Ciência & Engenharia (Science & Engineering Journal) ISSN 0103-944X
14 (1): 95 - 99, 2005

UM AVALIADOR DE EXPRESSÕES EM FORTRAN

Wilton Pereira da Silva[1], Cleide M. D. P. S. Silva[1], Ivomar Brito Soares[2]
José Luís do Nascimento[2], Cleiton Diniz Pereira da Silva e Silva[3]

[1,2]Centro de Ciências e Tecnologia
Universidade Federal de Campina Grande
R. Aprígio Veloso, 882, Campina Grande
PB – Brasil, CEP 58109-970
[1]Professores do Departamento de Física
wiltonps@uol.com.br
[2]Alunos de Eng. Elétrica e do Programa de Bolsas de Iniciação Científica – PIBIC – CNPq
ivomarbrito@uol.com.br ; jluisn@dee.ufcg.edu.br
[3]Instituto Tecnológico de Aeronáutica (ITA)
Doutorado em Eng. Eletrônica, SP, Brasil
cleitondiniz@directnet.com.br

ABSTRACT

This paper aims to communicate the development of an expression evaluator (function parser) for the Fortran programming language, to make its source code available and to show the results of a comparative analysis among its performance and two other available ones (the only ones found in successive searches on the internet), in open source, for this language. The developed parser presented a performance significantly superior, in speed, to the similar tested.
Keywords: Expression evaluator, function parser, open source, Fortran, softwares for engineering.

RESUMO

Este artigo tem o objetivo de comunicar o desenvolvimento de um avaliador de expressões (parser) para a linguagem Fortran, de disponibilizar o seu código fonte e de apresentar os resultados obtidos na comparação entre o seu desempenho e o de dois outros disponíveis (únicos encontrados em sucessivas buscas na internet), em código aberto, para essa linguagem. O parser desenvolvido apresentou um bom desempenho, com performance significativamente superior, em rapidez, aos similares testados.
Palavras Chave: Avaliador de expressões, “parser” para funções, código aberto, Fortran, programas para engenharia.

1 - INTRODUÇÃO

Há alguns anos, professores do Grupo de Instrumentação para o Ensino de Física (GIEF) do Departamento de Física (DF) do Centro de Ciências e Tecnologia (CCT) da Universidade Federal da Paraíba (UFPB) desenvolveram, na plataforma DOS, usando a linguagem de programação Fortran (77), um programa de ajuste de curvas. O programa, então chamado “Ajuste”, destinava-se à análise de dados experimentais e utilizava a técnica de regressão não-linear. Tal programa foi inicialmente utilizado por alunos da disciplina Física Experimental I oferecida pelo DF/CCT/UFPB, e posteriormente por alunos e pesquisadores de outras universidades brasileiras. Como se sabe, a plataforma DOS tem enfrentado um constante declínio em seu uso, e vem sendo sistematicamente substituída por novas plataformas, mais poderosas e amigáveis. Atualmente sua utilização praticamente inexiste e os programas já desenvolvidos tiveram que ser refeitos nessas novas plataformas.

Em face deste panorama, os criadores do “Ajuste” decidiram desenvolver uma versão deste programa para a plataforma Windows e, nessa nova versão, o seu nome passou a ser “LAB Fit Curve Fitting Software” (LabFit) [1]. Como se sabe, um software de ajuste de curvas normalmente tem uma biblioteca com um conjunto de funções pré-definidas e uma opção para que o usuário possa escrever a sua própria função de ajuste, o que requer um código específico chamado de “avaliador de expressões” (parser) para o cálculo de tal função para um dado conjunto de valores das variáveis independentes. Durante o processo de migração, além do estudo de novos comandos da linguagem (Fortran 90, [2]) e da parte gráfica ([3] e [4]) surgiu a necessidade de se resolver um problema: não se conseguia, na época, obter um avaliador de expressões, em código aberto, desenvolvido em Fortran, cuja implementação fosse adequada às necessidades do programa em desenvolvimento. Na verdade, repetidas buscas na internet findaram em um único avaliador disponível para Fortran, desenvolvido na Austrália [5]. Mas o parser australiano, embora fosse de fácil incorporação ao programa e útil em muitas aplicações, era proibitivamente lento para os propósitos requeridos: regressão não-linear. Num programa de ajuste de curvas podem ocorrer centenas de milhares de iterações, durante o processo de busca da convergência [6], e isso requer que o código para a avaliação da função de ajuste seja extremamente eficiente e otimizado.

Para a versão DOS do software, não havia a necessidade de um avaliador de expressões devido ao recurso utilizado: o programa fonte era embarcado, em fragmentos, que eram literalmente emendados, através de um arquivo de lote, à função a ser calculada, especificada pelo usuário, formando um único programa fonte que, então, era compilado através de um compilador Fortran [7], também embarcado. Naturalmente, este recurso inexiste em Windows devido à complexidade e ao tamanho dos compiladores para essa plataforma, além de problemas relacionados às limitações quanto ao direito de uso. O caminho natural para contornar tal obstáculo seria a utilização de um parser, mas o único disponível para a linguagem utilizada não foi considerado adequado às finalidades do programa em desenvolvimento. Assim, só restou a alternativa do desenvolvimento de um avaliador de expressões para o LabFit e, para isso, foram inicialmente estabelecidos os requisitos que tal avaliador deveria ter ao ser usado pelo software.

2 – REQUISITOS DO AVALIADOR DE EXPRESSÕES

Em ajuste de curvas, utilizando-se a técnica de regressão não-linear, tem-se um processo iterativo em que um ponto ótimo é obtido através de uma condição de convergência. Assim, tal processo pode envolver milhares de iterações, o que demanda tempo. Isso requer que o código para o avaliador de uma dada experessão seja eficiente e otimizado. Para o caso específico de regressão, um parser deve atender a duas exigências. A primeira é que, uma vez lida a string com a função de ajuste, o processo de montagem da expressão aritmética correspondente seja feito uma única vez, logo após à leitura da string. Isso garante uma grande economia de tempo em um processo iterativo. A segunda exigência é que, uma vez interpretada e montada a expressão a ser avaliada, o cálculo de tal expressão para um conjunto de valores das variáveis seja feito da forma mais otimizada possível, o que garante a rapidez necessária, posto que o processo pode ser repetido por centenas de milhares de vezes.

3 – AVALIAÇÃO: NOÇÕES GERAIS

O procedimento no qual um programa recebe uma string contendo uma expressão numérica como, por exemplo, (4*3)/10^2, e retorna a resposta apropriada é chamado genericamente de avaliação de expressões. Tal procedimento é a base de todos os compiladores e interpretadores de linguagem, de programas de matemática e de tudo o mais que necessite interpretar expressões matemáticas de uma forma tal que o computador possa usá-las.

O algoritmo utilizado no desenvolvimento do parser foi o descendente recursivo. Tal algoritmo é descrito em alguns livros sobre a linguagem C (ver, por exemplo, [8]) e foi adaptado para Fortran, já que a bibliografia disponível sobre o tema para esta linguagem é praticamente inexistente. Mesmo assim, para se obter alguma familiaridade com o tema, de uma forma geral, pode-se recomendar, além da referência [8], a leitura das referências [9 e 10]. Como o objetivo deste artigo é apenas o de comunicar a criação de um parser desenvolvido em Fortran e de disponibilizar o seu código fonte (e não o de discutir o algoritmo, em si), serão mostrados apenas os passos iniciais do estudo efetuado.

3.1. Elementos De Uma Expressão: Precedência

Para os propósitos de desenvolvimento de um parser, deve-se partir do pressuposto de que expressões matemáticas sejam formadas pelos seguintes itens:

- Números;
- Operadores: + adição, - subtração, / divisão, * multiplicação, ** ou ^ potenciação;
- Parêntesis;
- Funções: sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, sind, cosd, tand, log, log10, nint, anint, aint, exp, sqrt, abs, floor;
- Variáveis.

Todos os itens anteriores podem ser combinados, obedecendo às regras da álgebra, para formar expressões matemáticas. Seguem alguns exemplos:

```
1/(a+b*x**(c-1) + 4.321)

a+(b-a)/(1+exp(-c*(x-d)))

a+b*log(x1)+c*log(x1)**2+d*log(x2)+e*log(x2)**2
(x+y+z+x*y+x*z+y*z+x/y+x/z+y/z+x*cos(x)+y*sin(y)+z
*tan(z)*2/(x+y+z+x*y+x*z+y*z+x/y+x/z+y/z+x*cos(x)+y
*sin(y)+z*tan(z))*3+sqrt(x*y*z+x+y+z)*log10(sqrt(x*2+
y*2+z*2)+x+y+z))
```

No desenvolvimento de um parser, um conceito importante é o da precedência de operadores e de funções [11]. Tal conceito diz qual é a operação que deve ser realizada primeiro, o que define a seqüência em que as operações devem ser executadas, com o propósito de se obter uma correta avaliação da expressão interpretada. Para os propósitos deste avaliador foi assumida a seguinte precedência:

Precedência dos Operadores

Maior Precedência	Parêntesis
	Funções
	** ou ^
	*, /
Menor Precedência	+, -

Como exemplo, seja avaliar a seguinte expressão:

16 - 3 * 4.

Naturalmente, essa expressão tem como resultado o valor quatro. Apesar de parecer uma tarefa fácil a criação de um código fonte que calcule o valor final para essa string específica, a questão que deve ser colocada é a criação de um parser que dá a resposta correta para qualquer expressão arbitrária. Não se pode simplesmente tomar os operadores em ordem, da esquerda para a direita,

pelo fato de que a multiplicação deve ser feita antes da subtração. O problema fica ainda mais complexo quando são adicionados à expressão: parêntesis, potenciação, variáveis e funções.

3.2. Análise De Uma Expressão

Existem várias possibilidades para se analisar e montar uma expressão matemática. No caso de um analisador descendente recursivo, as expressões são imaginadas como sendo estruturas recursivas, ou seja, tais expressões são definidas em termos delas próprias. Apenas para se ter uma noção básica das regras e da idéia utilizadas no desenvolvimento do avaliador, seja imaginar uma expressão contendo apenas os seguintes elementos: +, -, *, / e parêntesis. Neste caso, a expressão pode ser definida, a partir da leitura da string, com a utilização das seguintes regras básicas [12]:

Expressão → termo [+termo][-termo];

Termo → fator [*fator][/fator];

Fator → variável, número ou expressão.

Na nomenclatura apresentada anteriormente, os colchetes designam um elemento opcional e o símbolo "→" significa "produz". Tais regras são normalmente chamadas de "regras de produção da expressão". Assim, pode-se interpretar a definição de um termo como: "termo produz fator vezes fator ou fator dividido por fator". A precedência dos operadores está implícita na maneira como a expressão é escrita e o interpretador, ao ler a expressão em uma string, deve identificar as prioridades na seqüência das operações e ainda executar tais operações na seqüência identificada.

Para ilustrar a utilização das regras apresentadas seja tomar a expressão 6+3*D, na qual é possível identificar dois termos: o primeiro é o número 6 e o segundo é o produto dado por 3*D. O segundo termo tem dois fatores: 3 e D. Conforme se vê, esses dois fatores são constituídos por um número e por uma variável.

Essas regras apresentadas são a essência de um analisador descendente recursivo, que é basicamente um conjunto de funções mutuamente recursivas que operam de forma encadeada. A cada passo, o analisador executa as operações especificadas na seqüência algebricamente correta. Para se ter uma idéia mais concreta de como esse processo funciona, seja analisar a expressão a seguir:

9/3 – (100 + 56)

Inicialmente toma-se o primeiro termo, 9/3. Então, identifica-se cada fator e executa-se a divisão dos inteiros. O valor do resultado é 3;

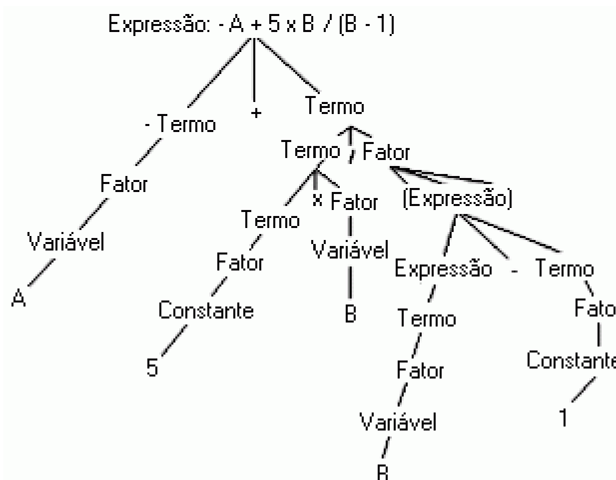
Toma-se o segundo termo, (100+56). Nesse ponto, começa-se a analisar recursivamente a segunda subexpressão. Toma-se cada um dos dois fatores que, então, são somados. O valor do resultado é 156;

Retorna-se da chamada recursiva e subtrai-se 156 de 3. A resposta é –153.

Há dois pontos básicos a serem lembrados sobre essa visão recursiva das expressões. Primeiro, a precedência dos operadores está implícita na maneira como as regras de produção são definidas. Segundo, esse método de análise e montagem de expressões é muito semelhante à forma como os humanos avaliam expressões matemáticas.

3.3. Árvore Sintática

Para que as noções apresentadas possam ser melhor compreendidas, será apresentada, a seguir, uma árvore sintática referente ao processo descendente recursivo para a expressão – A + 5 * B / (B – 1) [13].



Árvore sintática para a expressão – A + 5 * B / (B – 1).

4 – O CÓDIGO FONTE DESENVOLVIDO

O código fonte foi desenvolvido para atender aos requisitos básicos pré-definidos na seção 2 para as necessidades específicas do LabFit, de acordo com as regras que foram estabelecidas na seção 3. O resultado final está disponível no site indicado na referência [14].

Basicamente, o código é constituído das seguintes peças: 1) o parser, propriamente dito, contido em um módulo denominado interpreter.f90, 2) o programa principal, chamado test.f90, contendo exemplos de utilização. Uma vez que o programa principal tenha a informação da string, ele chama uma subrotina do módulo interpreter.f90, denominada "init", cuja função é fazer a interpretação, isto é, a montagem da expressão matemática. Isso feito, a função que avalia a expressão, chamada "evaluate", é evocada e o valor de tal expressão é retornado para um dado conjunto de valores das variáveis. O processo de avaliação da expressão pode ser repetido para outros conjuntos de valores das variáveis, sem a necessidade de remontagem da expressão, por quantas vezes forem necessárias. Há ainda a possibilidade de, numa dada execução, mudar a string e repetir o processo de interpretação e avaliação para a nova expressão especificada. Para tal, basta chamar a subrotina com o

nome de "destroyfunc", antes de informar a nova string e repetir o processo descrito anteriormente.

A filosofia empregada no desenvolvimento do parser foi a mesma utilizada normalmente no desenvolvimento de módulos, de uma forma geral. Isto quer dizer que os programadores em Fortran que utilizarão em seus programas o parser desenvolvido não têm a necessidade de conhecer o seu código fonte. Basta acrescentar o módulo interpreter.f90 em seus projetos, definir a string e chamar a subrotina "init", seguida da evocação da função "evaluate" conforme é mostrado no programa exemplo test.f90.

5 - ANÁLISE COMPARATIVA

Durante o desenvolvimento do parser os seus criadores tomaram conhecimento, via internet, de um outro similar desenvolvido na Alemanha [15], recém concluído. O desempenho deste parser foi testado incorporando-o ao LabFit e os resultados foram considerados bons. Mesmo assim, foi realizado um conjunto de testes para analisar a performance do avaliador desenvolvido em comparação com os outros dois similares disponíveis. O resultado destes testes indicaria o parser a ser aproveitado no software de ajuste de curvas.

Os testes consistiram na realização de cinco milhões de iterações para vinte e cinco expressões distintas, com a medição do tempo que cada avaliador gastou para realizar tal tarefa. Os testes foram realizados num computador Intel™ Pentium III, 128 Mbytes de memória RAM. A compilação foi feita no Compaq Visual Fortran (CVF) 6.5 utilizando-se a opção QuickWin Application. O tempo gasto para o cálculo de cada expressão devidamente compilada também foi medido, para que se tivesse uma noção do quão lento é um parser em Fortran, em comparação com o menor tempo possível. A diferenciação na performance dos avaliadores foi feita baseada apenas no tempo gasto para a execução das tarefas, porque os resultados numéricos de todos eles foram equivalentes, e compatíveis com o resultado numérico da expressão compilada. O relatório completo para todas as expressões foi disponibilizado na internet [16]. A seguir serão mostrados os resultados obtidos para cinco das vinte e cinco expressões testadas.

1ª Expressão:
(x+y+z+x*y+x*z+y*z+x/y+x/z+y/z+x*cos(x)+y*sin(y)+z*tan(z)*2/(x+y+z+x*y+x*z+y*z+x/y+x/z+y/z+x*cos(x)+y*sin(y)+z*tan(z))*3+sqrt(x*y*z+x+y+z)*log10(sqrt(x*2+y*2+z*2)+x+y+z))

Para x = 0.175, y = 0.110 e z = 0.900, tem-se:
Resultado final: 5.481916
Tempo do cálculo direto do compilador: 9.660s
Tempo do parser desenvolvido (LabFit): 48.125s
Tempo do parser australiano (Stuart Midgley): 167.07s
Tempo do parser alemão (Roland Schmehl): 78.693s

2ª Expressão:
a+b*x1

Para a = 0.900, b = 0.100 e x1 = 0.508, tem-se:
Resultado final: 0.9508000
Tempo do cálculo direto do compilador: 0.073s
Tempo do parser desenvolvido (LabFit): 1.904s
Tempo do parser australiano (Stuart Midgley): 26.999s
Tempo do parser alemão (Roland Schmehl): 3.255s

3ª Expressão:
cosh(log(abs(y*z+x**2+x1**x2)))+a*d*(exp(c*f)+154.3)

Para x = 0.175, y = 0.110, z = 0.900, a = 0.900, c = 0.110, d = 0.120, f = 0.140, x1 = 0.508 e x2 = 30.000, tem-se:
Resultado final: 20.69617
Tempo do cálculo direto do compilador: 7.265s
Tempo do parser desenvolvido (LabFit): 17.122s
Tempo do parser australiano (Stuart Midgley): 58.364s
Tempo do parser alemão (Roland Schmehl): 23.697s

4ª Expressão:
atan(sinh(log(abs(exp(z/x)*sqrt(y+a**c+f*e)))))

Para x = 0.175, y = 0.110, z = 0.900, a = 0.900, c = 0.110, f = 0.140 e e = 0.130, tem-se:
Resultado final: 1.559742
Tempo do cálculo direto do compilador: 9.591s
Tempo do parser desenvolvido (LabFit): 15.841s
Tempo do parser australiano (Stuart Midgley): 51.506s
Tempo do parser alemão (Roland Schmehl): 20.717s

5ª Expressão:
atan(sinh(log(abs(exp(z/x)*sqrt(y+a**c+f*e)))))*cos(log(abs(sqrt(y+a**c+f*e))))

Para x = 0.175, y = 0.110, z = 0.900, a = 0.900, c = 0.110, f = 0.140 e e = 0.130, tem-se:
Resultado final: 1.557368
Tempo do cálculo direto do compilador: 14.258s
Tempo do parser desenvolvido (LabFit): 24.518s
Tempo do parser australiano (Stuart Midgley): 76.528s
Tempo do parser alemão (Roland Schmehl): 32.915s

6 – CONCLUSÕES

A partir da análise dos dados apresentados na seção 5, nota-se que o avaliador de expressões desenvolvido apresentou, para os requisitos desejados, uma performance muito superior à dos outros dois existentes, em todos os testes efetuados. Assim, o parser não só foi utilizado no software LabFit, como também está sendo disponibilizado na internet para a utilização gratuita por parte dos interessados. Os criadores do parser acreditam ter colaborado, juntamente com os outros que desenvolveram trabalhos similares, para suprir um grande vazio na linguagem Fortran, ainda hoje muito usada por engenheiros e cientistas. A publicação deste artigo deve colaborar para que os usuários dessa linguagem tomem conhecimento da nova ferramenta, e para que dela tirem o máximo proveito em seus programas.

7 - REFERÊNCIAS

[1] W. P. SILVA, C. M. D. P. S. SILVA, *LAB Fit Curve Fitting Software*, online, disponível na internet em <www.labfit.net>, acesso em 29/11/2004

[2] S. J. CHAPMAN, "Fortran 90/95 for Scientists and Engineers", WCB/McGraw-Hill, 1st Edition, 1998.

[3] N. LAURENCE, "Compaq Visual Fortran – A Guide to Creating Windows Applications", Digital Press, Woburn MA, USA, 2002.

[4] W. P. SILVA ET AL, *VFortran Tutorial*, online, disponível na internet em <www.extensao.hpg.com.br>, acesso em 29/11/2004

[5] S. MIDGLEY, Department of Physics, University of Western Australia, *Parser Australiano*, online, disponível na internet em <http://smidgley.customer.netspace.net.au/fortran>, acesso em 29/11/2004

[6] Silva, W. P. E Silva, C. M. D. P. S., "Tratamento de Dados Experimentais".UFPB Editora Universitária, João Pessoa, PB, 2ª Edição, (1998).

[7] University of Waterloo, WATFOR77.EXE V 3.1, Agosto 1989.

[8] H. SCHILDT, "C Completo e Total", Makron Books, 3rd edition, pp. 584 – 606, 1996.

[9] L. J. DYADKIN, "Multibox Parser: No More Handwritten Lexical Analyzers", *IEEE Computer Society, Computer.org, Computer,* Vol. **12**, n° 5, Sep. 1995, pp. 61-67.

[10] p. R. Henriques et al, "Automatic Generation of Language-based Tool", *Eletronic Notes in Theoretical Computer Science,* 65, n° 3, (2002), pp. 1-20.

[11] A. V. AHO, R. SETHI, J. D. ULMAN, "Compilers – Principles, Techniques, and Tools", Addison-Wesley Publishing Company, pp. 203 – 208, 1987.

[12] c. N. Fischer and r. J. Leblanc jr., "Crafting a Compiler With C", The Benjamin/Cummings Publishing Company, pp. 382 – 387, 1991.

[13] F. L BAUER, F. L. DE REMER, A. P. ERSHOV, D. GRIES, M. GRIFFITHS, U. HILL, J. J. HORNING, C. H. A. KOSTER, W. M. MCKEEMAN, P. C. POOLE, W. M. WAITE, "Compiler Construction – An Advanced Course", Springer-Verlag, 2nd Edition, p. 6, 1976.

[14] I. B. SOARES, J. L. NASCIMENTO, W. P. SILVA, *"Código Fonte do Parser",* online, disponível na internet em <http://zeus.df.ufcg.edu.br/labfit/functionparser.htm>, acesso em 29/11/2004.

[15] R. SCHMEHL, "Parser Alemão", online, disponível na internet em www.its.uni-karlsruhe.de/~schmehl/functionparserE.html, acesso em 29/11/2004

[16] I. B. SOARES, J. L. NASCIMENTO, W. P. SILVA, "Relatório Completo dos Testes Realizados", online, disponível na internet em < http://zeus.df.ufcg.edu.br/labfit/report.htm >, acesso em 29/11/2004

8 -DADOS BIOGRÁFICOS DO AUTOR PRINCIPAL

Wilton Pereira da Silva nasceu em 1953 em Goiânia-GO. É graduado em Engenharia Elétrica pela Universidade Federal de Goiás e tem mestrado em Engenharia Elétrica pela Universidade Federal da Paraíba. É coordenador do Grupo de Instrumentação para o Ensino de Física da UFCG, onde trabalha desde 1979. É autor dos livros "Tratamento de Dados Experimentais" (1998) e "Mecânica Experimental para Físicos e Engenheiros" (2000), ambos lançados pela UFPB/Editora Universitária. Também é autor do LAB Fit Curve Fitting Software (1999-2004).