the author's point of view, are outlined. The integration of the debugger with the other components of ADS (syntax-driven editor and interpreter) as well as the user interface and the syntax of the debugging commands are described.

Este projeto foi desenvolvido com o apoio financeiro de CNPq, FINEP e SID-Informática.

* Mestrando em Ciência da Computação, UFRGS-CPGCC.
Bacharel em Ciência da Computação, UFRGS 87;
** Eng, MsC, DPhil;
Professor pesquisador do CPGCC/DI/UFRGS.
+ Curso de Pós Graduação em Ciências da Computação, UFRGS, Caixa Postal 1501, Porto Alegre, RS.

## 1. INTRODUÇÃO

A presença de erros em programas pode ser encarada como um fenômeno fundamental e o programa livre de erros como um conceito abstrato, irreal.

Na realidade, não há regras a seguir para garantir que um programa está livre de erros. É verdade que muitos erros podem ser achados pelo compilador ou outras ferramentas estáticas como um verificador de interfaces ou um manipulador de referências cruzadas. No entanto, alguns erros críticos necessitam de outras técnicas e ferramentas diferentes, mais adequadas para a pesquisa e remoção de erros. Estas técnicas propiciam um diálogo entre o programador e o computador cuja intenção é obter informações sobre o comportamento do programa. Durante este diálogo, convencionalmente chamado depuração, o programa é iterativamente corrigido e melhorado. As ferramentas que auxiliam a realização das tarefas de depuração denominam-se depuradores.

Teste e depuração são atividades distintas mas intimamente relacionadas uma vez que o resultado dos testes constitui a entrada para a depuração. A integração das duas constitui o que se denomina de Fase de Teste e Depuração (FTD).

[Ramamoorthy 77] afirma que a FTD atinge 15% do custo total de produção de software e, segundo [Boehm 73], ela toma aproximadamente 40% do esforço total de desenvolvimento. Para sistemas que exigem alta confiabilidade (por exemplo, diagnóstico médico, tráfego aéreo), o custo da FTD pode resultar em 3 a 5 vezes o custo de todas as outras fases do projeto juntas [Pressman 82].

Assim, teste e depuração de programas assumem um papel fundamental no ciclo de vida do software. No entanto, apesar de um grande esforço estar sendo dedicado a metodologias, linguagens e ferramentas para a escrita de programas com poucos erros, apenas uma pequena parcela deste esforço é destinada ao desenvolvimento de facilidades para localização e correção de erros [Wilkes 76].

[Houghton 82], em um conjunto representativo de ferramentas de desenvolvimento de software (quase 400 catalogadas), identificou apenas 3% que se classificavam como ferramentas de teste e depuração de programas.

Além disto, os poucos depuradores existentes não são utilizados pois:

a. não estão disponíveis para um determinado equipamento e/ou linguagem ou não podem permanecer na memória junto com os programas a serem depurados;

b. não são agradáveis de usar possuindo comandos complexos e dificilmente memorizáveis e uma interface com o usuário não amigável;

c. são freqüentemente projetados muito após o sistema operacional e as linguagens disponíveis terem sido implementados. Isto geralmente resulta em um depurador não integrado ao sistema;

d. não usam a mesma sintaxe da linguagem de programação, tornando-se mais difícil aprender a usá-los;

e. referenciam código objeto (endereços e instruções) enquanto o nível de compreensão do usuário é o código fonte (símbolos e comandos);

f. possuem um pequeno número de funções disponíveis, que não preenchem as necessidades de monitoração desejáveis;

g. são direcionados a uma linguagem ou aplicação específica, sem possibilidade de adaptação a outras.

## 2. REQUISITOS DE UM BOM DEPURADOR

Os depuradores existentes possuem algumas facilidades interessantes mas as necessidades dos usuários não são plenamente atendidas. A experiência dos autores permite considerar um bom depurador uma ferramenta que preencha os seguintes requisitos:

a. ser simbólico, manipulando as entidades do programa (variáveis, tipos de dados, rotinas, comandos, etc) a nível de programa fonte;

b. acessar (visualizar e modificar) o código fonte, eliminando a necessidade de listagens e possibilitando a correção dos erros detectados durante a sessão de depuração;

c. não interferir no código do usuário, ou seja, não alterar a funcionalidade do código mas apenas monitorá-lo;

d. não necessitar mudanças de arquitetura do ambiente para ser utilizado (hardware, microcódigo, sistema operacional), possibilitando uma maior portabilidade;

e. prover um histórico da sessão de depuração, armazenando os comandos solicitados e as alterações de código e valores realizadas, permitindo análise da sessão após seu encerramento bem como mecanismos de interrupção/retomada e reaproveitamento de comandos anteriores;

f. aparecer como parte integrante do ambiente de desenvolvimento de modo que, quando um erro acontece, a alternativa natural ao usuário passa a ser o uso do depurador;

g. reconhecer a sintaxe e a semântica da linguagem do código fonte, para manipular expressões e comandos de forma interativa, como por exemplo chamadas de rotinas "ad hoc";

h. ser fácil de usar:
- comandos simples e poderosos
- interface amigável
- informações adequadas ao usuário;

i. suportar múltiplas linguagens.

Este último requisito deve ser melhor explicado. A maior parte das tarefas de um depurador é , de fato, independente de linguagem. Uma estrutura básica adequada para a construção de um depurador isolaria os módulos dependentes da linguagem dos não dependentes. Identificar, no entanto, de maneira determinística, estes módulos não é uma tarefa trivial e exige um estudo aprofundado das funções projetadas para o depurador e das estruturas de controle e de dados necessárias para a realização destas funções. A adaptação de figuras específicas das linguagens a uma estrutura básica poderia ser feita, então, de maneira localizada. Isto torna relativamente fácil o suporte a linguagens adicionais. Além disto, é menos custoso escrever um depurador que manipula várias linguagens do que escrever vários depuradores cada um para uma linguagem [Beander 83].

A unificação funcional e sintática, obtida através de uma interface homogênea, propicia uma maior desenvoltura para o usuário.

## 3. UM DEPURADOR PARA O ADS

O Ambiente de Desenvolvimento de Software (ADS) [Price 85], um ambiente interativo com facilidades integradas para criação, edição, interpretação, teste e depuração de programas, está sendo construído no CPGCC/UFRGS.

Foi projetado e desenvolvido um depurador alto-nível para prover o ADS com uma ferramenta de depuração, que procura atender os requisitos acima mencionados, tornando viável o uso do ADS como um ambiente de programação para uma família de linguagens.

As diversas funções do ADS estão encapsuladas em seus diferentes módulos componentes. A estrutura funcional dos módulos pode ser encarada como um estrutura em camadas. Cada módulo é construído utilizando facilidades oferecidas pelo módulos das camadas interiores. O núcleo do ADS é o Editor Dirigido pela Sintaxe (EDS) e o interpretador utiliza funções do editor. Construído como uma camada externa ao interpretador, o depurador pode então utilizar recursos oferecidos pelo editor e interpretador.

### 3.1 Integração com o Editor

A integração do depurador com o editor EDS dá-se principalmente através das estruturas de dados e rotinas de interface com o usuário.

O depurador usa a estrutura de árvores para representação abstrata de programas e as primitivas de navegação associadas a esta estrutura disponíveis no EDS. Rotinas para a modificação da árvore para inserção e remoção de nodos para depuração são da mesma forma utilizadas. A Tabela de Símbolos, gerada pelo reconhecedor do editor, é também acessada pelo depurador.

A interface com o usuário prevista no depurador está toda calcada em facilidades como "pretty-printing", navegação, paginação e exibição de textos utilizando primitivas do EDS. Estas primitivas são usadas porque o EDS é a interface homogênea entre o conjunto de ferramentas do ADS e o usuário, provendo a padronização dos procedimentos de ativação das ferramentas.

### 3.2 Integração com o Interpretador

O depurador utiliza, para a monitoração do comportamento do programa, os mecanismos de avaliação de expressões e de execução do interpretador.

O controle de execução é feito pelo depurador, que identifica e trata os comandos de depuração instalados, acompanha chamadas de rotinas e a manipulação da Tabela de Símbolos. Não há interferência do depurador se um evento não possui nenhum comando de depuração associado ou se o modo de depuração não está ativo.

Quando um erro de execução acontece, o depurador interrompe o programa e passa o controle ao usuário, esperando a solicitação de algum comando de depuração.

Com isto, construiu-se um sistema em que tanto a submissão do programa como o teste e depuração são incrementais. O ambiente de teste e depuração torna-se o mesmo de preparação do programa. As facilidades de consulta a tabela de símbolos foram extendidas para consulta e modificação a variáveis de execução. A execução, navegando sobre a árvore de representação do programa, pode ser interrompida e continuada por comandos do usuário. Além disto, partes do programa podem estar sendo testadas à medida em que vão sendo editadas/modificadas.

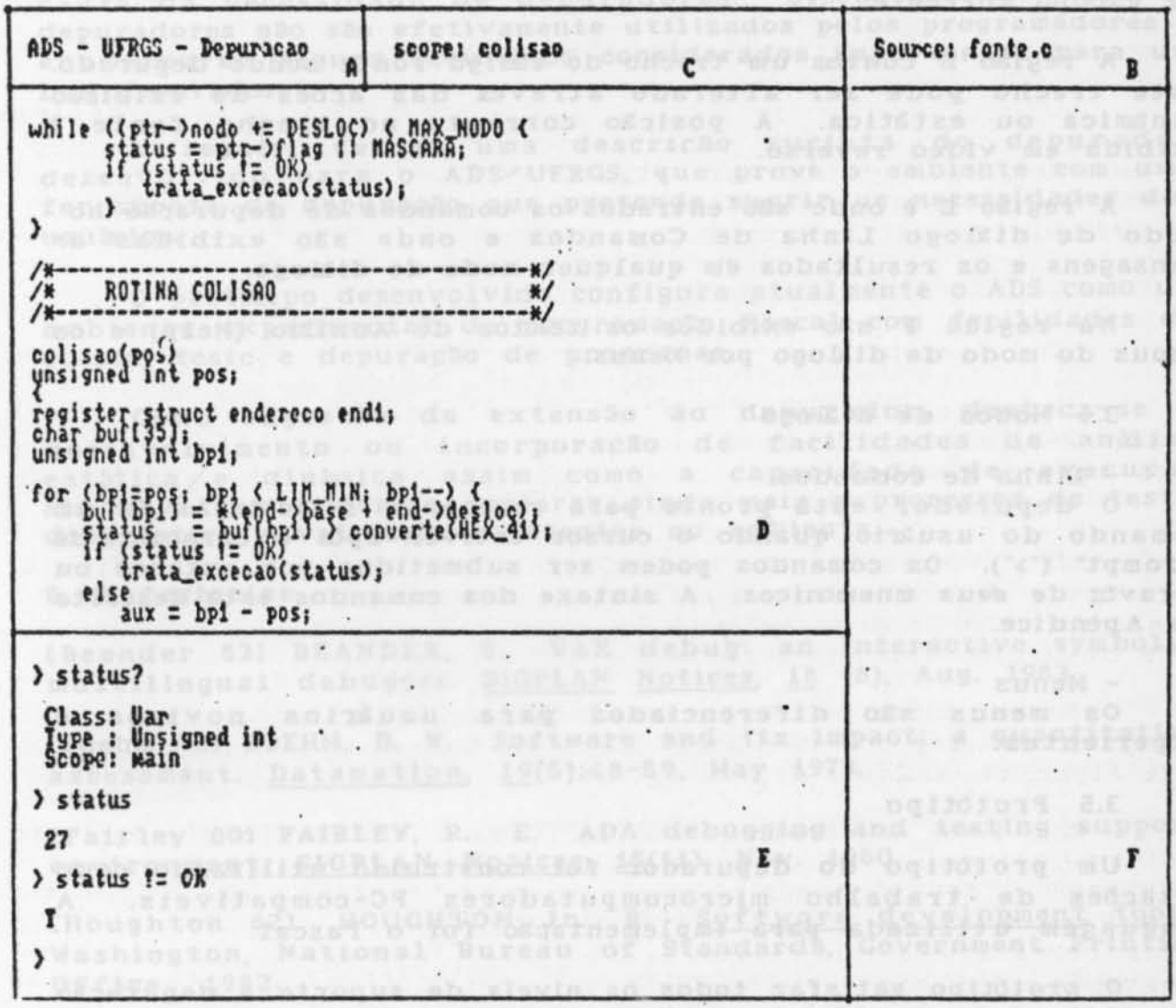
### 3.3 Interface com o usuário

A figura FigTela mostra o layout da tela de interação com o usuário no modo Depuração.

A seguir a explicação das regiões das telas.

A região A informa o nome do ambiente de desenvolvimento de Software e o modo de operação corrente do sistema. Atualmente, os modos de operação são Edição, Interpretação e Depuração.

A região B mostra o nome do arquivo que contém o código fonte sendo depurado.




FigTela. Layout da tela de interação com o usuário

A região C mostra o escopo corrente válido para o ponto atual da sessão. Em geral, a posição corrente do cursor pertence ao escopo corrente.

A região D contém um trecho do código fonte sendo depurado. Este trecho pode ser alterado através das ações de exibição dinâmica ou estática. A posição corrente no trecho fonte é exibida em vídeo reverso.

A região E é onde são entrados os comandos de depuração no modo de diálogo Linha de Comandos e onde são exibidas as mensagens e os resultados em qualquer modo de diálogo.

Na região F são exibidos os textos de Auxílio (Help) e os menus do modo de diálogo por Menus.

### 3.4 Modos de diálogo

- Linha de comandos

O depurador está pronto para receber a solicitação de um comando do usuário quando o cursor estiver após o caracter de "prompt" (">"). Os comandos podem ser submetidos por extenso ou através de seus mnemônicos. A sintaxe dos comandos está descrita no Apêndice.

- Menus

Os menus são diferenciados para usuários novíços e experientes.

### 3.5 Protótipo

Um protótipo do depurador foi construído utilizando como estações de trabalho microcomputadores PC-compatíveis. A linguagem utilizada para implementação foi o Pascal.

O protótipo satisfaz todos os níveis de suporte a depuração identificados por [Fairley 80]:

1. Comandos de saída para diagnósticos;
2. "Dumps" de valores de estruturas de dados;
3. Rastreamento seletivo ("selective trace");
4. Diagnósticos controlados por asserções;
5. Breakpoints estabelecidos/executados interativamente;
6. Retrospecção de estado ("backtrace");
7. Breakpoints condicionais;
8. Modificação do estado de execução (valores de estruturas de dados);
9. Modificações localizadas (num trecho) do código fonte;
10. Modificações arbitrárias (em vários trechos) do código fonte.

Maiores detalhes sobre o projeto e os comandos do depurador e a construção do protótipo podem ser encontrados em [Pimenta 87].

## 4. CONCLUSOES

Neste artigo, apresentou-se a existência de erros como a causa da necessidade de depuradores. Discutiu-se por que os depuradores não são efetivamente utilizados pelos programadores e enumerou-se alguns requisitos considerados importantes para um bom depurador.

A seguir, fez-se uma descrição sucinta do depurador desenvolvido para o ADS/UFRGS, que provê o ambiente com uma ferramenta de depuração que pretende suprir as necessidades dos usuários.

O protótipo desenvolvido configura atualmente o ADS como um ambiente experimental de programação Pascal com facilidades de edição, teste e depuração de programas.

Como sugestão de extensão ao depurador, destaca-se o desenvolvimento ou incorporação de facilidades de análise estática e dinâmica assim como a capacidade de execução simbólica, que poderiam acelerar ainda mais o processos de teste, diagnóstico e validação existentes no ambiente.

## 8. Referências

[Beander 83] BEANDER, B. VAX debug: an interactive symbolic, multilingual debugger. SIGPLAN Notices, 18 (8), Aug. 1983.

[Boehm 73] BOEHM, B. W. Software and its impact: a quantitative assessment. Datamation, 19(5):48-59, May 1973.

[Fairley 80] FAIRLEY, R. E. ADA debugging and testing support environment. SIGPLAN Notices, 15(11), Nov. 1980.

[Houghton 82] HOUGHTON Jr., R. Software development tools. Washington, National Bureau of Standards, Government Printing Office, 1982.

[Pimenta 87] PIMENTA, M. S. Um depurador simbólico multilinguagem integrado a um ambiente de desenvolvimento de software. Porto Alegre, CIC/UFRGS, 1987. (Trabalho de Diplomação).

[Pressman 82] PRESSMAN, R. S. Software engineering: a practitioner's approach. Auckland, McGraw-Hill, 1982.

[Price 85] PRICE, R. T. De um editor dirigido por sintaxe a um ambiente para desenvolvimento de software. In: CONGRESSO NACIONAL DE INFORMATICA, 18., São Paulo, 1985. Anais. São Paulo, SUCESU, 1985. V.1.

[Ramamoorthy 77] RAMAMOORTHY, C. V. & HO, S. F. Testing large software with automated software evaluation systems. In: YEH, Raymond, ed. Current trends in program methodology. Englewood Cliffs, Prentice-Hall, 1977. V. 2.

[Wilkes 76] WILKES, M. V. Software engineering and structured programming. IEEE Transactions on Software Engineering, SE-2, Dec. 1976.

## APENDICE

### SINTAXE DOS COMANDOS DO DEPURADOR

```
<comando_Exibir_Valor> ::= <Var>
<comando_Atribuir_Valor> ::= <Var> = <Expressão>
<comando_Avaliar_Expressão> ::= <Expressao>

<comando_Executar_Código> ::= GO <Label>  |
                              GO  TOP     |
                              GO          |
                              GO <Nome_sinônimo_de_posição>
<comando_Exibir_Mensagem> ::= MSG <String>
<comando_Chamar_Rotina> ::= <Rotina> <<param1>,<param2>,<param3>,..>
<comando_Exibir_Fonte> ::= TYPE ON | TYPE OFF

<comando_Informações_do_Programa> ::=
        <subcomando_Informações_de_Identificadores>      |
        <subcomando_Informações_sobre_Tipos_de_Dados>    |
        <subcomando_Informações_sobre_Arquivos>

<subcomando_Informações_de_Identificadores> ::= <Identificador>?
<subcomando_Informações_sobre_Tipos_de_Dados> ::= <Tipo_de_dado>?
<subcomando_Informações_sobre_Arquivos> ::= FILE <Nome_do_arquivo>?

<comando_Níveis_de_Escopo> ::= SCOPE TREE

<comando_Execução_Passo_a_Passo> ::= STEP <Tamanho_do_passo> |
                                     STEP OFF

<comando_Instantâneo> ::= SNAPSHOT            |
                          SNAPSHOT GLOBAL     |
                          SNAPSHOT ALL

<comando_Armazenar_Estado> ::= CHECKPOINT
<comando_Restaurar_Estado> :: = BACKTRACE <Idcheck>

<comando_Histórico> ::= LOG ON                    |
                        LOG OFF                   |
                        LOG  LIST  <Idsessão>     |
                        LOG  CLEAR  <Idsessão>    |

<comando_Definição_de_Sinônimos> ::=
            <Nome_de_Sinônimo> EQU CURRENT                  |
            <Nome_de_Sinônimo> EQU <Identificador>          |
            <Nome_de_Sinônimo> EQU <Expressão>              |
            <Nome_de_Sinônimo> EQU <Símbolo_do_depurador>   |
            <Nome_de_Sinônimo> EQU <Lista_de_comandos>      |
```

```
                <Nome_de_Sinônimo> EQU OFF
<comando_Execução_a_partir_do_Histórico> ::= RESTART <Sessão>

<comando_Rastreamento> ::= <rastreamento_de_execução>      ;
                           <rastreamento_de_subrotinas>    ;
                           <rastreamento_de_alteração>

<rastreamento_de_execução> ::=
                    EXECUTION TRACE [<Sinônimo_de_posição>] ON ;
                    EXECUTION TRACE [<Sinônimo_de_posição>] OFF

<rastreamento_de_subrotinas> ::=
                    CALL TRACE [<Lista_de_subrotinas>] ON ;
                    CALL TRACE [<Lista_de_subrotinas>] OFF

<rastreamento_de_alteração> ::=
                    VALUE TRACE <Lista_de_var> ON ;
                    VALUE TRACE [<Lista_de_var>] OFF

<comando_Ponto_de_Suspensão> ::=
                <breakpoint_incondicional>    ;
                <breakpoint_condicional>      ;
                <breakpoint_por_asserção>     ;
                <breakpoint_por_atualização>  ;
                <lista_breakpoints>           ;
                <desabilita_breakpoints>      ;
                <habilita_breakpoints>        ;
                <remove_breakpoints>

<breakpoint_incondicional> ::= BREAK [<Nome_de_macro>]

<breakpoint_condicional> ::= BREAK WHEN <Condição> [<Nome_de_macro>]

<breakpoint_por_asserção> ::=
                    BREAK ASSERTION <Asserção> [<Nome_de_macro>]

<breakpoint_por_atualização> ::=
                    BREAK ON CHANGES <Var> [<Nome_de_macro>]

<lista_breakpoints> ::= BREAK LIST

<desabilita_breakpoints> ::= BREAK DISABLE <Lista_de_breakpoints> ;
                             BREAK DISABLE *

<habilita_breakpoints> ::= BREAK ENABLE <Lista_de_breakpoints> ;
                           BREAK ENABLE *

<remove_breakpoints> ::= BREAK CLEAR <Lista_de_breakpoints> ;
                         BREAK CLEAR *
```