CAMILO DE LELIS GONDIM MEDEIROS

PROJETO E IMPLEMENTAÇÃO DE UM SISTEMA TRANSFORMADOR DE PROGRAMAS FORTRAN.

Dissertação apresentada ao CURSO DE MESTRADO EM SISTEMAS E COMPUTAÇÃO da Universidade Federal da Paraíba, em cumprimento a parte das exigências para obtenção do Grau de Mestre.

GIUSEPPE MONGIOVI
Orientador

MARIO TOYOTARO HATTORI
Co-orientador

CAMPINA GRANDE
JANEIRO - 1986

A Clara e Camila.

Agradecimentos,

Ao professor Giuseppe Mongiovi, que foi bem mais que "O Orientador";

Muito especiais ao professor Mário Toyotaro Hattori pelas suas extremadas boa vontade, paciência, palavras de encorajamento, etc (não cabe tudo aqui);

Ao professor Ulrich Schiel por suas sugestões e pelos constantes incentivos;

Ao professor Marcus Costa Sampaio, Chefe do DSC, que sempre foi compreensivo e encorajador;

Ao professor Jacques Sauvé pela originalidade com que sempre estimulou o desenvolvimento deste trabalho, inclusive exigindo o cumprimento de prazos;

A todos os demais professores e funcionários do DSC, e

Também, muito mais que merecidos, a Emília Clara, minha esposa, e a Camila Carolina, minha filha, que tanto me apoiaram e que por tantas vezes se viram privadas da minha presença física por causa do desenvolvimento deste trabalho.

Camilo.

M488p Medeiros, Camilo de Lelis Gondim
Projeto e implementacao de um sistema transformador de programas / Camilo de Lelis Gondim Medeiros. - Campina Grande, 1986.
118 f.

Dissertacao (Mestrado em Sistemas e Computacao) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Programas de Sistemas 2. Programas FORTAN - Portabilidade 3. Dissertacao I. Mongiovi, Giuseppe II. Hattori, Mario Toyotaro III. Universidade Federal da Paraiba - Campina Grande (PB) IV. Título

CDU 004.45(043)

PROJETO E IMPLEMENTAÇÃO DE UM SISTEMA
TRANSFORMADOR DE PROGRAMAS FORTRAN.

CAMILO DE LELIS GONDIM MEDEIROS

DISSERTAÇÃO APROVADA EM 17/01/86.

GIUSEPPE MONGIOVI
Orientador

MARIO TOYOTARO HATTORI
Componente da Banca

ULRICH SCHIEL
Componente da Banca

CAMPINA GRANDE
JANEIRO - 1986

RESUMO

Neste trabalho são discutidos vários aspectos referentes à portabilidade de programas FORTRAN (com ênfase em subprogramas de Análise Numérica) e a aplicabilidade dos Sistemas Transformacionais como uma ferramenta de suporte para o desenvolvimento de "software". São estudados quatro dos principais Sistemas Transformacionais existentes e apresentados o projeto e a implementação de um Sistema Protótipo para efetuar a conversão automática de precisão, substituir chamadas a subprogramas por código em linha, converter programas para rodar em outros ambientes de computação e efetuar a conversão da documentação interna de programas que obedecem a uma padronização previamente estabelecida.

ABSTRACT

The several aspects refering the FORTRAN programs portability (emphasizing the Numerical Analysis subprograms), and the applicability of the Transformational Systems, as a tool to support software development are discussed. It is presented the analysis of four of the major Transformational Systems, and the design and implementation of a Prototype System to realize the following operations: automatic conversion precision; substitution of subprograms call by in line code; programs conversion to run on several computer environments; and conversion of a pre-defined standard internal programs documentation.

SUMARIO.

LISTA DAS ILUSTRAÇÕES.

1. INTRODUÇÃO.

1.1. APRESENTAÇÃO.

Este trabalho contém alguns dos resultados de nossos estudos sobre portabilidade de software.

Nele são considerados vários aspectos de portabilidade de programas e de subprogramas escritos em FORTRAN, com ênfase em subprogramas que implementam algorítmos da área de Análise Numérica.

Há, de alguns anos para cá, um interesse crescente de Universidades e de outras Instituições de Pesquisa de todo o mundo na busca de um objetivo comum: conseguir que programas de computador escritos para uma determinada máquina possam ser executados em outra máquina sem necessidade de grandes adaptações. Atualmente, inclusive os ambientes Industriais e Comerciais de Processamento de Dados se preocupam com este problema. Mesmo com o advento de novos computadores, de novas linguagens de programação de nível cada vez mais alto e de padronização para as linguagens já existentes, ainda hoje os programas escritos para uma dada máquina quase sempre não podem ser executados sem adaptações em uma outra máquina. A situação chega a ser tão crítica que programas

escritos numa certa linguagem para rodar numa dada máquina precisam ser transformados quando se utilizam diferentes compiladores dessa linguagem para essa mesma máquina.

É, portanto, de extrema importância a produção de software que possa ser executado em várias máquinas com o mínimo de alterações.

Para dar suporte ao processo de desenvolvimento de software surgem novas Metodologias de Programação com frequência.

Há, também, um interesse crescente por Sistemas de Desenvolvimento de Software, que estão se tornando dispositivos indispensáveis para a programação de computadores. Entre tais Sistemas, os Sistemas Transformacionais ocupam um papel de destaque.

Os Sistemas Transformacionais já são empregados com êxito na derivação de várias versões de unidades de programa a partir de uma versão básica. Estas derivações podem ser levadas a efeito, por exemplo, para a produção de versões de uma unidade de programa em uma nova precisão distinta daquela da versão existente, ou para a produção de uma versão otimizada da unidade de programa. Um bom Sistema Transformacional deve ser confiável, viável e exigir o mínimo de intervenção humana na sua utilização. Embora vários Sistemas Transformacionais tenham sido desenvolvidos, poucos possuem todas essas qualidades. A utilização de bons

Sistemas Transformacionais é grandemente encorajada pela redução que ocasiona aos custos de desenvolvimento de Software e pelo grau de confiabilidade do Software assim produzido.

1.2. IMPORTÂNCIA DO TRABALHO.

Temos no Campus II da UFPb aproximadamente 20 computadores (com vários Sistemas Operacionais) que, entre outras tarefas, são bastante utilizados para a solução de problemas científicos.

Em um destes computadores existe a Biblioteca Transportável de Análise Numérica (BITAN) com cerca de 200 rotinas que são implementações de algorítmos matemáticos.

A UFPb interessa-se por difundir a utilização da BITAN e um primeiro passo nesse sentido, será a sua implementação em outras máquinas da própria Universidade. Posteriormente, a utilização da BITAN poderá ser ampliada transferindo-a para outras instituições.

Para auxiliar a difusão da BITAN e para facilitar o seu transporte para outras máquinas, direcionamos nossos estudos para os Sistemas Transformacionais. Este direcionamento culminou com o projeto e desenvolvimento de um sistema protótipo, o qual descrevemos neste trabalho.

1.3. CARACTERISTICAS DO PROTOTIPO.

O Sistema Protótipo foi chamado de Sistema Transformador de Programas FORTRAN (TRAPO) e será aplicado à biblioteca existente.

O TRAPO pode converter programas FORTRAN, previamente padronizados, para rodar em outros ambientes de computação, pode mudar a precisão da aritmética utilizada, pode converter a documentação interna dos programas e pode ainda substituir chamadas a subprogramas por código em linha.

O funcionamento do TRAPO é controlado por diretivas do usuário especificadas na Linguagem Interativa de Seleção de Alternativas (LISA). Os comandos disponíveis na LISA são apresentados seguidamente ao usuário na forma de um "menu" que por ele deve ser preenchido. A LISA dispõe, além das opções para o direcionamento da conversão a ser efetuada, de um comando para mostrar a situação corrente e também de um comando para anular uma sessão de trabalho.

O Sistema efetua as conversões semi-automaticamente numa tentativa de minimizar a intervenção humana nas transformações - apenas os ajustes finais, após a conversão, são feitos manualmente.

Os interessados na BITAN receberão as cópias das rotinas transformadas diretamente do NPD/UFPb na forma de "deck de distribuição".

Uma vez que o Sistema manipulará com muitas seqüências de caracteres e levando em consideração as facilidades oferecidas pela Linguagem PASCAL, o que inclui também uma opção para identificar as construções utilizadas e que não fazem parte do padrão da linguagem, decidimos pela sua utilização para o desenvolvimento do Sistema.

1.4. VISÃO GLOBAL DO TRABALHO.

Inicialmente, no segundo capítulo, falamos sobre transformações de programas e seus tipos, e apresentamos uma resenha sobre quatro dos principais Sistemas de Transportabilidade de sotware que começaram a ser desenvolvidos nos anos 70. A resenha foi elaborada de modo a facilitar um estudo comparativo dos Sistemas descritos.

No terceiro capítulo, fazemos considerações sobre Sistemas de Transportabilidade de Software Matemático e sobre a BITAN, e apresentamos o projeto do Sistema Protótipo desenvolvido.

A descrição dos programas do Sistema Protótipo e detalhes de implementação são apresentados no quarto capítulo.

Finalmente, no quinto capítulo, concluimos fazendo uma análise dos resultados obtidos, relacionando as dificuldades encontradas para a implementação do protótipo, identificando pontos que poderiam ser melhorados e apresentando sugestões

para estudos posteriores.

2. RESENHA.

2.1. INTRODUÇÃO.

Neste capítulo apresentamos conceitos de transportabilidade e de transformação de programas além de descrevermos quatro dos mais difundidos Sistemas de Transportabilidade de Software Matemático atualmente em uso: os Sistemas da IMSL ([1], [2], [4]), de Krogh ([4], [16]), do NAG ([3], [4], [10], [12], [13], [14])) e TAMPR ([4], [5], [9], [20]).

Para facilitar um estudo comparativo, a descrição de cada um dos sistemas citados é feita em quatro etapas: na primeira etapa, descrevemos o ambiente em que o sistema foi desenvolvido e onde seria executado; na segunda etapa, apresentamos os objetivos que orientaram o desenvolvimento do sistema; na terceira etapa mostramos a abordagem seguida para atingir os objetivos do sistema e, na última etapa, fazemos alguns comentários sobre o sistema, numa tentativa de mostrar as diferenças entre ele e seus similares.

2.2. TRANSFORMAÇÃO E TIPOS.

Partsch & Steinbrueggen [20] definem um "esquema de programa" como sendo "a representação de uma classe de programas relacionados". Um esquema de programa é originado do programa correspondente por parametrização.

"Programação Transformacional", também segundo [20], "é uma metodologia de construção de programas por aplicação sucessiva de mapeamentos de um esquema de programa para um outro, mantendo determinadas relações semânticas".

Um "Sistema de Transformações" (ou Sistema Transformacional), ainda segundo [20], "é um sistema implementado para dar suporte à programação transformacional".

A meta mais comum de um sistema transformacional é dar suporte à modificação de programas, o que faz com que o interesse por programação transformacional e por sistemas de transformação como auxílio ao processo de desenvolvimento de programas aumente a cada dia.

Isso inclui a adaptação de programas para, por exemplo, rodar em um novo ambiente, o que significa dizer que os sistemas transformacionais são úteis como ferramentas de suporte na portabilidade de programas.

Segundo Poole & Waite [21] "Portabilidade é uma medida da facilidade com que um programa pode ser transferido de um

ambiente para outro: se o esforço requerido para mover o programa é muito menor do que o requerido para implementá-lo inicialmente, então nós dizemos que ele é altamente portátil".

Ford [11] afirma que "um programa é portátil entre as configurações a e b se ele computa em níveis prescritos de precisão e eficiência em cada ambiente de computação, sem qualquer mudança".

Para Ford [11], "um programa é transportável entre as configurações a e b se ele computa em níveis prescritos de precisão e eficiência em cada ambiente de computação com mudanças efetuadas automaticamente por um programa de computador".

Brown & Hall [6] utilizando uma idéia semelhante à de Poole & Waite anteriormente apresentada, afirmam que "um programa é portátil se o esforço para movê-lo para um novo ambiente é muito menor do que o esforço para re-escrevê-lo para o novo ambiente".

Porém, é em Aird at alli [2] que vamos encontrar uma série completa de definições e conceitos relacionados com portabilidade de software, dentre os quais selecionamos os seguintes por apresentarem uma maior afinidade com o nosso trabalho.

"Uma Unidade de Programa (UP) é um programa principal ou um subprograma, como definido no padrão ANSI".

"Um programa é um conjunto de unidades contendo precisamente um programa principal".

"Uma Unidade de Programa é Transportável se ela e seu material de suporte (incluindo documentos) contêm informações explícitas para produzir versões modificadas da Unidade de Programa".

"Uma Unidade de Programa é Portátil por Processador se ela é uma Unidade de Programa Transportável que contêm todas as informações de modificação em uma forma legível por máquinas e existe um pré-processador para produzir versões modificadas automaticamente".

"Uma Unidade de Programa é Portátil por Conversor se ela é portátil por processador e a Unidade de Programa não necessita de mudanças para compilar e executar em pelo menos um ambiente".

"Uma Unidade de Programa é Portátil para um conjunto de ambientes, se pode ser corretamente compilada e executada, sem qualquer modificação para cada ambiente no conjunto".

"Um deck base é um conjunto de Unidades de Programa a ser processado por um Sistema Transformacional".

"Um deck de distribuição é obtido com a remoção de todas as instruções do conversor contidas em um deck base".

Por analogia a Tanenbaum et alli [25] que definem "máquina destino" como sendo "a máquina para onde um

programa deve ser movido", definimos "ambiente destino" como sendo o ambiente (hardware + software) para onde um programa deve ser movido.

A expressão "conversão pirata" usada várias vezes neste texto, deve ser entendida como uma conversão (de qualquer natureza e extensão) levada a efeito por uma entidade não autorizada e/ou sem intruções explícitas para efetuá-la.

A figura 1, abaixo, mostra a evolução dos conceitos relativos a portabilidade apresentados anteriormente:

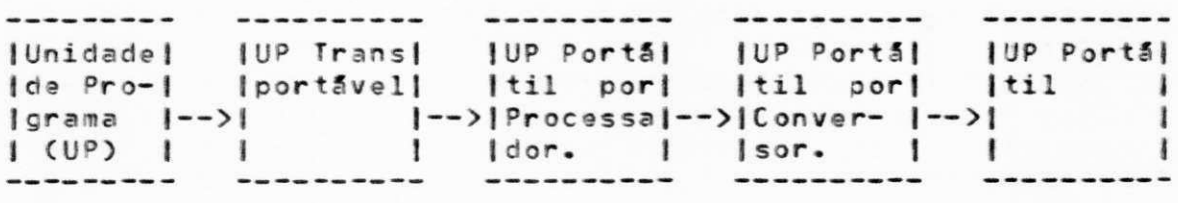


Figura 1 - Evolução dos conceitos de portabilidade.

2.3. O SISTEMA ESPECIALIZADOR DE KROGH.

2.3.1. Ambiente.

Fred T. Krogh e Shirley Singletary desenvolveram um pré-processador FORTRAN no Jet Propulsion Laboratory (California Institute of Technology) que foi chamado de "O Especializador".

O Especializador foi desenvolvido como uma ferramenta de uso individual para um especialista em análise numérica. O Especializador foi projetado como uma ferramenta para

facilitar o trabalho de um indivíduo que estivesse desenvolvendo um programa e desejasse torná-lo disponível para outros indivíduos usando computadores diferentes do seu. Uma vez que o usuário do Especializador não seria um especialista em desenvolvimento de bibliotecas, ele não foi projetado visando controlar a distribuição de programas.

2.3.2. Objetivos.

O Especializador tinha como objetivos básicos:

(1) Prover um formalismo para permitir a conversão automática de um programa para uso em diferentes máquinas, inclusive conversão de precisão;

(2) Prover um meio de especificar diferentes opções em um programa;

(3) Efetuar conversões (de programas) a baixo custo e ser transportável para várias máquinas; e

(4) Os objetivos de (1) a (3) deveriam ser obtidos de modo a minimizar a possibilidade de conversões piratas.

2.3.3. Abordagem.

Para alcançar os objetivos citados, decidiu-se que a versão mestre do programa seria mantida em forma executável, com as informações que identificam as diversas versões codificadas em comentários especiais. O Especializador

interpreta os comentários especiais da versão mestre para produzir uma nova versão.

Os comentários especiais podem ser de dois tipos:

(1) Para identificar uma seleção condicional de comandos FORTRAN a serem marcados como comentários ou não, dependendo de parâmetros de seleção e

(2) Para representar uma substituição de comando por um texto gerado pelo Especializador a partir dos comentários especiais.

O Especializador contém uma tabela com os valores de parâmetros ambientais caracterizando as máquinas mais usuais.

2.3.4. Comentários.

Com a abordagem utilizada, cada versão executável de um programa produzida pelo Especializador contém as informações que especificam as outras versões que são disponíveis. Portanto, quem recebe uma cópia do programa pode ver as outras versões que são disponíveis e isto desencoraja grandemente a produção de conversões piratas.

O Especializador evoluiu bastante desde sua primeira versão e já dispõe de vários comandos e de alto grau de parametrização e de automação para a especificação de conversões.

Com relação às dependências de máquina que não possam ser parametrizadas, um programador que usar o Especializador deve conhecer, em detalhes, o ambiente destino. O programador deve também saber escrever comandos apropriados para o Especializador, pois é através destes comandos, escritos em comentários especiais ao longo do programa, que ele especifica as várias opções a serem selecionadas visando a conversão para o ambiente destino (programas sem comentários especiais não são alterados ao serem submetidos ao Especializador).

O Especializador alcançou os objetivos estabelecidos no início do seu projeto (segundo [4]), tendo apenas duas desvantagens principais: é um pouco grosseiro para ser aprendido e usado e os comentários especiais ocultam um pouco o fluxo geral dos programas.

2.4. O SISTEMA "CONVERSOR" DA IMSL.

2.4.1. Ambiente.

A IMSL (International Mathematical and Statistical Libraries) de Houston, Texas, produz e dá suporte a uma biblioteca de subrotinas em FORTRAN nas áreas de matemática e estatística para cerca de 400 clientes com sete ambientes de computadores diferentes. Para facilitar o desenvolvimento e a manutenção dessa biblioteca, auxiliando na solução de seus problemas de portabilidade, a IMSL

desenvolveu o "Conversor FORTRAN".

Geralmente os códigos da IMSL diferem de ambiente para ambiente quando há necessidade para tais diferencas (p. ex., por considerações de eficiência), e portanto, são evitadas dependências desnecessárias de computadores. Na maioria dos códigos, 90% ou mais das linhas são comuns a todas as versões.

Tanto os programas da IMSL, quanto suas adaptações e ajustes finais, são feitas por pessoal pago.

Os clientes da IMSL recebem decks de distribuição diretamente da própria IMSL uma vez que deseja-se proteção contra conversões piratas, por usuários, de uma máquina para outra. O uso do Conversor é transparente para os clientes da IMSL.

2.4.2. Objetivos.

Com o desenvolvimento do Conversor FORTRAN, a IMSL esperava atingir os seguintes objetivos:

(1) O Conversor deveria ser uma ferramenta para simplificar a preparação de programas FORTRAN transportáveis por pessoal "de casa" com um mínimo de treinamento adicional;

(2) Deveria prover as necessidades de dependências explícitas de máquina nos programas;

(3) Deveria simplificar a conversão de um programa de uma precisão para outra;

(4) Sua utilização na preparação de programas para a biblioteca deveria ser transparente para seus clientes;

(5) Deveria ser transportável e possibilitar execuções a baixo custo; e

(6) A abordagem utilizada deveria permitir projeto e implementação rápidos.

Para facilitar o projeto e a implementação, não foram incluídas no Conversor ferramentas para modificar automaticamente programas já existentes na biblioteca.

2.4.3. Abordagem.

A abordagem geral utilizada no Conversor é muito semelhante àquela utilizada no Especializador de Krogh, apesar do desenvolvimento inteiramente independente e das diferenças existentes entre os objetivos de ambos.

A versão mestre de um programa é mantida em forma executável e contém, como comentários especiais, as informações que especificam as mudanças necessárias para a produção de outras versões.

Para atingir o objetivo (1), a Linguagem de Controle do Conversor destaca um pequeno número de recursos facilmente entendíveis.

Visando alcançar o objetivo (4), o Conversor pode produzir como saída: ou um deck base, que é uma versão de um programa, executável em um ambiente específico, e que contém comandos do Conversor; ou um deck de distribuição, que é uma versão de um programa executável em um ambiente específico, mas que não contém comandos do Conversor.

O Conversor obtém informações referentes à conversão de três fontes: instruções contidas no deck base, opções selecionadas para conversão, e de uma Tabela Mestre.

O Conversor possui uma opção para efetuar a conversão automática de precisão de um deck base de simples para dupla e vice-versa.

2.4.4. Comentários.

O Conversor FORTRAN da IMSL, do mesmo modo que o Especializador de Krogh, pode ser classificado como portátil por conversor (veja [2]) e, portanto, o deck base está sempre em forma executável. Esta abordagem tem a vantagem de permitir que um programa na forma de deck base seja depurado em uma máquina sem necessidade de pré-processamento.

A utilização da opção deck de distribuição (que não possui comandos do Conversor) atende ao objetivo (4) além de

desencorajar a produção de conversões piratas.

A conversão automática de precisão é uma forma de reduzir o trabalho do programador para obter versões de precisão simples ou dupla para um programa. A conversão automática de precisão tem um alto custo de execução uma vez que cada caráter do programa deve ser examinado e não mais somente aqueles dos comentários especiais.

Como acontece com o Especializador de Krogh, o Conversor requer o conhecimento do ambiente destino pelo programador e também possui uma Tabela Mestre com parâmetros de máquinas.

O Conversor atingiu os objetivos a que se propunha. A simplicidade dos comandos do Conversor, que são fáceis de aprender e de usar, fazem dele uma ferramenta atraente para programadores (segundo [4]).

2.5. O SISTEMA DE ARQUIVOS DA BIBLIOTECA MESTRE DO NAG.

2.5.1. Ambiente.

O Numerical Algorithms Group (NAG) surgiu da necessidade de desenvolvimento pelas Universidades Britânicas, de uma biblioteca de algorítmos numéricos em FORTRAN e ALGOL, para o computador ICL 1906 A.

Posteriormente, ao ser constatado que representantes de Universidades com outros tipos de máquinas interessavam-se

pela biblioteca do NAG, foi decidido que a biblioteca seria estendida para diversas outras máquinas.

Cada Universidade contribuía com programas para uma área da biblioteca (capítulo) na qual tinha um bom nível de conhecimento e recebia em mudança toda a biblioteca. A coordenação da biblioteca era uma tarefa da Equipe do Escritório Central do NAG.

Por questões econômicas, o NAG transformou-se em uma biblioteca comercial com organização descentralizada: os programas são desenvolvidos, em vários lugares, por especialistas em análise numérica (contribuidores) que não precisam incluir informações adicionais sobre as várias versões para outras máquinas, e são enviados para o Escritório Central. Os programas são modificados por implementadores (especialistas em uma máquina particular), quando necessário, para implementação em novas máquinas. Cada implementação completada é retornada para o Escritório Central que torna a biblioteca disponível para seus clientes.

A Equipe do Escritório Central desenvolveu um Sistema que protege a integridade dos programas contra erros grosseiros no gerenciamento do fluxo de programas (contribuídos e implementados) e é aplicável à biblioteca existente.

Um outro aspecto interessante do Sistema de Arquivos da

Biblioteca Mestre (MLFS) é que as implementações para máquinas diferentes são efetuadas assincronamente (uma não depende da outra).

O MLFS é basicamente um sistema de gerenciamento do texto fonte e como tal identifica e armazena as mudanças efetuadas por algum grupo na implementação ou atualização de cada versão básica da biblioteca.

2.5.2. Objetivos.

O MLFS, para o ambiente descrito, deveria alcançar os seguintes objetivos:

(1) Simplificar as atividades da biblioteca no Escritório Central do NAG, observando os objetivos de (3) a (6);

(2) Ser aplicável à biblioteca existente sem necessidade de modificações;

(3) Fornecer informações sobre o estado das várias implementações e ajudar a identificar divergências entre elas;

(4) Manter um histórico sobre as implementações (permitir desenvolvimento assíncrono);

(5) Proteger uma implementação contra influência de outros implementadores; e

(6) Auxiliar na revisão de implementações para um programa quando sua versão "contribuída" sofre atualização.

2.5.3. Abordagem.

Na abordagem seguida pelo NAG, a exemplo do que ocorre nas abordagens seguidas no Conversor (da IMSL) e no Especializador (de Krogh), há a utilização de uma versão do arquivo mestre do programa consistindo de comandos do programa e de comandos de controle intercalados. Por outro lado, diferente de tais Sistemas, no MLFS os comandos de controle são inseridos automaticamente.

Para produzir uma versão da biblioteca para uma determinada máquina, o MLFS efetua uma seleção sobre o arquivo mestre tendo em conta os comandos de controle presentes nesse arquivo. Estes comandos de controle indicam as implementações e versões para as quais são aplicáveis os comandos do programa.

O texto fonte das Unidades de Programa da biblioteca pode variar de uma versão para outra de uma mesma máquina. Portanto, é possível encontrar, no arquivo mestre, comandos do programa que não são utilizáveis para nenhuma das versões atualmente produzidas pela biblioteca. Estes comandos servem para manter um histórico da biblioteca.

Estudos estatísticos mostraram que 70% de todas as mudanças efetuadas ao implementar uma dada versão da

biblioteca foram devidas à conversão de precisão da biblioteca de simples para dupla ou vice-versa. Essa constatação levou o Escritório Central do NAG à decisão de desenvolver um Transformador Automático de Precisão que entre outras funções, tem a de resolver os problemas de constantes e construções que são dependentes de máquina.

A produção automática de comandos é feita por um programa (o "anti-editor") que compara um programa executável (fornecido por um implementador) para uma dada máquina, com a versão do arquivo mestre. Após a comparação o arquivo mestre é modificado de modo que o programa recebido passa a ser selecionável do arquivo mestre.

2.5.4. Comentários.

O arquivo mestre do MLFS não é um programa em forma executável, apesar de assemelhar-se conceitualmente ao deck base do Conversor e do Especializador e, portanto, o MLFS pode ser classificado como portátil por processador (veja [2]).

Esta abordagem é aceitável no contexto do NAG em face da inexistência de motivações para uso de uma biblioteca mestre executável: o Escritório Central do NAG não tem interesse em cientificar seus usuários da existência de outras versões e não há necessidade de uma forma executável para depuração pois as modificações nos programas executáveis são feitas em lugares distantes e reincorporadas

automaticamente na biblioteca mestre por um programa do Sistema.

O "anti-editor" permite que a biblioteca existente seja incorporada automaticamente pelo MLFS, como era requerido pelo objetivo (2).

Para atingir o objetivo (3) podem ser usados programas geradores de relatório para processar os comandos de controle existentes no MLFS.

Um histórico do desenvolvimento (objetivo (4)) é obtido registrando-se as versões a que pertencem os comandos do programa.

O MLFS é independente de indicadores estabelecidos nos programas, mas ainda requer que os implementadores tenham um conhecimento detalhado do ambiente destino.

Em resumo, os objetivos do projeto foram alcançados pelo MLFS (segundo [4]).

2.6. O SISTEMA "TAMPR" DE NAIS.

2.6.1. Ambiente.

O Sistema TAMPR (Transformation-Assisted Multiple Program Realization) foi desenvolvido a partir de 1973 no Argonne National Laboratory (Argonne, Illinois - EUA), um ambiente não comercial de pesquisas sobre metodologias e

ferramentas para desenvolvimento de software matemático.

O Sistema TAMPR surgiu como resultado das limitações e da experiência obtida com o desenvolvimento do seu predecessor que foi o "Sistema Especializador-Generalizador".

O Especializador-Generalizador foi concebido para operar sobre um conjunto de programas escritos usando aritmética de precisão dupla em FORTRAN. Inicialmente, o Sistema deveria converter esses programas para utilização em pelo menos cinco outros computadores.

O Sistema deveria automatizar a tarefa de conversão de precisão dos programas de dupla para simples e manter uma única versão mestre dos programas, em forma executável, que continha comandos de controle bastante rudimentares.

O Especializador-Generalizador entrou em operação em 1971 e funcionava bem, mesmo sem efetuar uma análise detalhada do texto fonte, com algumas limitações óbvias. Por exemplo, se o Sistema fosse utilizado para converter um programa arbitrário, que não houvesse sido escrito obedecendo a determinados padrões, ele poderia efetuar algumas mudanças indevidas. Além disso, o projeto do Sistema não permitia a pronta adição de novas funções.

Com a utilização do Especializador-Generalizador foram detectadas novas características desejáveis para ferramentas de software.

As principais características seriam a formatação automática de programas e a capacidade para especificação e execução de mudanças (de manutenção) que devessem ser efetuadas na maioria dos programas de um pacote.

Além dos fatos citados, havia dois outros que fortaleciam a idéia de desenvolvimento de um novo sistema: o interesse no desenvolvimento de pacotes em áreas mais complexas da Análise Numérica (do que Álgebra Linear) e no desenvolvimento descentralizado de software.

2.6.2. Objetivos.

A experiência obtida com o Sistema Especializador-Generalizador e a evolução do ambiente relacionado com ele, levaram ao estabelecimento dos seguintes objetivos para o Sistema TAMPR:

(1) O Sistema deveria reduzir, tanto quanto possível, a quantidade de programas armazenados na forma mestre; para tanto, deveriam ser automatizados os processos de derivação de novas versões para precisão simples e dupla e de novas versões dependentes de máquina;

(2) Para escrever programas transportáveis, o Sistema não deveria requerer do programador, um conhecimento detalhado dos ambientes destino;

(3) Deveria aceitar programas FORTRAN comuns, executáveis em alguma máquina, e convertê-los para a forma mestre, se possível sem a presença de marcadores especiais no programa fonte;

(4) O processo de derivação das versões deveria ser tão confiável quanto possível;

(5) Deveria unificar o formato e a estrutura dos programas; e

(6) Deveria ser flexível o bastante para atingir requisitos ainda desconhecidos, como por exemplo, a introdução de otimizações e a formatação de programas de acordo com especificações ainda não conhecidas.

Em parte por ser o Sistema TAMPR visto como um projeto de pesquisa de longa duração e em parte pela disponibilidade do Especializador-Generalizador, não foram feitas quaisquer restrições particulares sobre o seu desenvolvimento (quer sobre o tempo requerido para desenvolvimento, quer sobre velocidade de execução ou transportabilidade do Sistema).

2.6.3. Abordagem.

A abordagem utilizada no Sistema TAMPR tem algumas semelhanças com aquelas utilizadas no Especializador, no Conversor e no MLFS, apesar de ser substancialmente diferente de todas elas. A exemplo do que acontece naqueles Sistemas, no TAMPR existe uma versão mestre (não executável)

dos programas chamada de "forma canônica abstrata". Além disso, o Sistema TAMPR também consiste de um conjunto de processadores (funções de produção) que operam sobre a forma canônica abstrata para produzir as várias versões para um mesmo programa.

A operação do Sistema TAMPR consiste na aplicação de uma função de produção a um protótipo para obter uma versão particular daquele programa. As funções de produção são genéricas e podem ser aplicadas em sequência para a obtenção de transformações compostas confiáveis.

O pré-processamento é efetuado pelo Sistema TAMPR em três passos:

(1) Um reconhecedor converte um programa fonte FORTRAN (forma concreta) em forma abstrata cuja representação interna é uma árvore sintática para a versão do TAMPR de uma gramática FORTRAN;

(2) São aplicadas Transformações Intra-Gramaticais (IGT's) à forma abstrata para produzir a árvore transformada (forma abstrata canônica); e

(3) A árvore transformada é re-convertida para a forma fonte por um processo de formatação.

Com vistas à obtenção de flexibilidade (objetivo (6)) o Sistema TAMPR foi projetado para possuir alguns processadores programáveis: o "formatador" é um processador

programável em Linguagem de Controle de Formato (FCL) que converte da forma canônica abstrata para, por exemplo, FORTRAN executável; o "interpretador de transformações" que é o responsável pelas passagens da forma abstrata para uma outra forma, também abstrata, de acordo com as especificações escritas na linguagem das IGT's - Transformações Intra-Gramaticais (porque garantem a correção sintática do programa transformado), veja [5]; e o "reconhecedor" (programável na gramática BNF) que converte de FORTRAN para a forma abstrata.

2.6.4. Comentários.

A provisão de linguagens de finalidade especial para a especificação de processos e a capacidade para efetuar muitas operações sem a necessidade de inclusão de marcadores especiais no texto do programa de entrada, ambos requeridos pelo objetivo (6), são os aspectos que mais influem na utilização do Sistema TAMPR.

Estes processos levam o usuário, do Sistema TAMPR, a escrever as especificações das mudanças de modo independente dos programas particulares para os quais elas são aplicáveis, sendo requerido, portanto, um mínimo conhecimento do ambiente destino (objetivo (3)).

No Sistema TAMPR os processos de derivação de programas são confiáveis por serem especificados, em uma linguagem de alto nível, separados dos programas (objetivo (4)).

Como características negativas do Sistema podem ser citadas: o custo elevado de processamento (ocasionado por sua generalidade), a ausência de facilidades para manusear programas realmente dependentes de máquina e a não transportabilidade do Sistema.

O desenvolvimento do Sistema TAMPR trouxe como principal contribuição a idéia das funções de produção. As funções de produção, entre outros benefícios, tornam possível o transporte de programas para máquinas cujas implementações de FORTRAN violam o padrão, além de permitirem derivar mais de um programa a partir de uma única versão mestre (o que significa uma redução do número de programas na versão mestre).

Como pode ser visto, o Sistema TAMPR atingiu os objetivos propostos no início do seu projeto (segundo [4]).

2.7. CONCLUSÃO.

A comparação dos sistemas de transportabilidade apresentados mostrou que existem vários aspectos dos objetivos e das abordagens que são comuns a tais sistemas mesmo quando os ambientes divergem consideravelmente.

Os sistemas geralmente adotam a filosofia de utilização de uma versão mestre dos programas, se possível em número reduzido e constante, a partir da qual são produzidas as demais versões. A produção das novas versões geralmente é

feita no mínimo quase que automaticamente e visando confiabilidade.

A partir da versão mestre, as abordagens para transportabilidade de software matemático seguem duas correntes: a de portabilidade pura e a de funções de produção. Na portabilidade pura um programa protótipo geral é executável em todas as máquinas, enquanto que, com as funções de produção as várias versões específicas para cada máquina são construídas a partir do mesmo programa protótipo geral. Um estudo mais aprofundado das duas abordagens é encontrado em [4], [5] e [9].

Informações adicionais sobre portabilidade de software também são encontradas em [17] e [25].

Para um melhor visualização dos comandos de controle dos sistemas estudados apresentaremos um exemplo de conversão que foi extraído de [4]. O exemplo trata da conversão de um trecho de programa que calcula o produto interno de dois vetores.

Em máquinas com dez ou menos algarismos significativos em precisão simples (um computador IBM 4341, por exemplo), o trecho seria:

```
      DO 100 I = 1, N
      SOMA = SOMA + DBLE(X(I)) * DBLE(Y(I))
100   CONTINUE
```

Em máquinas com mais de dez algarismos significativos em precisão simples (um computador CDC 7600, por exemplo), o trecho equivalente seria:

```
      DO 100 I = 1, N
      SOMA = SOMA + X(I) * Y(I)
100   CONTINUE
```

Utilizando o Especializador de Krogh, teríamos:

```
      DO 100 I = 1, N
C$$   IF((.IMP. .EQ. 'S') .AND. (.EPS. .GE. 1.E-10))
C     SOMA = SOMA + DBLE(X(I)) * DBLE(Y(I))
C$$   ELSE
      SOMA = SOMA + X(I) * Y(I)
C$$   ENDIF
100   CONTINUE
```

Onde: IMP = Precisão (S ou D) e
EPS = Precisão Relativa.

Utilizando o Conversor da IMSL, teríamos:

```
      DO 100 I = 1, N
C$    IF(SSIGN .LE. 10) 1 LINE, 1 LINE
C     SOMA = SOMA + DBLE(X(I)) * DBLE(Y(I))
      SOMA = SOMA + X(I) * Y(I)
100   CONTINUE
```

Utilizando o Sistema do NAG, teríamos:

- Versão Contribuída (ICL 1906A):

```
      DO 100 I = 1, N
      SOMA = SOMA + X(I) * Y(I)
100   CONTINUE
```

- Versão IBM (Retornada por um implementador):

```
      DO 100 I = 1, N
      SOMA = SOMA + DBLE(X(I)) * DBLE(Y(I))
100   CONTINUE
```

Comparando estas versões, o "anti-editor" produz:

```
      DO 100 I = 1, N
*600BG 0100 99Z9
*60588       0200
      SOMA = SOMA + X(I) * Y(I)
*600BB 0200 99Z9
      SOMA = SOMA + DBLE(X(I)) * DBLE(Y(I))
*600BG 0100 99Z9
100   CONTINUE
```

Utilizando o Sistema IAMPR, teríamos:

- Versão Mestre, apropriada para máquinas com número de algarismos significativos menor ou igual a dez em precisão simples:

```
      DO 100 I = 1, N
      SOMA = SOMA + DBLE(X(I) * DBLE(Y(I))
100   CONTINUE
```

A transformação IAMPR:

```
      <primary>
      {.SD. DBLE(<primary> "1") ==> <primary> "1" .SC.}
```

Produz:

```
      DO 100 I = 1, N
      SOMA = SOMA + X(I) * Y(I)
100   CONTINUE
```

Um resumo do estudo comparativo dos quatro sistemas é apresentado nos Quadros I, II e III a seguir. A presença de um sinal de interrogação em algumas posições dos quadros significa que não foi possível encontrar, na bibliografia pesquisada, as informações referentes a estas posições. O hífen foi utilizado para indicar que determinado critério não se aplica a um dado Sistema, enquanto que espaços em branco são utilizados para representar as opções não disponíveis.

QUADRO I.

SISTEMA / AMBIENTE	KROGH	IMSL	NAG	TAMPR
Clientes	Uso Individual	400	vários	vários
Comercial/Pesquisa	P	C	P-->C	P
Controla Distribuição	Não	Sim	Sim	Sim

QUADRO II.

SISTEMA / OBJETIVOS	KROGH	IMSL	NAG	TAMPR
Conversão de Precisão	Possibilitar	Simplificar	Sim (APT)	Sim
Conversão de Ambientes	Sim	Sim	Predizer algumas mudanças	Sim
Conversão da Documentação Interna				
Conversões a Baixo Custo	Sim	Sim		
Ser Transportável	Sim	Sim		
Evitar Pirateamento	Sim		Sim	
Ser Transparente aos Usuários Finais		Sim		Sim
Projeto e Implementação Rápidos		Sim		
Efetuar Otimizações				Permite
Fornecer Histórico sobre as Implementações			Sim	
Ser Aplicável a uma Biblioteca específica			Sim	Qualquer uma
Formatar Programas				Sim
Não Exigir Conhecimento do Ambiente Destino				Sim

QUADRO III.

\ SISTEMA / ABORDAGEM/COMENTÁRIOS	KROGH	IMSL	NAG	TAMPR
Versão Mestre Executável?	Sim	Sim	Não	Não
Comandos de Conversão no Texto Fonte?	Sim	Sim	Sim: automaticamente.	Não
Utiliza Tabela Mestre?	Sim	Sim	Sim	-
Exige Conhecimento do Ambiente Destino?	Sim	Sim	Sim	Não
Utiliza Processadores Programáveis?	Não	Não	Não	Sim
Escrita dos Comandos de Conversão	+-Grosseiro	Fácil de Aprender e usar	Pelo Sistema	Especificada em BNF
Proteção contra Pirateamento	Cada UP contém todas as inform.	Deck de Distr.	Deck de Distr.	Cada UP já é um Deck de Distr.
Tempo de Desenvolvimento	?	?	18Homens-mês	Não limitado
Atingiu os Objetivos?	Sim	Sim	Sim	Sim

3. PROJETO DE UM SISTEMA PROTOTIPO.

Embora o desenvolvimento de Software de Análise Numérica seja feito tradicionalmente utilizando a Linguagem de Programação FORTRAN, a sua transportabilidade nem sempre é assegurada. Para solucionar os problemas de portabilidade, uma solução frequentemente adotada consiste em efetuar manualmente todas as adaptações até que o software produza os resultados desejados. Uma outra solução, modernamente adotada, para o mesmo problema, é a utilização dos Sistemas Transformacionais que, em geral, transformam parcialmente programas FORTRAN previamente padronizados.

Um bom Sistema Transformacional deve ser confiável, viável e requerer um mínimo de intervenção humana na sua utilização. A redução dos custos de desenvolvimento e o grau de confiabilidade do software assim produzido são os dois aspectos que mais encorajam a utilização dos Sistemas Transformacionais. Para que um Sistema Transformacional tenha ampla aceitação é aconselhável que ele exija o mínimo possível de padronização na escrita dos programas que serão transformados. Se possível, o Sistema deve fornecer os programas transformados com uma boa formatação. Alterar automaticamente comentários, mudar a precisão da aritmética

utilizada e efetuar otimizações em programas fonte, são outras características importantes dos Sistemas Transformacionais.

Neste capítulo apresentamos, como estudo de caso, o projeto de um Sistema Protótipo que chamaremos de Sistema Transformador de Programas FORTRAN (TRAPO) e que apesar de apenas parcialmente implementado, já dispõe de várias das características citadas anteriormente. Na seção 3.1 descrevemos o ambiente de desenvolvimento do protótipo, na seção 3.2 apresentamos os objetivos estabelecidos para o Sistema, na seção 3.3 detalhamos a abordagem de desenvolvimento e na seção 3.4 descrevemos os vários componentes do Sistema.

3.1. AMBIENTE.

O Sistema está sendo desenvolvido para transformar inicialmente uma biblioteca de rotinas numéricas existente em uma máquina IBM 4341 no Núcleo de Processamento de Dados da Universidade Federal da Paraíba - Campus II (a BITAN), para a obtenção de novas versões da biblioteca.

A BITAN conta atualmente com cerca de 200 subprogramas escritos em FORTRAN e há um grande interesse por parte da instituição em difundir a biblioteca para utilização em outras máquinas existentes na própria Universidade e, posteriormente, para intercâmbio com outros Centros de Pesquisa.

3.3. ABORDAGEM.

O TRAPO está sendo aplicado a uma biblioteca de rotinas numéricas, já existente, para a produção de rotinas transportáveis. Um dos requisitos a serem atingidos pelo TRAPO é a sua aplicação à biblioteca existente sem a necessidade de maiores adaptações. Para alcançar este objetivo, foi assumida a existência de uma série de restrições que devem estar presentes nos códigos fonte da versão mestre das unidades de programa da BITAN. A seguir relacionamos estas restrições. A seção 3.3.1 descreve-as em maiores detalhes juntamente com algumas outras.

(1) Todas as unidades de programa presentes na BITAN estão escritas corretamente e em forma executável (não serão efetuadas verificações a esse respeito);

(2) O ambiente destino foi previamente definido na Tabela Mestre do Sistema (discutida na seção 4.1.3): caso contrário, obtém-se uma mensagem de erro e o Sistema termina prematuramente a sessão de trabalho sem efetuar as transformações;

(3) Todas as variáveis estão declaradas no início das unidades de programa;

(4) Os nomes das variáveis usadas nas unidades de programa e os nomes dos subprogramas obedecem a uma padronização pré-estabelecida;

(5) Os COMMON das unidades de programa da BITAN são transparentes ao usuário;

(6) As unidades de programa da BITAN não utilizam aritmética do tipo misto;

(7) As constantes são definidas nos comandos DATA; e

(8) A documentação interna das unidades de programa da BITAN está em uma forma padronizada e transportável.

O Sistema deixa inalterados os itens correspondentes a cada uma das restrições de (3) a (8) caso elas deixem de ser observadas.

As restrições de (2) a (8) anteriormente citadas fazem referência a padrões no estilo de programação da biblioteca. Alguns destes padrões já foram considerados no desenvolvimento da BITAN, outros, que aparecem como sugestões, deverão ser incluídos nas novas rotinas que forem adicionadas à BITAN e em rotinas já existentes que venham a sofrer atualizações. É importante salientar que é recomendável seguir estes padrões tendo em vista principalmente a legibilidade e a transportabilidade das rotinas, e que a não adoção de alguns deles não implica na rejeição das rotinas pelo TRAPO - mesmo neste caso, o Sistema tentará efetuar o máximo em termos de transformação, não podendo, por exemplo, converter uma documentação interna não padronizada. A seguir, apresentamos os padrões existentes (seção 3.3.1) e, posteriormente, as sugestões

(seção 3.3.2).

3.3.1. A Padronização Existente.

A BITAN foi desenvolvida seguindo uma série de padrões de estilo de programação. Os padrões seguidos podem ser encontrados de forma detalhada em Hattori [15], de onde extraímos o seguinte resumo:

I - Conjunto de Caracteres.

Deve ser usado o conjunto comum à maioria das impressoras existentes. Não devem ser usados os sinais de maior (>) e menor (<) nos comentários - ao invés, use .GT. e .LT..

II - Subconjunto do FORTRAN.

Deve ser usado o subconjunto definido pelo PFORT que dispõe de um programa verificador desenvolvido por Ryder [22].

III - Parâmetros Ambientais.

As grandezas do ambiente de computação devem ser parametrizadas de acordo com as recomendações do Grupo de Trabalho WG 2.5 da IFIP (International Federation for Information Processing) descritas por Ford [11]. Os parâmetros geralmente mais utilizados são:

(a) - Parâmetros de Aritmética.

(1) Base usada para representação de números em ponto flutuante (SRADIX, DRADIX);

(2) Comprimento da Mantissa: o número de dígitos na base - RADIX armazenados na mantissa, incluindo, se for o caso, o dígito implícito quando o primeiro bit da mantissa de um número normalizado não é armazenado (SDIGIT, DDIGIT);

(3) Precisão Relativa: o menor número positivo x tal que (1.0 - x) < 1.0 < (1.0 + x), em que (1.0 - x) e (1.0 + x) são valores armazenados como resultados computados (SRELPR, DRELPR);

(4) Overflow: o maior i tal que todos os inteiros no intervalo [-i, i] pertencem ao sistema de números do tipo INTEGER (IOVFLO); o maior número x tal que x e -x pertencem ambos ao tipo REAL ou DOUBLE PRECISION (SOVFLO, DOVFLO);

(5) Underflow: o menor número real positivo x tal que x e -x são representáveis como elementos do conjunto de números do tipo REAL ou DOUBLE PRECISION (SUNFLO, DUNFLO); e

(6) Intervalo Simétrico: o maior inteiro i tal que qualquer operação aritmética 'op' é executável para todos os inteiros a e b satisfazendo |a| < i,

|b| < i, desde que o resultado matemático exato de a op b não exceda i em valor absoluto (IRANGE); o maior número real x tal que as operações aritméticas op são corretamente executadas para todos os elementos a e b do conjunto de números dos tipos REAL e DOUBLE PRECISION, desde que a e b e o resultado matemático exato de a op b não tenham nenhum valor absoluto fora do intervalo [1/x, x] (SRANGE, DRANGE).

(b) - Parâmetros de Entrada/Saída.

(1) Unidade Padrão de Entrada: número lógico da unidade padrão de entrada (NIN) e

(2) Unidade Padrão de Saída: número lógico da unidade padrão de Saída (NOUT);

(c) - Parâmetros Diversos.

(1) Número de caracteres por palavra (NCHAR);

(2) Tamanho de uma página de memória (NIPAGE); e

(3) Número de dígitos decimais (NIDEC, NSDEC, NDDEC).

Outros parâmetros são encontrados em [7], [8], [11] e [27].

IV - Compilador para Desenvolvimento.

Recomenda-se utilizar o compilador WATFIV devido às suas facilidades de depuração.

V - Regras para Obter Uma Boa Estrutura.

Mesmo sendo pobres as estruturas de controle do FORTRAN-66-ANSI, recomenda-se que além de evitar construções que usam truques sutis, sejam seguidas as regras apresentadas a seguir, para obter uma melhor estrutura.

(a) - Declaração de Tipos.

Devem ser declarados os tipos de todos os identificadores, incluindo os tamanhos de conjuntos, em comandos de especificação de tipos.

Os tipos devem ser declarados na ordem INTEGER, LOGICAL, REAL, DOUBLE PRECISION, COMPLEX, seguido da declaração EXTERNAL. Em cada declaração os identificadores devem estar classificados lexicograficamente, incluido os conjuntos com seus respectivos tamanhos.

Fica proibida a utilização de DIMENSION.

A declaração dos tipos dos parâmetros nos subprogramas deve ser separada das variáveis locais e deve ser feita na ordem em que aparecem, respeitada a

sequência acima.

Declarar separadamente os tipos de funções intrínsecas ou não.

(b) - O Comando de Repetição 'DO'.

Algumas regras devem ser seguidas quando da utilização do comando DO:

(1) Definir cada domínio de DO pelo comando DO e o CONTINUE correspondente;

(2) Utilizar variáveis simples para os limites e incrementos da variável de controle e cujos valores sejam sempre positivos;

(3) Fica proibida a transferência de fora para dentro do domínio de um DO; e

(4) Salvar a variável de controle do DO em outra variável caso seja necessário utilizá-la após o término do ciclo.

(c) - Comandos IF.

Fica proibida a utilização do IF aritmético.

(d) - Números de Comando.

Os números de comando devem aparecer em ordem crescente com incrementos maiores que 1. Recomenda-se também que:

(1) O primeiro número de comando seja 100 ou 200 (para facilitar possíveis acréscimos no início da unidade de programa);

(2) Sejam evitados números de comandos não referenciados; e

(3) Seja utilizado um número de comando com CONTINUE quando esse número for um destino de transferência por GO TO.

(e) - Exceções Aritméticas.

Deve ser evitada a produção de diagnósticos pela utilização de exceções aritméticas (overflow, underflow).

(f) - COMMON e EQUIVALENCE.

A utilização de COMMON só é permitida se for transparente ao usuário e nunca deve servir para associar variáveis com áreas de memória para economizar espaço. Recomenda-se que seja evitado o uso de EQUIVALENCE pois tende a obscurecer a unidade de

programa.

(g) - Subprogramas FUNCTION com efeitos colaterais.

Nunca escrever subprogramas FUNCTION com efeitos colaterais.

(h) - Ordenação dos Comandos.

Deve ser obedecida a seguinte ordem:

(1) Declaração de SUBROUTINE ou FUNCTION;

(2) Declaração de tipos de parâmetros:

- parâmetros simples do tipo INTEGER; e
- parâmetros que são conjuntos do tipo INTEGER;
- parâmetros de outros tipos, na sequência LOGICAL, REAL, DOUBLE PRECISION e COMPLEX;

(3) Documentação interna: descrição de cada parâmetro na chamada e no retorno, separadamente;

(4) Declaração de tipos para identificadores em COMMON seguida da declaração COMMON. Cada bloco de COMMON deve vir acompanhado de declaração dos identificadores que estão neste bloco;

(5) Declaração de tipos de todas as variáveis locais, funções e funções intrínsecas na sequência: INTEGER, LOGICAL, REAL, DOUBLE PRECISION, COMPLEX, EXTERNAL e EQUIVALENCE;

(6) Inicialização de constantes locais pela declaração DATA;

(7) Funções Declarações;

(8) Outros comandos, exceto END, do subprograma. Recomenda-se a utilização de um único RETURN e que os FORMAT acompanhem os comandos de entrada/saída que primeiro lhes referenciem; e

(9) END.

(i) - Convenções de Formatação.

As convenções adotadas são descritas a seguir.

(1) As declarações SUBROUTINE, FUNCTION, a declaração de tipos e a declaração DATA devem começar na coluna 7;

(2) A continuação de comandos longos deve utilizar um X na coluna 6 e a linha de continuação deve começar deslocada (endentada) pelo menos três (3) posições. O "ponto de quebra" de uma linha deve se dar depois de um operador binário (+, -, *, /, **) ou lógico (.AND., .OR., .NOT.); depois de uma "/",

num DATA; depois de um fecha parêntesis num EQUIVALENCE; depois de uma vírgula num CALL, GO TO computado, num GO TO com atribuição e numa lista;

(3) Os comandos entre um DO e o CONTINUE correspondente devem ser endentados com três posições;

(4) Os comandos correspondentes à parte do "ELSE" num IF lógico devem ser endentados com três posições. A parte correspondente a THEN deve começar com CONTINUE na mesma coluna do IF:

```
      IF ( ... ) GO TO 10
         Parte "ELSE"
   10 CONTINUE
      Parte "THEN"
      ...
```

(5) Utilização de espaços.

- Antes e depois de operadores binários (+, -, *, /, **), de operadores lógicos (.AND., .OR., .NOT.) e operadores relacionais;

- Antes e depois do operador de atribuição;

- Num comando DO antes e depois do número de comando, mas não entre os parâmetros;

- Antes e depois de "(" e de ")" num IF;

- Depois de um GO TO;

- Depois da vírgula numa lista; e

- Nas declarações de tipo depois de INTEGER, LOGICAL, REAL, DOUBLE PRECISION e COMPLEX.

(6) Utilização de comentários em branco (apenas um C na coluna 1). Deve ser utilizado um comentário em branco nos seguintes lugares:

- Depois da declaração SUBROUTINE ou FUNCTION;

- Depois da declaração de tipos dos argumentos;

- Após cada COMMON;

- Após o último comando DATA;

- Antes de um DO e após o CONTINUE correspondente; num ninho de DO's basta seguir a regra com o DO mais externo;

- Antes de um IF (lógico); e

- Depois da última função declaração seguida de uma linha demarcatória (recomenda-se de traços "-").

(j) - Convenções de Nomenclatura.

A única regra imposta é de indicar pela primeira letra o tipo da versão, convencionando-se que I, S, D, C e Z (se permitido) indiquem as versões inteira, precisão simples, precisão dupla, tipo complexo e tipo complexo de dupla precisão.

VI - Documentação Interna.

A disciplina adotada é descrita a seguir:

(a) Declaração FUNCTION ou SUBROUTINE;

(b) Declaração dos parâmetros (conforme seção 5.8);

(c) Descrição sucinta da finalidade do subprograma;

(d) Descrição dos parâmetros na chamada;

(e) Descrição dos parâmetros no retorno;

(f) Citação dos subprogramas externos utilizados;

(g) Citação dos subprogramas intrínsecos utilizados;

(h) Referências, se houver;

(i) Instituição, departamento, implementador e data da versão; e

(j) Corpo do subprograma: todas as seções comentadas de modo que facilite a sua manutenção.

VII - Construções Inseguras Quanto à Portabilidade.

Muitas construções permitidas em FORTRAN podem causar problemas de portabilidade. A seguir, apresentamos as mais comuns. Maiores informações podem ser conseguidas em [15] e [22].

(a) - Conversão de tipos.

Evite que conversões implícitas de tipos ocorram nos subprogramas. Utilize funções intrínsecas para efetuar conversões explicitamente. Evite usar funções intrínsecas que causam conversão implícita de tipos de argumentos e de resultados (por exemplo, MAX1, MIN1, AMAX0 e AMIN0), e não utilize funções intrínsecas não padronizadas.

(b) - Funções Intrínsecas Problemáticas.

As funções AMOD, MOD, DMOD, SIGN, ISIGN e DSIGN não são definidas quando o segundo argumento é zero. Certifique-se de que o segundo argumento não é nulo antes de utilizá-las.

(c) - Constantes e o DATA.

As constantes utilizadas nas unidades de programa devem ser representadas por variáveis cujos valores são definidos num DATA. Quando possível, utilize constantes na forma inteira nos DATA.

(d) - O comando DO.

Devem ser introduzidos mecanismos para decidir o que fazer quando o valor inicial da variável de controle for maior que o final em um comando DO.

A variável de controle num DO implícito, a exemplo do que acontece com o DO explícito, torna-se indefinida após a execução do DO.

(e) - Tipo Complexo de Dupla Precisão.

Os programas que utilizam o tipo Complexo de Dupla Precisão não são portáteis pois este tipo não é padronizado. Seria recomendado construir as operações utilizando aritmética REAL.

(f) - Cadeia de Caracteres.

Para manipulação de cadeias de caracteres só utilizar a conversão A1 com o tipo INTEGER.

A definição de caracteres num DATA deve ser feita

utilizando a conversão H (1H por caráter).

(g) - Conversão Alfanumérica num FORMAT.

Toda conversão alfanumérica num FORMAT deve ser feita no formato H.

(h) - Conversão dos Tipos REAL e DOUBLE PRECISION.

Para evitar problemas de portabilidade que ocorrem algumas vezes quando a saída dos tipos REAL e DOUBLE PRECISION é feita com todos os dígitos possíveis, aconselha-se deixar para o usuário a inclusão de sua rotina de saída (mesmo requerendo um parâmetro adicional no subprograma).

(i) - Entrada/Saída.

Devem ser utilizadas apenas as formas padronizadas de entrada e saída:

```
        READ (NIN, NF) Lista de Variáveis
    NF  FORMAT( ... )
e
        WRITE (NOUT, NF) Lista de Expressões
    NF  FORMAT( ... ),
```

onde NF representa um número de comando.

Nos subprogramas nunca deve haver entrada e, quando possível, evitar saída.

(j) - Constantes.

Certas constantes irracionais ou racionais têm representação infinita (como, por exemplo, 1/3 = 0.333333) e podem causar problemas de portabilidade. Deve ser seguida a convenção:

(1) Num DATA todas as constantes reais que têm representação decimal finita devem ser definidas na forma exponencial e

(2) As constantes que aparecem em fórmulas devem ser mantidas em forma racional, com os numeradores e os denominadores na forma exponencial. As constantes irracionais que têm representação simbólica (raiz quadrada, por exemplo) devem ser substituídas, quando possível, por uma chamada a uma função intrínseca.

3.3.2. A Padronização Proposta.

Embora a padronização existente atenda a muitas normas de boa programação, sentimos a necessidade de levar a efeito alguns acréscimos. Estes acréscimos serão efetuados às Convenções de Formatação (seção 3.3.1), visando melhorar a legibilidade das unidades de programa; e à Documentação

Interna (seção 3.3.1), com vistas à obtenção de uma documentação interna transportável.

I - Convenções de Formatação.

Aconselhamos seguir todas as convenções de formatação descritas na seção 3.3.1 e mais:

(a) Linhas em Branco.

Além dos lugares já convencionados, propomos que seja incluído um comentário em branco após qualquer desvio incondicional, para indicar uma quebra no fluxo do programa.

Para dar destaque aos comentários explicativos (veja seção 3.3.2), eles devem ser antecedidos e seguidos por uma linha de comentário em branco.

(b) Parêntesis.

Devem ser incluídos alguns parêntesis opcionais, nas expressões aritméticas, para melhorar a legibilidade e para prevenir possíveis erros.

II - Documentação Interna.

Apesar de concordarmos com as convenções utilizadas para escrever a documentação interna (veja seção 3.3.1), sugerimos alguns detalhamentos dos ítens 3 a 10 e propomos mudanças na ordem de apresentação de tais itens. Estes itens estão relacionados com a forma de escrever os

comentários e com o seu conteúdo.

Há duas classes de comentários em programas: os Comentários de Prólogo e os Comentários Explicativos. A escolha do conteúdo destes comentários é tarefa do programador. Diferente dos comentários tradicionalmente utilizados, cujas informações são irrelevantes em tempo de compilação (e por isso são desprezados pelos compiladores), no nosso caso serão utilizados comentários de dois tipos: os comentários constantes e os comentários variáveis.

Os comentários constantes são escritos da maneira convencional, começando com o caráter 'C' e contendo informações que não mudam para as várias versões de uma mesma unidade de programa.

Os comentários variáveis, identificados por 'C&', são comentários especiais cujas informações variam de versão para versão de uma mesma unidade de programa e serão consideradas pelo Sistema ao efetuar qualquer tipo de transformação.

Visto que a documentação interna das unidades de programa da biblioteca deve ser escrita levando em consideração sua transportabilidade, propomos algumas padronizações sobre a forma de escrevê-la para que a mesma possa também ser transportável com um mínimo de esforço. Os programadores que estiverem escrevendo software transportável deverão ter sempre em mente o princípio básico

de que deverão ser evitadas quaisquer referências a nomes de variáveis, de equipamentos, de precisão, ou de quaisquer outras entidades que sirvam, de alguma forma, para associar a unidade de programa a uma versão específica de unidade de programa, à exceção dos lugares destinados a tal finalidade.

(a) - Comentários de Prólogo.

Cada unidade de programa da biblioteca contém alguns comentários no início para explicar o que ela faz. Estes comentários são chamados de Comentários de Prólogo.

O programador deve incluir nos Comentários de Prólogo, no mínimo, como sugerido por Tassel [26], as seguintes informações, observadas as convenções de padronização que incluímos:

(1) Uma descrição do que a unidade de programa faz, escrita em comentários convencionais da linguagem FORTRAN;

(2) Versão: o programador deve incluir três linhas de comentários variáveis para identificar a versão da unidade de programa: a primeira linha contém "C& VERSÃO:", a segunda contém "C& EQUIPAMENTO:" seguido da identificação do equipamento utilizado, e a última contém "C& PRECISÃO:" seguido da precisão usada na versão. Por questões de

eficiência de execução decidiu-se pela utilização de comentários variáveis o que dispensa o Sistema de analisar um grande número de comentários (os comentários convencionais que, quase sempre, são maioria).

Quando a unidade de programa for submetida ao TRAPO para a geração de novas versões, o Sistema se encarrega de preencher os valores referentes a "EQUIPAMENTO" e "PRECISÃO" com informações obtidas do "menu" preenchido pelo usuário.

(3) Utilização: como chamar ou utilizar a unidade de programa. Deve ser incluída uma linha contendo "C& UTILIZAÇÃO:", seguida do exemplo de utilização, que deve ser colocado em uma nova linha de comentário variável começando com os caracteres "C&". Como esta é uma linha especial de comentário, caso necessite ser continuada, a linha de continuação também deverá começar com os caracteres "C&";

(4) Uma lista e descrição de todos os parâmetros. A lista deve ser antecedida por uma linha contendo "C& PARÂMETROS:" e deve conter, para cada parâmetro, uma linha começando com os caracteres "C&" seguidos do parâmetro e do seu tipo (acompanhado da dimensão ou dimensões, se aplicável) separados por pelo menos um espaço em

branco um do outro, seguida de uma ou mais linha(s) de comentários descrevendo o parâmetro. A lista descreve inicialmente os parâmetros que são passados para o subprograma e a seguir, os parâmetros que são retornados do subprograma. Na chamada, os parâmetros obedecem também a este sequenciamento. Sugere-se que a descrição de cada parâmetro seja iniciada na mesma posição de tabulação que aquela usada para iniciar o tipo do parâmetro;

(5) Uma lista dos subprogramas utilizados. A lista deve ser antecedida por uma linha contendo "C& SUBPROGRAMAS:". Este comentário deve ser iniciado com os caracteres "C&" e deve conter o nome e a classe de todos os subprogramas utilizados na unidade de programa (um por linha). Adicionalmente, cada nome de subprograma pode vir acompanhado, além da classe, também por uma breve descrição do que faz o subprograma, nesta ordem. Primeiro descrever os subprogramas externos e depois os intrínsecos que foram utilizados;

(6) O nome de qualquer método científico utilizado, acompanhado de referência sobre onde mais informações possam ser encontradas;

(7) Alguma indicação do tempo requerido para execução;

(8) Tamanho da unidade de programa:

-número de comandos;

-memória requerida;

(9) Requisitos especiais de operação;

(10) Autor; e

(11) Data em que a unidade de programa foi escrita ou revista.

Todas estas informações devem ser colocadas diretamente no corpo da unidade de programa para que possam ser encontradas facilmente. Assim, a unidade de programa será sua própria documentação.

(b) - Comentários Explicativos.

A segunda classe de comentários, os Comentários Explicativos, são inseridos na unidade de programa para explicar qualquer código fonte cujo entendimento não seja possível com a simples leitura do código fonte (detalhamento do item 10 da convenção adotada anteriormente). Também aqui, o programador deverá evitar referências a entidades que possam servir para associar a unidade de programa a uma determinada versão

da unidade de programa.

A quantidade e o conteúdo dos comentários são dois aspectos bastante importantes. Recomenda-se utilizar pelo menos um Comentário Explicativo para cada dez (10) linhas de código fonte.

Quanto ao conteúdo, recomenda-se ao programador que assuma a familiaridade do leitor com a linguagem FORTRAN e, portanto, os comentários devem explicar a finalidade de um grupo de comandos do programa, e não, descrever a operação dos comandos. Por exemplo, antes do comando:

```
      IF ( Q .NE. 0 ) GO TO 320,
```

um comentário como:

```
C       DESVIO SE Q DIFERENTE DE ZERO
```

não é um bom comentário pois tenta explicar a sintaxe do comando em vez de informar por que um desvio deve ser usado. Por outro lado, o comentário:

```
C
C       QUANDO O PARAMETRO DE MARQUARDT E ZERO
C       ATRIBUI-LHE O MENOR DOS VALORES DO TRAÇO E
C       NORMA INFINITA (MAXIMA SOMA DOS ELEMENTOS
C       DA LINHA) DA MATRIZ NORMAL.
C
```

tirado de uma unidade de programa da BITAN, onde aparece antes do comando acima, é um bom comentário pois informa porque um desvio deve ser usado.

Uma vez que propusemos algumas alterações para as Convenções de Formatação e para a Documentação Interna, e considerando que o Sistema deve ser aplicável à BITAN sem exigir modificações (objetivo 4), então há que se propor, também, uma rotina para transformar automaticamente a BITAN de modo a adequá-la à padronização proposta.

3.3.2. Rotina de Conversão Para a Padronização Proposta.

Para adequar as Convenções de Formatação à forma proposta, a rotina deverá incluir linhas em branco onde ainda não existe:

(1) após os desvios incondicionais e

(2) antes e depois de cada bloco de Comentários Explicativos.

A Rotina de Transformação da Documentação Interna atuará sobre os comentários iniciais de cada unidade de programa convertendo-os para a forma proposta de Comentários de Prólogo.

Os itens que aparecem como Comentários de Prólogo mas que não existiam anteriormente, ficarão parcialmente em branco. Por exemplo, o terceiro Comentário de Prólogo ficará sempre com a forma:

```
C&      3. UTILIZAÇÃO:
```

Posteriormente, à medida que as unidades de programa forem sendo submetidas a manutenções ou quando houver disponibilidade de pessoal, estes comentários serão completados pela equipe de manutenção.

3.3.4. Observações.

Com a série de convenções proposta para a padronização da documentação interna, espera-se obter unidades de programa com documentação interna transportável.

Os comentários de prólogo de tipos 2, 3, 4 e 5 são comentários variáveis (começam com "C&") e serão analisados pelo módulo do Sistema responsável pela conversão da

documentação interna.

Os demais comentários, inclusive os explicativos, são escritos da maneira convencional (começando com o caráter C) com a pequena restrição de não ter o C inicial seguido do caráter "&", e portanto, não serão levados em consideração ao se efetuar a conversão.

Se por alguma razão o programador sentir a necessidade de escrever nos comentários o nome de entidades que estejam relacionadas com uma versão específica de unidade de programa, isto só será permitido para os comentários de prólogo de tipos (3), (4) e (5), que são comentários variáveis, com a imposição de que as entidades sejam analisadas quanto à possibilidade de conversão - seus nomes, se possível, serão convertidos de acordo com o caráter inicial padrão para a precisão (conforme a seção 3.3.1).

Em última instância, se o programador tiver que efetuar referência em um comentário explicativo a nomes de entidades que estejam relacionadas com uma versão específica de unidade de programa, este comentário deverá ser um comentário variável e, portanto, será iniciado com "C&&" (para diferenciar dos comentários de prólogo que iniciam por "C&") e o(s) nome(s) da(s) referida(s) entidade(s) dever(á)(ão) vir antecedido(s) pelo caráter "&" em cada referência, para que o Sistema possa identificar as transformações a serem efetuadas.

3.4. OS COMPONENTES DO SISTEMA.

Nesta seção descrevemos os módulos básicos do Sistema e discutimos alguns itens considerados no trabalho de conversão, apontando possíveis soluções para os problemas encontrados.

O Sistema funciona como um pré-processador (opera somente com código fonte, não efetua geração de código) e, quando estiver totalmente implementado, será composto de quatro módulos, descritos nas próximas subseções:

(1) Módulo de Conversão de Precisão;

(2) Módulo de Conversão de Ambientes;

(3) Módulo de Substituição de Chamadas a Subprogramas por Código em Linha; e

(4) Módulo de Conversão da Documentação Interna.

A entrada para o Sistema tem a forma de deck base descrito em um subconjunto do FORTRAN-66 (definido por Ryder [22], com as convenções apresentadas na seção 3.1.1) juntamente com os comandos escritos usando a Linguagem Interativa de Seleção de Alternativas do Pré-processador (LISA), posteriormente descrita. As únicas verificações sintáticas que são efetuadas pelo Sistema estão relacionadas com os comandos escritos utilizando LISA.

A saída produzida pelo Sistema é ou um deck base ou um

deck de distribuição, dependendo das opções que forem especificadas nos comandos escritos em LISA.

Se o tempo de execução for um fator limitante, então algumas rotinas básicas, principalmente para manipulação de caracteres, devem ser re-codificadas em assembler.

A seguir, apresentamos uma descrição do funcionamento dos módulos do Sistema.

3.4.1. Módulo de Conversão de Precisão.

O Módulo de Conversão de Precisão executa quatro tipos de conversões de precisão:

(1) Aritmética Complexa de Precisão Dupla para Aritmética Complexa de Precisão Simples (Z --> C);

(2) Aritmética Complexa de Precisão Simples para Aritmética Complexa de Precisão Dupla (C --> Z), inversa da conversão tipo (1);

(3) Aritmética de Precisão Dupla para Aritmética de Precisão Simples (D --> S); e

(4) Aritmética de Precisão Simples para Aritmética de Precisão Dupla (S --> D), inversa da conversão tipo (3).

Os itens afetados por uma conversão de precisão geralmente são: declaração de tipo de variáveis, variáveis,

constantes decimais, funções intrínsecas e externas e declarações FORMAT. A Tabela 1 mostra as ações executadas em cada um de tais casos para as conversões citadas. Ao se efetuar a conversão de precisão de uma unidade de programa podem ocorrer várias situações que devem ser contornadas para que a nova versão da unidade de programa funcione normalmente. A seguir relacionamos estas situações e apresentamos possíveis soluções:

(a) Obtenção de um nome de função não existente após a conversão.

Solução:

Substituição da chamada por um desmembramento equivalente;

(b) Overflow no tamanho da linha de codificação (7 - 72) ocasionado, por exemplo, pela inclusão de um E em um expoente de uma constante decimal que não o possuía; ou pelo desmembramento de uma chamada, ocasionado pela inexistência de uma função cujo nome foi obtido após a conversão. Uma substituição de valor num comando DATA também pode ocasionar esse overflow.

Solução:

Utilização de linha de continuação;

CONVERSÃO	(1)	(2)	(3)	(4)
ITEM	Z --> C	C --> Z	D --> S	S --> D
Declaração de tipo das variáveis	Substitui DOUBLE COMPLEX por COMPLEX	Substitui COMPLEX por DOUBLE COMPLEX	Substitui DOUBLE PRECISION por REAL	Substitui REAL por DOUBLE PRECISION
Constantes Decimais	Substitui D por E, ou inclui E0 quando não encontrar o D, nas partes Real e/ou imaginária.	Substitui E por D, ou inclui D0 quando não encontrar o E, nas partes Real e/ou imaginária.	Substitui D por E, ou inclui E0 quando não for encontrado o D.	Substitui D por E, ou inclui D0 quando não for encontrado o E.
Chamadas a Funções Externas.	Altera nome da Função: substitui o Z inicial por C.	Altera nome da Função: substitui o C inicial por Z.	Altera nome da Função: substitui o D inicial por S.	Altera nome da Função: substitui o S inicial. por D.
Chamadas a Funcões Intrínsecas.	Substitui o nome da Função por seu equivalente da Tabela Mestre.	Substitui o nome da Função por seu equivalente da Tabela Mestre.	Substitui o nome da Função por seu equivalente da Tabela Mestre.	Substitui o nome da Função por equivalente da Tabela Mestre.
Declarações FORMAT			Substitui D por E nos códigos de formato	Substitui E por D nos códigos de formato

Tabela 1: Ações do Módulo de Conversão de Precisão.

(c) Irreversibilidade da conversão, ocasionada pela existência de expressões aritméticas do tipo misto no texto fonte.

Solução:

(i) Não utilização de expressões aritméticas do tipo misto nas unidades de programa, conforme seção 3.3.1 (solução adotada) ou

(ii) Elas devem ser explicitamente indicadas no texto fonte (exige marcadores especiais).

(d) Overflow na largura total para um dado dispositivo, ocasionado pelo aumento da largura de um código de formato.

Solução:

(i) Utilização de códigos de formato com dimensão máxima no código original, de modo que não seja necessária modificação no tamanho do campo (solução adotada); ou

(ii) Dividir o formato em vários registros;

(e) A documentação interna pode deixar de refletir a realidade.

Solução:

Conversão da documentação interna todas as vezes que efetuar uma conversão de precisão, conversão de ambientes ou otimização.

Uma vez que todas as variáveis são declaradas no início

das unidades de programa e são escritas de acordo com as Convenções de Nomenclatura (seção 3.3.1), o Sistema tem duas opções para identificar as variáveis a serem trocadas com a precisão: através das declarações de tipo ou através da letra inicial do nome da variável. O Sistema utiliza a primeira opção para transformar o tipo das variáveis, enquanto que a segunda opção é utilizada apenas para efeitos de verificação: quando encontra uma variável cujo nome não está padronizado de conformidade com a precisão da unidade de programa, o Sistema acusa esta ocorrência através de uma mensagem de erro.

Em uma chamada a subprograma, sempre que converte o nome de uma função intrínseca, o Sistema verifica a existência do novo nome pesquisando-o na Tabela Mestre. Se existe a função cujo nome foi obtido com a conversão, então o Sistema efetua a substituição normalmente. Em caso de inexistência da função cujo nome foi obtido com a conversão, então o Sistema substitui a chamada por um desmembramento equivalente (obtido também da pesquisa à Tabela Mestre). Isto ocorre quando, por exemplo, efetuando uma conversão do tipo (4) acima (S --> D), o Sistema encontra uma chamada a FLOAT ( ). Neste caso, o Sistema substitui o nome da função intrínseca (FLOAT) por um desmembramento equivalente (DBLE(FLOAT( )).

Todas as variáveis definidas nos comandos DATA são pesquisadas quanto à sua existência na Tabela Mestre. Para

aqueles identificadores encontrados na Tabela Mestre, o Sistema efetua uma substituição do valor correspondente no comando DATA pelo valor encontrado na Tabela Mestre e verifica a possibilidade de ocorrência de overflow da linha de codificação.

A aplicação em cadeia de conversões dos tipos (1) e (2) (ou (3) e (4)) alternadamente, a uma unidade de programa e às unidades de programa obtidas após as conversões subsequentes, deve produzir sempre unidades de programa computacionalmente equivalentes. Então, por exemplo, se X é uma unidade de programa utilizando aritmética de precisão dupla, a cadeia de conversões da figura 2, abaixo,

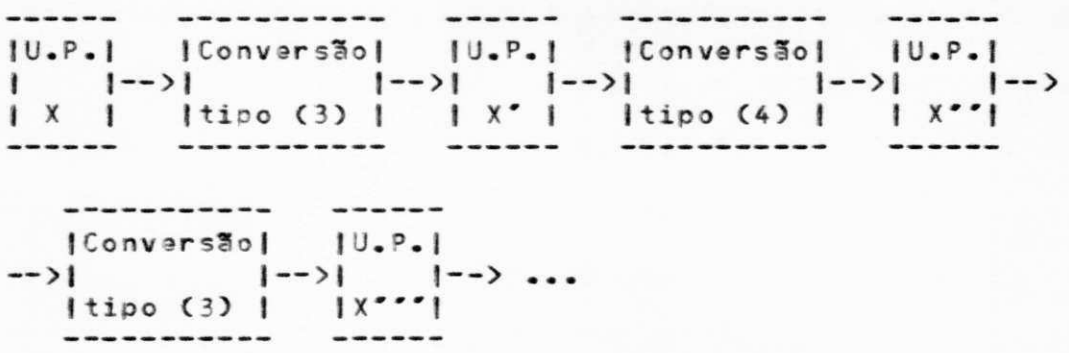


Figura 2 - Uma cadeia de conversões.

produz X'' computacionalmente equivalente a X, X''' computacionalmente equivalente a X', etc.; o que significa que a conversão de precisão é reversível. Este é um aspecto interessante do Sistema e foi utilizado para testar o funcionamento do módulo de conversão de precisão.

3.4.2. Módulo de Conversão de Ambientes.

O Módulo de Conversão de Ambientes converterá as unidades de programa adaptando-as para rodar em um novo ambiente que é especificado ao solicitar a conversão. Caso o ambiente destino especificado não esteja parametrizado na Tabela Mestre, o usuário receberá uma advertência e o Sistema cancelará a sessão de trabalho sem efetuar nenhuma transformação.

A transferência entre ambientes distintos será efetuada através da utilização de fitas magnéticas para o transporte dos programas fonte. As fitas magnéticas serão acompanhadas de um pequeno manual contendo informações sobre a instalação da BITAN no ambiente destino.

Algumas unidades de programa da BITAN fazem referências a determinadas grandezas (chamadas de parâmetros ambientais) que são dependentes do ambiente de computação. Os parâmetros ambientais mais comuns pertencem a três conjuntos (conjunto aritmético, parâmetros básicos de entrada/saída e parâmetros diversos) e foram descritos anteriormente na seção 3.3.1. Os nomes adotados foram sugeridos pelo IFIP WG 2.5 em junho de 1976 (veja [11] para maiores detalhes). As constantes dependentes do ambiente têm os seus valores definidos na Tabela Mestre para as várias máquinas e para as várias precisões, utilizando-se representação decimal para a definição.

Com relação às dependências de máquina que não possam ser parametrizadas, um programador que utilizar o TRAPO deve conhecer, em detalhes, o ambiente destino - para poder efetuar os ajustes finais que forem requeridos.

Uma vez que a versão de distribuição da BITAN será entregue ao usuário acompanhada de programas para teste, e de resultados demonstrativos de testes efetuados no ambiente origem, tornar-se-á fácil identificar possíveis problemas ocorridos com o transporte para o novo ambiente.

Os identificadores, ao serem encontrados pelo Sistema em um comando DATA, serão pesquisados quanto à sua existência na Tabela Mestre. Para aqueles identificadores que forem encontrados na Tabela Mestre (por serem parâmetros ambientais), haverá uma substituição do valor correspondente no comando DATA e será efetuada uma verificação quanto à possibilidade de ocorrência de overflow da linha de codificação.

Ocorrendo o overflow da linha de codificação, o Sistema utilizará linhas de continuação (iniciando sempre pelo caráter "T" (de TRAPO), para diferenciar das linhas de continuação utilizadas pelo programador e que iniciam por "X", ambos na coluna 6).

Os identificadores que não forem encontrados na Tabela Mestre possivelmente representam constantes matemáticas que, certamente, não precisam ser parametrizadas.

Em qualquer tempo, desde que haja uma necessidade comprovada para tal, as entradas da Tabela Mestre podem ser atualizadas para permitir a inclusão de novos parâmetros ou de parâmetros definindo uma nova máquina. No capítulo 4, ao descrevermos a Tabela Mestre, voltaremos a tratar deste assunto.

3.4.3. Módulo de Substituição de Chamadas a Subprogramas por Código em Linha.

Como estabelecido pelo objetivo (5), o Sistema deve prover, opcionalmente, um pouco de otimização. Esta otimização será conseguida com a substituição de chamadas a subprogramas por código em linha. Esta é uma conversão nem sempre disponível em sistemas como o nosso e quase sempre pode deixar de ser incluída numa primeira fase de desenvolvimento do Sistema, ficando a sua inclusão para uma fase em que o Sistema já se encontre em plena operação. Dentre os Sistemas estudados apenas o TAMPR dispõe desta facilidade (veja [5]).

É importante salientar que a utilização de chamadas a subprogramas, e mais especificamente, as chamadas a BLAS (BASIC LINEAR ALGEBRA SUBPROGRAMS) não é necessariamente ineficiente. Boyle & Matz [5] afirmam que "um breve estudo de Moler (Moler, C. B. "Timings with Basic Linear Algebra Modules", Argonne National Laboratory, Private Communication, Jun/1975) mostrou que as chamadas a BLAS

(versões em Linguagem Assembler, ou igualmente em alguns casos, versões FORTRAN) em conjunção com compiladores FORTRAN não altamente otimizadores (especialmente aqueles que efetuam a multiplicação implicada por subscritos duplos) são realmente mais eficientes do que código em linha".

As deficiências na implementação da passagem de parâmetros ou a verificação rigorosa dos limites de ARRAYS em alguns compiladores FORTRAN podem proibir completamente o uso de BLAS. Nestes casos, portanto, para obter unidades de programa transportáveis, a substituição deve ser feita para todas as chamadas a BLAS, independentemente de considerações sobre eficiência. Isto significa que a inclusão, no Sistema, do Módulo de Substituição de Chamadas a Subprogramas por Código em Linha é mandatória em tais situações.

Mesmo restringindo o conjunto de subprogramas a serem substituídos, àquele escrito pelo programador e mais os BLAS, há uma série de pontos que devem ser considerados quando se pensa em substituir chamadas a subprogramas por código em linha. Entre os pontos a serem ponderados na substituição, destacam-se:

(I) Quem decide sobre as chamadas a serem substituídas?

(II) Como é tomada a decisão de efetuar uma determinada substituição? e

(III)Como será efetuada a substituição?

Com relação aos pontos (I) e (II) acima, há três possibilidades:

(a) Por questões de simplicidade de implementação, pode ser decidido que todas as chamadas a subprogramas serão substituídas por código em linha equivalente. Esta abordagem tem desvantagens óbvias tais como, por exemplo, o aumento indiscriminado do tamanho do código fonte e consequentemente o aumento do espaço requerido pelo código objeto correspondente. Outra desvantagem da abordagem é o fato de não considerar o contexto onde ocorre a chamada para decidir, baseado em uma análise de vários itens do contexto, se a substituição traz benefícios ou não.

(b) Também por questões relacionadas com a simplicidade de implementação, pode ser decidido que o programador deverá informar ao Sistema todas as rotinas para as quais deve ser feita a substituição da chamada por código em linha. Temos então duas opções: ou o programador informa em comentários especiais o(s) nome(s) da(s) rotina(s) a ser(em) substituída(s), ou o programador identifica através da chamada, onde é desejada uma substituição. A primeira opção tem as mesmas desvantagens que o caso "a" acima. A segunda opção parte do pressuposto que o programador é esclarecido suficientemente a ponto de saber os

lugares da unidade de programa onde é vantajosa a substituição de uma chamada a subprograma pelo código em linha equivalente. A sua principal desvantagem é o aumento do número de tarefas do programador.

(c) Em um projeto mais ambicioso, a decisão sobre que rotinas e em que chamadas é vantajoso fazer a substituição da chamada por código em linha, fica a cargo do próprio Sistema. Antes de se decidir por efetuar uma substituição, o Sistema deverá fazer uma série de ponderações. Posteriormente, nesta mesma seção, discutiremos estas ponderações.

A opção "a" é aplicável à biblioteca existente sem que seja necessário efetuar quaisquer alterações, contudo, o aumento indiscriminado do código pode inviabilizar a substituição. Uma solução intermediária e mais racional, que elimina o problema do aumento indiscriminado do código, consiste em substituir por código em linha somente as chamadas às rotinas de BLAS - em equipamentos com pouca disponibilidade de memória esta opção ainda pode ocasionar transtornos e impedir a substituição. As chamadas às rotinas de BLAS podem ser localizadas facilmente pelo Sistema.

A opção "b" não é aplicável à biblioteca existente pois requer alterações nos códigos fonte das unidades de programa da biblioteca, e isto vai de encontro ao estabelecido pelo objetivo (4).

A opção "c" é aplicável à biblioteca existente mas exige uma análise bem ponderada de diversos fatores relativos ao contexto onde ocorre a chamada.

Visto que na opção "a" todas as chamadas serão substituídas, o ponto (II) aplica-se somente aos casos "b" e "c".

Nestes casos, a decisão de efetuar uma determinada substituição deve ser tomada com base na análise do contexto.

Na ponderação devem ser considerados de um lado o tamanho do corpo do subprograma e o overhead ocasionado por uma chamada a subprograma (este é um aspecto dependente de cada compilador o que dificulta ainda mais a análise) e do outro lado, a intensidade com que ocorre a chamada.

A análise do contexto, portanto, deverá considerar principalmente os laços. Há basicamente duas categorias de laços: os laços explícitos (escritos, por exemplo, com a utilização de um comando DO) e os laços embutidos. Os laços embutidos são formados com o auxílio de comandos tais como IF, GO TO e CONTINUE, podendo ser identificados através de uma análise do fluxo de controle da unidade de programa.

A análise do fluxo de controle da unidade programa, em que chamadas a subprogramas devem ser substituídas por código em linha, apesar de servir para localizar os laços (explícitos/embutidos), nem sempre é útil para identificar a

frequência com que as chamadas são executadas pois este é um aspecto que muitas vezes só é conhecido em tempo de execução.

Em resumo, as chamadas a subprogramas que ocorrem em laços são candidatas potenciais à substituição por código em linha.

A seguir, abordamos o ponto (III). Para viabilizar a substituição, o código fonte do subprograma deve estar disponível ou junto à unidade de programa a ser transformada ou em algum outro meio que permita pronto acesso.

O Módulo de Substituição de Chamadas a Subprogramas por Código em Linha atua basicamente sobre:

(a) declarações dos subprogramas (SUBROUTINE, FUNCTION);

(b) declarações de tipo, dimensão, ..., internas ao subprograma;

(c) comandos RETURN/END;

(d) chamadas a subrotinas; e

(e) chamadas a funções: eliminação de chamadas presentes em comandos IF e WRITE, e também em expressões aritméticas e atribuições, possivelmente com a utilização de variáveis auxiliares.

Muitos detalhes devem ser estudados pelo implementador do módulo de substituição. Entre eles, merecem destaque:

(a) A substituição de parâmetros que são ARRAYS ou elementos de ARRAYS. A utilização de EQUIVALENCE pode ajudar na solução deste problema desde que sejam mantidas certas restrições sobre os parâmetros;

(b) A maneira de identificar quais são os parâmetros de entrada e quais são os parâmetros de retorno. Não se pode, por exemplo, partir da suposição que em subrotinas todos os parâmetros são de entrada e de retorno ao mesmo tempo;

(c) O modo de substituir mais de uma chamada a uma mesma função em um único comando FORTRAN (por exemplo, em uma expressão aritmética) caso as funções possam ter efeitos colaterais; e

(d) O método utilizado para diferenciar variáveis homônimas do programa principal e de um subprograma. A ocorrência de homônimos significa que um único nome está sendo utilizado para identificar variáveis com finalidades distintas. Com relação à ocorrência de homônimos há duas situações a serem consideradas: primeiro, quando o Sistema será utilizado para auxiliar no desenvolvimento de uma biblioteca de software numérico, pode-se restringir o conjunto de nomes utilizáveis nos programas e subprogramas de modo que passem a ser conjuntos disjuntos - uma vez que o trabalho de diferenciar os nomes das variáveis fica a cargo do programador, esta é mais uma carga para ele e

deveria, como tal, ser eliminada. Segundo, quando o Sistema será aplicado a uma biblioteca já existente e que não levou em consideração a restrição citada para a primeira situação, pode-se alterar (na hora de efetuar a substituição) todos os nomes de variáveis do subprograma que não são parâmetros de subrotinas, de modo que eles passem a ser únicos (por exemplo, fazendo com que todos terminem por ZZ, XX, ...). Esta solução, além de restringir o conjunto de nomes utilizáveis como nomes de variáveis e implicar na utilização de nomes menos significativos (eles devem agora diferir nos quatro primeiros caracteres), para que seja confiável requer que seja feita uma análise, automática ou não, para detectar possíveis ocorrências de sinônimos após a transformação.

Se uma unidade de programa chama mais de um subprograma, então todos os subprogramas envolvidos em tais chamadas deverão ter nomes únicos.

Foi visto que não é interessante efetuar a substituição de todas as chamadas a subprogramas por código em linha o que nos leva a concluir que apenas algumas chamadas devem ser substituídas.

Sugerimos que seja feito um levantamento estatístico a partir de dados reais, caso o programador que irá identificar as chamadas a serem substituídas não seja um bom conhecedor das unidades de programa da biblioteca, para

então decidir que chamadas devem ser substituídas. O levantamento estatístico mostrará a frequência com que são executadas as chamadas incluídas em laços.

Para as situações em que nem o programador possa informar quais as chamadas a serem substituídas nem o Sistema possa identificá-las, sugerimos que todas as chamadas às rotinas de BLAS sejam substituídas por código em linha equivalente.

Para reduzir as probabilidades de ocorrência de sinônimos após a transformação, todos os identificadores da rotina cuja chamada será substituída, devem ser trocados por novos identificadores (diferentes de todos os identificadores gerados para as outras rotinas) gerados automaticamente.

O algorítmo para efetuar a geração de novos identificadores deve, se possível, gerar sempre um mesmo conjunto de identificadores cada vez que é executado para uma mesma rotina, caso ele deva ser executado a cada chamada ao módulo de substituição de chamadas.

Uma solução alternativa, seria executar a mudança de identificadores uma só vez para toda a biblioteca e manter uma cópia adicional de cada rotina com os identificadores já modificados. Isto tem a desvantagem de exigir uma área adicional de armazenamento, mas leva à obtenção de um algorítmo mais simples para a geração de novos

identificadores (poderá, por exemplo, gerar identificadores formando uma sequência) além de ser mais rápido de executar.

Outras informações sobre substituição de chamadas a subprogramas por código em linha e outros tópicos relacionados podem ser encontradas em [5], [18], [19] e [24].

3.4.4. Módulo de Conversão da Documentação Interna.

Ao referenciarmos o Módulo de Conversão da Documentação Interna, temos que diferenciar dois aspectos que lhe são inerentes. O primeiro está relacionado com a conversão da documentação interna, já existente nas unidades de programa da BITAN, para a padronização proposta. Esta conversão ocorrerá apenas uma vez, antes de passarmos a utilizar o TRAPO para produzir novas versões da BITAN, e já foi discutida na seção 3.3.2.

O segundo aspecto diz respeito à conversão da documentação interna requerida por algum dos módulos do TRAPO, quando da sua aplicação para a produção de novas versões da biblioteca e será discutido a seguir.

O Módulo de Conversão da Documentação Interna analisará os comentários de prólogo, de tipos (2), (3), (4) e (5), e os comentários explicativos que iniciem por "C&&" quanto à necessidade de conversão.

Uma vez que os comentários explicativos geralmente não

contêm referências a entidades que identificam uma versão específica de unidade de programa (exceto aqueles iniciados por "C&&"), a tarefa de conversão da documentação interna será bastante simplificada.

Em todos estes comentários (especiais) o Sistema tentará identificar a entidade a ser convertida (vem antecedida por "&") e atuará de acordo com as ações especificadas na Tabela 2.

TIPO DE ENTIDADE	AÇÃO DO SISTEMA
Identificador	Se for possível, converte o nome do identificador de acordo com as "Convenções de Nomenclatura" (veja seção 3.3.1), da versão anterior para a versão requerida.
Constante	Se não é do tipo inteiro nem de caracteres, converte a sintaxe da constante, p. ex., 0.5E0. para 0.5D0.
Nome de Precisão: -Precisão Dupla -Precisão Simples -Complexo de Precisão Dupla, Complexo de Dupla Precisão ou Comp_Prec_Dup -Complexo de Precisão Simples, Complexo de Simples Precisão ou Cmp_Prec_Simp	Substitui o nome da precisão anterior (se pertencente a um dos tipos ao lado) pelo nome da precisão requerida, utilizando as seguintes notações para o nome da nova precisão: -Precisão Dupla (para DOUBLE PRECISION) -Precisão Simples (para REAL) -Comp_Prec_Dup (para DOUBLE COMPLEX) -Cmp_Prec_Simp (para COMPLEX)
Todos os demais	Nenhuma (O texto fica inalterado)

Tabela 2: Ações do Módulo de Conversão da Documentação Interna.

Como pode ser visto na Tabela 2, se a entidade que veio

antecedida por "&" no texto fonte não pode ser classificada como Identificador, Constante ou Nome de Precisão (escrito em uma das formas mostradas na Tabela) então o Sistema deixará o texto fonte como em sua forma original.

Mas, qualquer que seja o caso, o Sistema deixará o comentário iniciando por "C&&", e deixará o "&" antes das entidades, para que novas versões da unidade de programa possam ser produzidas a partir da que está sendo preparada, se vierem a ser requeridas. Isto possibilita a reversibilidade de conversão da documentação interna para uma forma "equivalente", sempre que necessário.

4. DESCRIÇÃO DOS PROGRAMAS E IMPLEMENTAÇÃO.

4.1. INTRODUÇÃO.

Neste capítulo apresentamos uma breve descrição dos programas que formam o TRAPO e fornecemos detalhes de sua implementação.

4.1.1. Generalidades.

A figura 3 mostra a estrutura do Sistema e seus programas, e dá, entre parêntesis, uma indicação da seção onde serão definidos.

Alguns destes programas serão implementados de imediato e a linguagem de programação utilizada será PASCAL.

Os programas que serão desenvolvidos no início serão: o Supervisor, o Scanner, a rotina de conversão de precisão (D --> S, S --> D), a rotina para efetuar a conversão da documentação interna, o Formatador e o Gerador. Foram escolhidos estes programas para desenvolvimento imediato por serem suficientes para termos uma idéia global das ações, capacidades e limitações do Sistema.

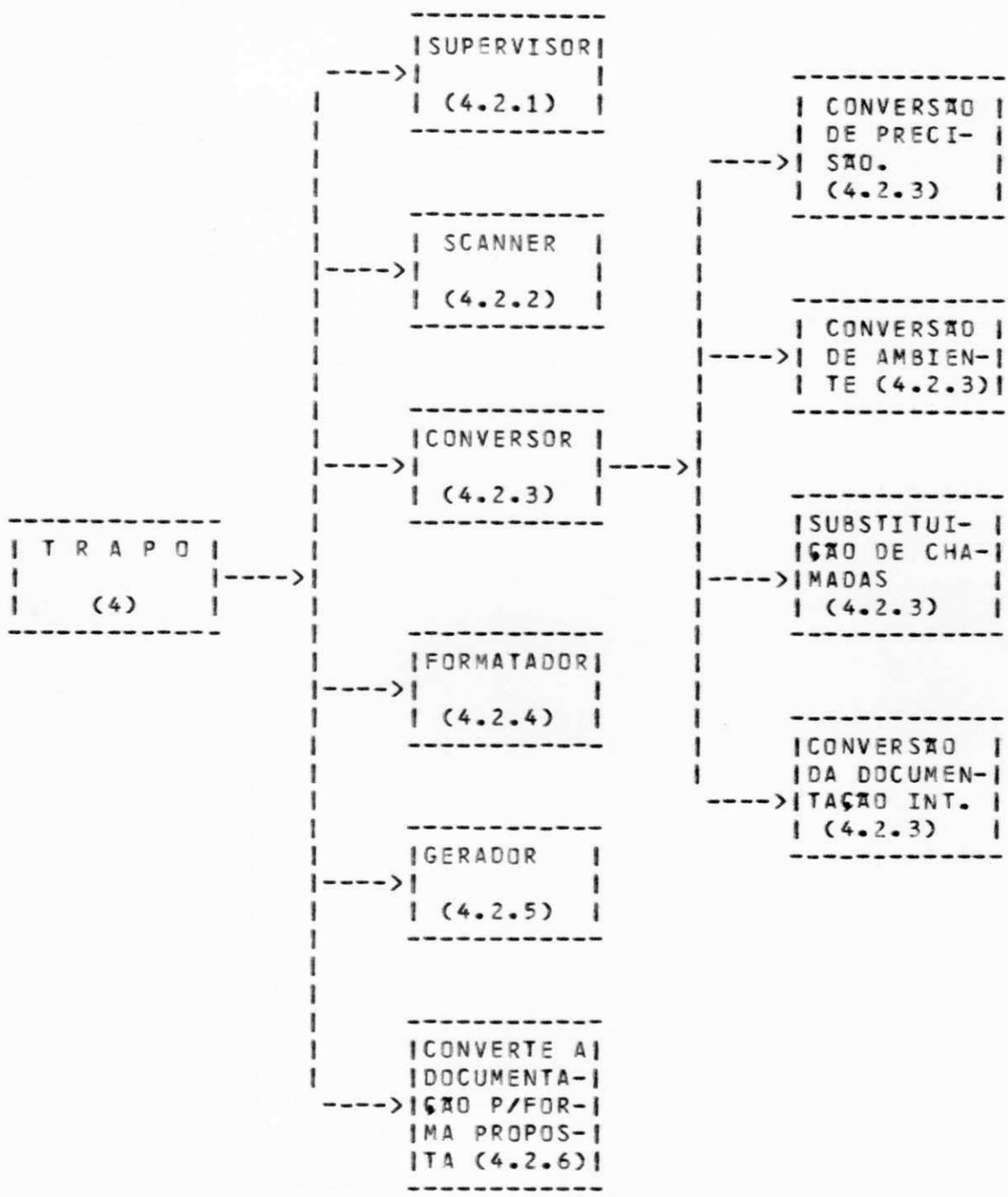


Figura 3 - Programas do Sistema Transformador de Programas FORTRAN (TRAPO).

Nas duas seções que seguem (4.1.2 e 4.1.3) descrevemos a Linguagem de Seleção de Alternativas (LISA) e a Tabela Mestre, respectivamente.

4.1.2. A LISA.

A Linguagem Interativa de Seleção de Alternativas do pre-processador (LISA) é uma linguagem bastante simples e será utilizada pelos usuários do TRAPO para comandar a transformação das unidades de programa.

Uma vez que o Sistema deve ser aplicável à BITAN sem modificações, as ações do TRAPO serão controladas por comandos externos às unidades de programa. Esta abordagem, além de não obscurecer o texto fonte, tem a vantagem de simplificar a geração de decks de distribuição.

Para solicitar algum tipo de transformação, após a ativação do Sistema, o usuário deverá preencher os campos vazios de um "menu" que lhe é apresentado pelo Sistema.

A tarefa de preenchimento do "menu" é o que chamamos de uma sessão de trabalho.

Em qualquer instante de uma sessão de trabalho o usuário poderá solicitar ao sistema que:

(1) Lhe seja mostrada uma tela com a situação corrente da sessão, contendo as informações já preenchidas até aquele instante, antes de continuar a apresentação do menu - para tanto, o usuário deverá digitar o caráter '?' ou

(2) Anule a sessão corrente e reinicie a apresentação do "menu" para uma nova sessão de trabalho - para fazer isto, o usuário deverá digitar o caráter '!'.

Sempre que anular uma sessão de trabalho, o usuário será questionado quanto ao desejo de continuar utilizando o TRAPO - responderá S (sim) ou N (não).

Quando o usuário completar o preenchimento do "menu", o sistema lhe apresenta uma tela, com o formato mostrado na figura 4, com todas as informações referentes à sessão corrente.

```
-------------------------------------------------
|  SISTEMA TRANSFORMADOR DE PROGRAMAS FORTRAN.   |
|                                                |
|  USUARIO:________                              |
|  UNIDADE DE PROGRAMA:________                  |
|  VERSAO ATUAL.                                 |
|     EQUIPAMENTO:                               |
|        01 - IBM-4341      02 - COBRA-530       |
|        03 - DEC-10                             |
|        CODIGO:__                               |
|     PRECISAO:                                  |
|        1 - Precisão Simples                    |
|        2 - Precisão Dupla                      |
|        3 - Complexo de Precisão Simples        |
|        4 - Complexo de Precisão Dupla          |
|        CODIGO:_                                |
|  VERSAO REQUERIDA.                             |
|     CODIGO DO EQUIPAMENTO:__                   |
|     CODIGO DA PRECISAO   :_                    |
|     DISTRIBUIÇAO (S?, N?):_                    |
|     OTIMIZAÇAO   (S?, N?):_                    |
-------------------------------------------------
```

Figura 4 - Exemplo de uma tela completa do TRAPO.

Os campos do "menu" devem ser preenchidos observando-se as regras fornecidas a seguir.

USUARIO:

Preencher com o nome do usuário com o máximo de oito (8) caracteres alfabéticos e/ou numéricos, iniciando por letra.

SENHA:

Deve ser preenchido com a senha do usuário, que terá no máximo oito (8) caracteres. Se preenchido com uma informação incorreta então o usuário será notificado e o Sistema será desativado.

UNIDADE DE PROGRAMA:

Informar a unidade de programa a ser transformada. Para indicar uma tarefa a ser executada para toda uma biblioteca, informar BIBLIO - esta opção não estará disponível inicialmente. Se preenchido com uma informação incorreta então o usuário será notificado e deverá preencher o campo com nova informação.

VERSAO ATUAL.

EQUIPAMENTO:

Informar o código do equipamento (com dois dígitos, se numérico) em que se encontra a unidade de programa, de acordo com as opções mostradas no "menu". Inicialmente, o Sistema disporá das seguintes opções para equipamento:

01 - IBM-4341

02 - COBRA-530

03 - DEC-10

? - Mostre a situação corrente

! - Anule esta sessão de trabalho.

Se preenchido com uma informação incorreta então o usuário será notificado e deverá preencher o campo com nova informação.

PRECISÃO:

Informar o código de identificação da precisão utilizada na unidade de programa de acordo com as opções mostradas no "menu". A seguir temos uma relação das opções para PRECISÃO:

1 - Precisão Simples

2 - Precisão Dupla

3 - Complexo de Precisão Simples

4 - Complexo de Precisão Dupla

? - Mostre a situação corrente

! - Anule esta sessão de trabalho.

Se preenchido com uma informação incorreta então o usuário será notificado e deverá preencher o campo com nova informação.

VERSÃO REQUERIDA.

CÓDIGO DO EQUIPAMENTO:

Informar o código do equipamento (com dois dígitos, se numérico) para onde será transportada a unidade de programa, de acordo com as opções mostradas no "menu". As opções disponíveis inicialmente serão:

01 - IBM-4341

02 - COBRA-530

03 - DEC-10

? - Mostre a situação corrente

! - Anule esta sessão de trabalho.

Se preenchido com uma informação incorreta então o usuário será notificado e deverá preencher o campo com nova informação.

CÓDIGO DA PRECISÃO:

Informar o código de identificação da precisão a ser utilizada na unidade de programa, de acordo com as opções mostradas no "menu". São os seguintes os códigos de identificação:

1 - Precisão Simples

2 - Precisão Dupla

3 - Complexo de Precisão Simples

4 - Complexo de Precisão Dupla

? - Mostre a situação corrente

! - Anule esta sessão de trabalho.

DISTRIBUIÇÃO ('S', 'N', '?', '!'):

Preencher com o caráter 'S' (de Sim) caso deseje obter um deck de distribuição como saída do Sistema e 'N' (de Não), caso contrário. Preencha com '?' se desejar que o Sistema mostre a situação corrente e '!' para que o Sistema anule toda a sessão corrente de trabalho. Qualquer outro caráter será rejeitado pelo Sistema.

OTIMIZAÇÃO ('S', 'N', '?', '!'):

Preencher com o caráter 'S' caso deseje executar a rotina de substituição de chamadas a subprogramas por código em linha e 'N', caso contrário. Preencha com '?' se desejar que o Sistema mostre a situação corrente e '!' para que o Sistema anule toda a sessão corrente de trabalho. Qualquer outro caráter será rejeitado pelo Sistema.

4.1.3. A Tabela Mestre.

A Tabela Mestre é uma estrutura de dados do Sistema em que são armazenadas várias informações que serão utilizadas nas conversões. São informações da Tabela Mestre: os parâmetros ambientais para cada ambiente de computação, os nomes das funções internas disponíveis nas várias precisões para os vários ambientes, os nomes das funções desmembráveis e seus respectivos desmembramentos e também uma lista de todos os identificadores declarados na unidade de programa a ser convertida.

A Tabela Mestre foi definida internamente como um vetor com dez (10) intervalos, cada intervalo com N_IDENTIFICADORES entradas, cada entrada sendo do tipo STRING com dezesseis (16) posições. Nos intervalos de um a quatro estão armazenados os nomes das funções internas que operam com aritmética complexa de dupla precisão, aritmética complexa de precisão simples, aritmética de precisão dupla e aritmética de precisão simples, respectivamente. Nos intervalos de seis a nove estão armazenados os identificadores da Unidade de Programa, de acordo com a precisão de cada um deles e na mesma sequência de precisão que aquela adotada para as funções internas.

Esta estrutura tem desvantagens óbvias tais como, por exemplo, a má utilização do espaço reservado, uma vez que cada intervalo é dimensionado pelo intervalo máximo requerido, mas facilita grandemente o trabalho de conversão

dos nomes das funções internas pois o nome da nova função é obtido sempre na mesma posição em relação ao início do intervalo para a nova precisão. Deste modo, uma função interna que ocupa, por exemplo, a quinta posição no terceiro intervalo (precisão dupla), tem a sua correspondente na quinta posição do quarto intervalo (precisão simples), e vice-versa.

Algumas funções internas não possuem um correspondente direto na nova precisão. Neste caso, a entrada da posição correspondente no intervalo desejado fica em branco e o nome da função aparece no quinto intervalo (funções que requerem desmembramentos). As entradas nas posições correspondentes no intervalo 10 são os desmembramentos equivalentes às funções em pauta.

Os intervalos cinco e dez existem devido à impossibilidade de se colocar os desmembramentos nas posições correspondentes nos intervalos desejados: uma vez que um desmembramento sempre contém mais de um símbolo, ele não poderia ser localizado quando fosse invertido o sentido da conversão.

Quando da implementação da Rotina de Conversão de Ambientes, a Tabela Mestre poderá ser ampliada transformando-se em uma matriz com várias entradas, uma para cada equipamento - cada entrada com a forma da Tabela atual. Será também necessário incluir na nova Tabela Mestre os intervalos relativos às várias constantes para os diversos

ambientes computacionais.

4.2. OS PROGRAMAS DO SISTEMA.

4.2.1. O Supervisor.

Entrada:

Comandos escritos na LISA.

Processamento:

Gerenciamento de todo o processo de conversão. É o Supervisor que ativa os vários módulos na sequência necessária para produzir a conversão requerida. Efetua também a análise dos comandos escritos na LISA para detectar possíveis erros de sintaxe ou opções inválidas. É capaz de localizar os seguintes tipos de erro:

(1) Usuário não autorizado a requerer transformações;

(2) Senha inválida;

(3) Unidade de programa inexistente;

(4) Equipamento não parametrizado;

(5) Precisão atual inválida;

(6) Precisão requerida não disponível ou inválida;

(7) Código de distribuição inválido;

(8) Código de otimização inválido;

(9) Conversão requerida não disponível;

(10) Código de continuação inválido;

(11) Unidade de Programa fora dos padrões estabelecidos - RESULTADOS IMPRECISOS;

(12) Limite da Tabela de Identificadores excedido: aumentar o valor de N_IDENTIFICADORES. EXECUÇÃO INTERROMPIDA!; e

(13) Precisão Atual Informada diverge da Precisão Atual da Unidade de Programa.

Saída:

Chamadas às diversas rotinas do Sistema ou mensage(m)(ns) de erro.

4.2.2. O Scanner.

É o responsável pela produção do programa fonte na forma interna. Utiliza a procedure interna "TOKEN" do PASCALVS para identificar os vários símbolos do programa fonte. A identificação dos símbolos segue a Tabela de Representação Interna apresentada a seguir:

REPI	SIGNIFICADO
01	Branco
02	Identificador
03	Inteiro s/ sinal
10	+
11	-
12	*
13	/
14	=
20	(
21	)
22	'
23	&
24	&&
25	.
26	,
27	:
30	Comentário
31	Continuação
32	Rótulo
33	Fim de Linha ('\')
99	Outros

Entrada:

Programa fonte escrito utilizando FORTRAN segundo a padronização proposta.

Processamento:

Lê o texto fonte, caráter a caráter, e para cada token identificado produz como saída um par ordenado (representação interna, símbolo). O conjunto de todos os pares ordenados, inclusive os referentes às várias partes dos comentários (serão utilizados pelo módulo de conversão da documentação interna, do Conversor) é o que chamamos de Programa Fonte na Forma Interna. O scanner inclui no programa fonte na forma interna também os pares ordenados referentes a indicadores de fim de linha (serão utilizados pelo Formatador).

Saída:

Programa fonte na forma interna.

4.2.3. O Conversor.

É formado pelas rotinas responsáveis pela Conversão de Precisão, Conversão de Ambientes, Conversão Automática da Documentação Interna e Substituição de Chamadas a Subprogramas por Código em Linha.

Entrada:

(a) Programa fonte na forma interna;

(b) Parâmetros informando a conversão (fornecidos pelo Supervisor); e

(c) Informações da Tabela Mestre.

Processamento:

Ativa a rotina de conversão requerida pelos comandos escritos em LISA, para produzir o Programa Interno Modificado (na versão solicitada). O Programa Interno Modificado é também um Programa Fonte na Forma Interna.

Saída:

Programa interno modificado.

I - Rotina de Conversão de Precisão.

Entrada:

(a) Programa fonte na forma interna;

(b) Parâmetros informando a conversão; e

(c) Informações da Tabela Mestre.

Processamento:

Converte a precisão da unidade de programa da precisão atual para a precisão requerida de acordo com as ações especificadas na Tabela de Ações do Módulo de Conversão

de Precisão (veja seção 3.4.1).

Saída:

Programa interno modificado (em uma nova precisão).

II - Rotina de Conversão de Ambiente.

Entrada:

(a) Programa fonte na forma interna;

(b) Parâmetros informando a conversão; e

(c) Informações da Tabela Mestre.

Processamento:

Converte a unidade de programa adaptando-a para rodar no equipamento requerido. Atua de acordo com o que foi especificado na seção 3.4.2.

Saída:

Programa interno modificado (adaptado para rodar em um novo ambiente).

III - Rotina de Substituição de Chamadas a Subprogramas por Código em Linha.

Entrada:

Programa fonte na forma interna.

Processamento:

A rotina se encarrega de substituir chamadas a subprogramas por código em linha e o processamento a ser efetuado dependerá dos critérios adotados para a escolha do tipo de substituição. A seção 3.4.3 apresenta uma discussão detalhada de tais critérios e deve ser consultada.

Saída:

Programa interno modificado (otimizado).

IV - Rotina de Conversão da Documentação Interna.

Entrada:

(a) Programa fonte na forma interna;

(b) Parâmetros informando a conversão; e

(c) Informações da Tabela Mestre.

Processamento:

Converte a documentação interna das unidades de programa de modo que passe a refletir a situação atual da unidade de programa. Atua conforme as especificações feitas na Tabela de Ações do Módulo de Conversão da Documentação Interna (veja seção 3.4.4).

Saída:

Programa interno modificado (com a documentação interna atualizada).

4.2.4. O Formatador.

Entrada:

Programa interno modificado.

Processamento:

O Formatador utiliza o programa interno modificado (conjunto modificado de pares ordenados) para produzir o Programa Fonte Modificado. O programa fonte modificado é um programa FORTRAN equivalente ao programa original mas, em uma nova versão.

Saída:

Programa fonte modificado.

4.2.5. O Gerador.

Entrada:

Programa fonte (em FORTRAN) ou programa fonte modificado (em FORTRAN).

Processamento:

O Gerador (gera deck de distribuição) elimina todos os &s presentes no texto de entrada para a produção de um

deck de distribuição correspondente. (Assemelha-se ao programa UNDOC [23] e aos programas geradores de deck de distribuição de alguns dos sistemas estudados).

Saída:

Deck de distribuição.

4.2.6. Módulo Auxiliar: Converte Documentação Interna para a Forma Proposta.

Entrada:

Programa fonte na forma interna.

Processamento:

O Módulo tenta converter, para a forma proposta, a documentação interna existente na unidade de programa. Atua sobre os comentários iniciais da unidade de programa convertendo-os para a forma de comentários de prólogo, e efetua a inclusão de linhas em branco. A seção 3.3.2 deve ser consultada sobre os detalhes destas operações.

Saída:

Programa interno modificado.

5. CONCLUSÃO.

Iniciamos este último capítulo fazendo uma análise dos resultados que foram obtidos com a utilização de uma versão ainda experimental do TRAPO. Nas seções seguintes relacionamos dificuldades encontradas durante a implementação do protótipo e apresentamos sugestões de tópicos para trabalhos posteriores.

Dentre os componentes do TRAPO atualmente estão implementados o Supervisor, o Scanner, o Gerador e duas das quatro rotinas do Conversor (responsáveis pela conversão de precisão e conversão da documentação interna, respectivamente) o que possibilita efetuar conversões automáticas de precisão.

5.1. ANÁLISE DE RESULTADOS OBTIDOS E PROCEDIMENTOS ADOTADOS DURANTE OS TESTES.

O funcionamento do TRAPO foi testado tirando-se proveito da sua característica de produzir conversões reversíveis (veja a seção 3.4.1). Assim sendo, a partir da Unidade de Programa "BALANC" da BITAN, à qual chamaremos de versão 1, foi produzida uma Unidade de Programa em uma nova precisão (versão 2) e, posteriormente, esta nova versão foi convertida para a precisão originalmente utilizada,

obtendo-se assim a versão 3. A versão 3, que tem precisão equivalente à da versão 1, foi transformada para a obtenção da versão 4, que possui a mesma precisão da versão 2. Para concluir, a versão 5 foi produzida a partir da versão 4, com precisão equivalente à das versões 1 e 3.

Utilizando-se o programa utilitário "COMPARE" (localiza registros diferentes em dois arquivos) do Sistema CMS da máquina utilizada, constatamos, como era esperado, que tanto as versões 2 e 4 (precisão simples, após a transformação) como as versões 3 e 5 (precisão dupla, após a transformação) se equivalem respectivamente. Por outro lado, a comparação da versão 1 com as versões 3 e 5 (que são semelhantes) serviu para mostrar que:

- Os nomes das rotinas passam a figurar na padronização estabelecida;

- As novas rotinas possuem, nos comentários de Prólogo, o nome do equipamento e da precisão utilizados, mesmo quando não existentes na versão original;

- Todas as constantes decimais possuem expoente, mesmo quando não existente na versão original;

- Algumas vezes é gerada uma linha de continuação por causa da inclusão de expoentes, ou por causa da mudança no nome da precisão; e

- que a documentação interna escrita nos comentários variáveis é sempre automaticamente mantida atualizada.

Outrossim, queremos salientar que não foram feitos testes com as rotinas transformadas utilizando valores pois, se por um lado eles eram dispensáveis - as rotinas produzidas por derivações sucessivas para uma mesma precisão eram semelhantes, por outro lado, os que seriam justificáveis eram de difícil implementação - a mudança na precisão ocasionará possíveis alterações nos resultados a serem obtidos e para efetuar uma análise mais precisa dos resultados obtidos será requerido bastante tempo e dedicação de um profissional da área de análise numérica. Fica, portanto, como sugestão, a implementação de baterias exaustivas de testes com valores para comprovar a validade das conversões de precisão efetuadas.

São apresentados no Anexo II, na sequência em que foram produzidas, as várias versões da rotina testada, com a última listagem correspondendo ao deck de distribuição gerado. O Anexo II termina com uma listagem de alguns erros obtidos, propositalmente, tentando utilizar o TRAPO.

5.2. DIFICULDADES ENCONTRADAS / PONTOS A MELHORAR.

A aplicação, em cadeia, de uma série de conversões da aritmética utilizada de uma precisão para outra e desta para a anterior, em uma mesma Unidade de Programa, mostra que o TRAPO produz sempre Unidades de Programa computacionalmente

equivalentes quando se inverte o sentido da conversão, e isto faz com que os resultados obtidos sejam bastante confiáveis.

Há, no entanto, uma situação em que as transformações não produzem Unidades de Programa computacionalmente equivalentes: quando a Unidade de Programa a ser transformada operar com valores em mais de uma precisão simultaneamente (por exemplo, alguns cálculos em precisão simples e outros em precisão dupla, o que exigiria quádrupla precisão) - mesmo assim, o usuário é informado desta ocorrência através do "ERRO 13: Precisão Atual Informada diverge da Precisão Atual da Unidade de Programa". Apesar de esta ser uma situação que raramente ocorre, este aspecto do Sistema deveria ser revisto futuramente.

O projeto da estrutura de dados da Tabela Mestre poderá ser revisto para que se consiga um melhor aproveitamento de espaço, possivelmente com a utilização de algorítmos de "hashing" eficientes.

Merecem também ser revisados dois itens relativos à comunicação do TRAPO com o Sistema Operacional da máquina em que ele foi desenvolvido. Nesta primeira versão fomos incapazes de fazer com que o TRAPO efetuasse a leitura da senha do usuário sem que ela ficasse exposta na tela do terminal. O segundo item relaciona-se com a Unidade de Programa a ser transformada: ela sempre deve estar em um arquivo cujo nome é "FILE" e cujo tipo é "ENTRADA" - ainda

não conseguimos fazer com que o Sistema leia diretamente o arquivo especificado pelo usuário em "Unidade de Programa". Atualmente o campo "UNIDADE DE PROGRAMA" serve apenas para verificar se a Unidade de Programa existe na biblioteca. Como consequência disto, a opção "BIBLIO", para conversão de toda a biblioteca, ainda não pôde ser implementada. Por outro lado, após a conversão, o programa modificado fica sempre em um arquivo cujo nome é "FILE" e cujo tipo é "PROG_MOD". Quando é solicitado um deck de distribuição, ele fica disponível em um arquivo chamado de "FILE" com tipo "DECK_DIS". Estes fatos devem sempre ser lembrados pelos usuários do TRAPO ao efetuar uma conversão.

5.3. SUGESTÕES PARA TRABALHOS POSTERIORES.

Vimos que a versão atualmente implementada do TRAPO funciona como um conversor automático de precisão. Para completar o Sistema ainda devem ser desenvolvidos: - o Módulo Auxiliar, para converter a documentação interna das Unidades de Programa para a forma que foi proposta; a Rotina de Substituição de Chamadas a Subprogramas por Código em Linha e a Rotina de Conversão de Ambientes, a qual reduzirá em grande escala o trabalho necessário para transportar a Biblioteca para um novo ambiente computacional. Uma vez que a Rotina de Conversão de Ambientes frequentemente deverá ser executada para novos ambientes, é aconselhável que ela seja fácil de modificar para incluir os parâmetros relativos aos novos ambientes computacionais. Uma vez que estas rotinas

fazem parte do TRAPO, o seu desenvolvimento deve levar em consideração os objetivos que foram estabelecidos no início do projeto (veja seção 3.2).

Para garantir a reversibilidade das conversões é que as constantes definidas nos comandos DATA permanecem com seus valores originais intocáveis após a conversão de precisão. Para assegurar que o número de dígitos esteja sempre de acordo com a precisão para a máquina utilizada, ao implementar a Rotina de Conversão de Ambientes deve ser incluído algum mecanismo para a substituição automática da definição de constantes nos comandos DATA.

Com vistas à obtenção de uma padronização ainda mais rigorosa, duas rotinas complementares serão aqui sugeridas: a primeira, para efetuar a formatação das Unidades de Programa obedecendo aos critérios de endentação sugeridos anteriormente (o Formatador conserva a formatação original da Unidade de Programa), e a segunda, para efetuar a geração, em ordem numérica crescente, dos números de comandos e a consequente substituição de todas as suas ocorrências. Esta última rotina deverá receber como parâmetros o valor inicial (menor rótulo) e o valor do incremento a ser adotado na numeração dos comandos.

Em uma fase posterior de desenvolvimento, o TRAPO poderá ser adaptado para que possa transformar Unidades de Programa que foram escritas utilizando FORTRAN mas que não obedeceram à padronização estabelecida inicialmente.

A Rotina de Conversão de Precisão futuramente também poderá ser modificada de modo que possa converter Unidades de Programa de uma precisão qualquer para qualquer outra precisão.

Um outro aspecto que também deve ser estudado diz respeito à inclusão (discutível) de facilidades que permitam a utilização de aritmética do tipo misto e de mais de um tipo de precisão por Unidade de Programa, mantendo a atual reversibilidade das Conversões.

5.4. COMENTÁRIOS FINAIS.

A utilização, em fase experimental, na conversão da BITAN (desenvolvida obedecendo à padronização citada na seção 3.3.1) mostrou que o TRAPO:

- quase não exige treinamento de seus usuários;
- simplifica a conversão dos programas de uma precisão para outra;
- é aplicável à biblioteca existente: visto que as adaptações necessárias podem ser feitas por um programa;
- converte a documentação interna da Unidade de Programa adaptando-a à nova versão produzida;

- é relativamente rápido ao efetuar as conversões;

- foi implementado em pouco mais de 90 dias; e

- é fácil de utilizar.

Isto atesta que, com exceção dos objetivos 3 e 5, todos os demais objetivos estabelecidos no início do projeto foram alcançados. O objetivo 3 (prover um modo de simplificar a conversão dos programas para utilização em outras máquinas) deve ser alcançado quando for implementada a Rotina de Conversão de Ambientes, enquanto que o objetivo 5 (prover um pouco de otimização) deve ser atingido com a implementação da Rotina de Substituição de Chamadas a Subprogramas por Código em Linha. Quando estas duas rotinas estiverem implementadas, o TRAPO estará concluído e certamente os objetivos do projeto terão sido atingidos em sua totalidade.

REFERÊNCIAS.

[1] Aird, Thomas J. - The IMSL FORTRAN Converter: An Aproach to Solving Portability Problems, Portability of Numerical Software, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, pp. 369-388.

[2] Aird, Thomas J.; Battiste, Edward L.; Gregory, Walton C. - Portability of Mathematical Software Coded in FORTRAN, ACM Trans. Math. Software, 3 (1977), pp. 113-127.

[3] Bentley, J. & Ford, Brian - On the Enhancement of Portability Within the NAG Project - A Statistical Survey, Portability of Numerical Software, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, pp. 505-528.

[4] Boyle, James M. - Mathematical Software Transportability Systems - Have the Variations a Theme?, Portability of Numerical Software, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, pp. 305-360.

[5] Boyle, J. & Matz, M. - Automating Multiple Program Realizations, Symposium on Computer Software Engineering, Polytechnic Institute of New York - Apr/1976, pp. 421-456.

[6] Brown, W. S. & Hall, A. D. - FORTRAN Portability via Models and Tools, Portability of Numerical Software, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, pp. 158-164.

[7] Cody, W. J. - Machine Parameters for Numerical Analysis, Portability of Numerical Software, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, pp. 49-67.

[8] Dekker, T. J. - Machine Requirements for Reliable, Portable Software, Portability of Numerical Software, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, pp. 22-36.

[9] Dritz, Kenneth W. - Multiple Program Realizations Using the TAMPR System, Portability of Numerical Software, W. Cowell (ed.), Brook, Illinois, Jun/1976, pp. 405-423.

[10] Du Croz, J. J.; Hague, S. J.; Siemieniuch, J. L. - Aids to Portability within the NAG Project, Portability of Numerical Software, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, pp. 389-404.

[11] Ford, Brian - Preparing Conventions for Parameters for Transportable Numerical Software, Portability of Numerical Software, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, pp. 68-91.

[12] Ford, Brian; Bentley, J.; Du Croz, J. J.; Hague, S. J. - The NAG Library 'Machine', Software - Practice and Experience, 9 (1979), pp. 65-72.

[13] Ford, Brian & Sayers, D. K. - Developing a Single Numerical Algorithms Library for Different Machine Ranges, ACM Trans. Math. Software, 2 (1976), pp. 115-131.

[14] Hague, Stephen J. & Ford, Brian - Portability - Prediction and Correction, Software - Practice and Experience, 6 (1976), pp. 61-69.

[15] Hattori, Mário T. - Uma Disciplina de Programação em FORTRAN-66. DSC/UFPb, Relatório Interno, Maio/1984.

[16] Krogh, Fred T. - Features for FORTRAN Portability, Portability of Numerical Software, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, pp. 361-367.

[17] Larmouth, J. - FORTRAN 77 Portability, Software - Practice & Experience, 11 (1981), pp. 1071-1117.

[18] Loveman, David B. - Program Improvement by Source to Source Transformation, 3rd ACM Symposium on Principles of Programming Languages, Atlanta, Georgia, Jan/1976, pp. 140-152.

[19] Maher, B. & Sleedman, D. H. - Automatic Program Improvement: Variable Usage Transformations, ACM Trans. Program. Lang. Syst., 5 (1983), pp. 236-264.

[20] Partsch, H. & Steinbrueggen, R. - Program Transformation Systems, Computing Surveys, 15 (1983), pp. 199-236.

[21] Poole, P. C. & Waite, W. M. - Portability and Adaptability - Software Engineering - An Advanced Course. F. L. Bauer (ed.), Springer_Verlag, pp. 183-277.

[22] Ryder, B. G. - The PFORT Verifier, Software - Practice and Experience, 4 (1974), pp. 359-377.

[23] Scowen, R. S. - Some Aids for Program Documentation, Software - Practice and Experience, 7 (1977), pp. 779-792.

[24] Standish, Thomas A.; Kibler, Dennis F.; Neighbors, James M. - Improving and Refining Programs by Program Manipulation, Proceedings of ACM Annual Conf - Houston, Texas, 1976, pp. 509-516.

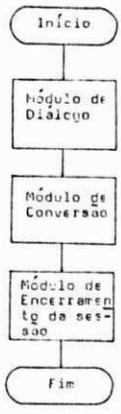
[25] Tanenbaum, A. S.; Klint, P.; Bohm, W. - Guidelines for Software Portability, Software - Practice and Experience, 8 (1978), pp. 681-698.

[26] Tassel, Dennie Van - Program Style, Design, Efficiency, Debugging and Testing. Prentice-Hall, Englewood Cliffs, N. J., 1974.

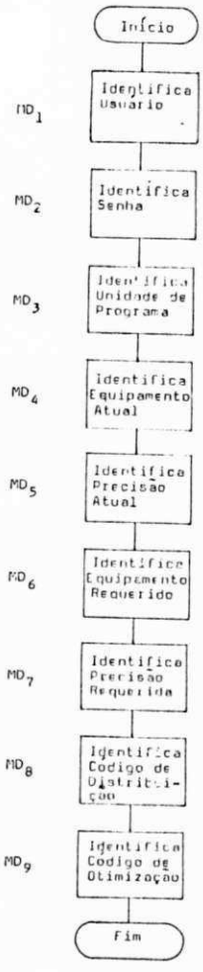
[27] Wyatt Jr.; W. T.; Lozier, D. W.; Orser, D. J. - A Portable Extended Precision Arithmetic Package and Library with FORTRAN Pre-compiler, ACM Trans. Math. Software, 2 (1976), pp. 209-231.

ANEXO I - MACRO-FLUXOGRAMA DO IRAPD.

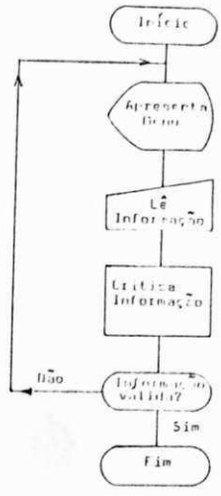
TRAPO: MACRO-FLUXOGRAMA



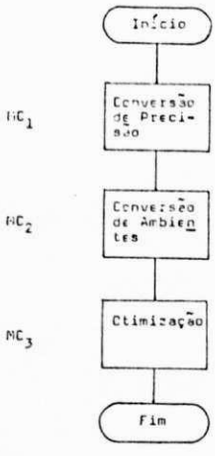
MÓDULO DE DIÁLOGO



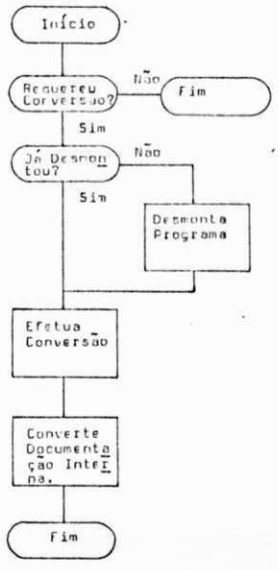
FORMA GERAL PARA CADA MD_i



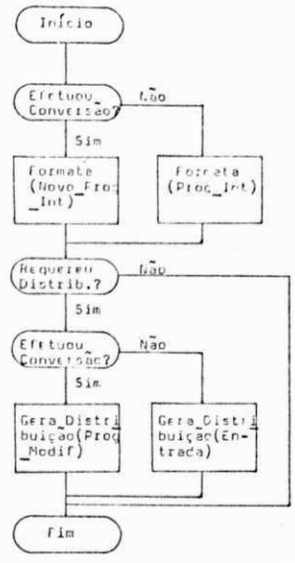
MÓDULO DE CONVERSÃO



FORMA GERAL PARA CADA MC_i



MÓDULO DE ENCERRAMENTO DA SESSÃO



ANEXO II - LISTAGENS DAS ROTINAS CONVERTIDAS.

```
      SUBROUTINE BALANC(NM,N,A,LOW,IGH,SCALE)
      DOUBLE PRECISION A(NM,N),SCALE(N)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C   ALGUNS COMENTARIOS FORAM ADICIONADOS NUMA TENTATIVA DE MOSTRAR
C   O FUNCIONAMENTO DA ROTINA DE CONVERSAO DA DOCUMENTACAO INTERNA:
C   OBSERVE O QUE ACONTECE COM AS INFORMACOES DAS PROXIMAS LINHAS
C   QUANDO ANTECEDIDAS POR AMPERSAND...
C&& ESTE COMENTARIO POSSUI A CONSTANTE 3. QUE NAO MUDARA, TEM
C&& A CONSTANTE &3. QUE MUDARA E A VARIAVEL &DMATRZ QUE MUDARA.
C&& A FUNCAO &DABS EM & PRECISAO DUPLA SERA ALTERADO JUNTAMENTE COM
C   O NOME DA PRECISAO, ENQUANTO QUE AS PALAVRAS PRECISAO DUPLA NAO
C&& MUDAM E AS PALAVRAS & PRECISAO DUPLA DEVEM MUDAR.
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C
C      ESTA SUBROTINA BALANCEA UMA MATRIZ REAL E ISOLA
C   AUTOVALORES SEMPRE QUE POSSIVEL.
C
C               UNIVERSIDADE FEDERAL DA PARAIBA
C               DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C               GRUPO DE ANALISE NUMERICA
C               MILCIADES MENTOR DE ARAUJO
C               VERSAO SET/85 (REVISAO POR HATTORI)
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C   OBSERVE QUE NA VERSAO ORIGINAL AS LINHAS QUE IDENTIFICAM A VERSAO
C   ESTAO EM BRANCO, ENQUANTO QUE NAS DEMAIS VERSOES ELAS ESTAO PREEN-
C   CHIDAS...
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C&  VERSAO:
C&  EQUIPAMENTO:
C&  PRECISAO:
C   PARAMETROS
C
C   NA CHAMADA
C      NM    INTEGER
C            EH UMA VARIAVEL INTEIRA DE ENTRADA QUE INDICA O NUMERO DE
C            LINHAS DA MATRIZ 'A'  COMO EH ESPECIFICADO NO COMANDO DI-
C            MENSION DO PROGRAMA PRINCIPAL.
C      N     INTEGER
C            EH A ORDEM DA MATRIZ.
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C  OBSERVE A PRECISAO DA PROXIMA LINHA: PERMANECE UM TIPO VALIDO NA
C  LINGUAGEM DE PROGRAMACAO...
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C&     A     &DOUBLE PRECISION (NM,N)
C            CONTEM A MATRIZ DE ENTRADA A SER BALANCEADA.
C
C   NO RETORNO
C      A        - CONTEM A MATRIZ BALANCEADA.
C      LOW,IGH  - SAO DOIS INTEIROS TAIS QUE A(I,J)
C                 EH IGUAL A ZERO SE -
C                    (1) I FOR MAIOR QUE J, E
C                    (2) J=1,...,LOW-1 OU I=IGH+1,...,N
C      SCALE    - CONTEM INFORMACAO DETERMINANDO AS PERMUTACOES E
C                 FATORES ESCALADOS USADOS.
C
C   SUPONHA QUE AS LINHAS DA SUBMATRIZ PRINCIPAL TENHAM SIDO BALANCEADAS
```

```
C    DE LOW PARA IGH, QUE P(J) DENOTA O INDICE TROCADO POR J DURANTE O
C    PASSO DE PERMUTACAO, E QUE OS ELEMENTOS DA MATRIZ DIAGONAL USADA
C    SAO DENOTADOS POR D(I,J). ENTAO
C        SCALE(J) = P(J)      PARA J=1,...,LOW-1
C                 = D(J,J)          J=LOW,...,IGH
C                 = P(J)            J=IGH+1,...,N.
C    A ORDEM NA QUAL AS TROCAS SAO EFETUADAS EH DE N PARA IGH+1, OU
C    ENTAO DE 1 PARA LOW-1.
C    NOTE QUE 1 RETORNARAH PARA IGH, SE IGH FOR FORMALMENTE ZERO.
C
C    ----------------------------------------------------------------
C
C               RADIX EH UM PARAMETRO DEPENDENTE DA MAQUINA QUE ESPECI-
C               FICA A BASE DA REPRESENTACAO DE PONTO FLUTUANTE DA MA-
C               QUINA.
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C    VERIFIQUE O QUE OCORRE COM AS DECLARACOES DE TIPO E COM A FUNCAO
C    INTERNA...
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      LOGICAL NOCONV
      DOUBLE PRECISION B2, C, F, G, R, S, DRADIX
      DOUBLE PRECISION DABS
      DOUBLE PRECISION  P95, UM, ZERO
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C  O PROXIMO COMANDO FOI PROPOSITALMENTE MODIFICADO. VEJA O QUE ACONTECE
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DATA         ZERO /0./,          P95 /.95D0/,              UM /1./
      DATA DRADIX/Z4210000000000000/
C
      B2 = DRADIX*DRADIX
      K=1
      L=N
      GO TO 100
C     ---------- PROCEDIMENTO IN-LINE PARA
C                TROCA DE LINHA E COLUNA ----------
   20 SCALE(M) = J
      IF (J .EQ. M) GO TO 50
C
         DO 30 I=1,L
            F = A(I,J)
            A(I,J) = A(I,M)
            A(I,M) = F
   30    CONTINUE
C
         DO 40 I=K,N
            F = A(J,I)
            A(J,I) = A(M,I)
            A(M,I) = F
   40    CONTINUE
C
   50 GO TO (80,130), IEXC
C     ---------- PESQUISA LINHAS PARA  ISOLAR UM
C                AUTOVALOR E COLOCA-LO PARA BAIXO ----------
   80 IF (L .EQ. 1) GO TO 280
      L=L-1
C     ---------- PARA J=L DECREMENTANDO J DE 1
C                ATE QUE ATINJA O VALOR 1 ----------
```

```
  100 DO 120 JJ=1,L
          J=L+1-JJ
C
          DO 110 I=1,L
             IF (I .EQ. J) GO TO 110
             IF (A(J,I) .NE. ZERO) GO TO 120
  110     CONTINUE
C
          M=L
          IEXC=1
          GO TO 20
  120 CONTINUE
C
      GO TO 140
C     ---------- PESQUISA COLUNAS PARA ISOLAR UM
C                AUTOVALOR E COLOCA-LO A ESQUERDA ----------
  130 K = K+1
C
  140 DO 170 J=K,L
C
          DO 150 I=K,L
             IF (I .EQ. J) GO TO 150
             IF (A(I,J) .NE. ZERO) GO TO 170
  150     CONTINUE
C
          M=K
          IEXC=2
          GO TO 20
  170 CONTINUE
C     ---------- BALANCEA A SUBMATRIZ DA LINHA K PARA L ----------
      DO 180 I=K,L
  180     SCALE(I) = UM
C     ---------- LOOP ITERATIVO PARA REDUCAO NORMAL ----------
  190 NOCONV = .FALSE.
C
      DO 270 I=K,L
         C = ZERO
         R = ZERO
         DO 200 J=K,L
            IF (J .EQ. I) GO TO 200
               C = C + DABS(A(J,I))
               R = R + DABS(A(I,J))
  200    CONTINUE
C
         G = R/DRADIX
         F = UM
         S = C+R
  210    IF (C .GE. G) GO TO 220
            F = F*DRADIX
            C = C * 82
            GO TO 210
  220    G = R * DRADIX
  230    IF (C .LT. G) GO TO 240
            F = F / DRADIX
            C = C / 82
            GO TO 230
C     ---------- BALANCEA ----------
  240    IF ((C + R) / F .GE. P95 + S) GO TO 270
```

```
               G = UM / F
               SCALE(I) = SCALE(I) * F
               NOCONV = .TRUE.
C
               DO 250 J = K,N
  250             A(I,J) = A(I,J) * G
C
               DO 260 J = 1,L
  260             A(J,I) = A(J,I) * F
C
  270 CONTINUE
C
      IF (NOCONV) GO TO 190
C
  280 LOW = K
      IGH = L
      RETURN
      END
```

```
      SUBROUTINE SALANC(NM,N,A,LOW,IGH,SCALE)
      REAL A(NM,N),SCALE(N)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C    ALGUNS COMENTARIOS FORAM ADICIONADOS NUMA TENTATIVA DE MOSTRAR
C    O FUNCIONAMENTO DA ROTINA DE CONVERSAO DA DOCUMENTACAO INTERNA:
C    OBSERVE O QUE ACONTECE COM AS INFORMACOES DAS PROXIMAS LINHAS
C    QUANDO ANTECEDIDAS POR AMPERSAND...
C&& ESTE COMENTARIO POSSUI A CONSTANTE 3. QUE NAO MUDARA, TEM
C&& A CONSTANTE &3.E O QUE MUDARA E A VARIAVEL &SMATRZ QUE MUDARA.
C&& A FUNCAO &ABS EM & PRECISAO SIMPLES SERA ALTERADO JUNTAMENTE COM
C    O NOME DA PRECISAO, ENQUANTO QUE AS PALAVRAS PRECISAO DUPLA NAO
C&& MUDAM E AS PALAVRAS & PRECISAO SIMPLES DEVEM MUDAR.
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C
C       ESTA SUBROTINA BALANCEA UMA MATRIZ REAL E ISOLA
C    AUTOVALORES SEMPRE QUE POSSIVEL.
C
C                  UNIVERSIDADE FEDERAL DA PARAIBA
C                  DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                  GRUPO DE ANALISE NUMERICA
C                  MILCIADES MENTOR DE ARAUJO
C                  VERSAO SET/85 (REVISAO POR HATTORI)
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C    OBSERVE QUE NA VERSAO ORIGINAL AS LINHAS QUE IDENTIFICAM A VERSAO
C    ESTAO EM BRANCO, ENQUANTO QUE NAS DEMAIS VERSOES ELAS ESTAO PREEN-
C    CHIDAS...
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C&   VERSAO:
C&   EQUIPAMENTO:IBM-4341
C&   PRECISAO:PRECISAO SIMPLES
C    PARAMETROS
C
C    NA CHAMADA
C       NM    INTEGER
C             EH UMA VARIAVEL INTEIRA DE ENTRADA QUE INDICA O NUMERO DE
C             LINHAS DA MATRIZ "A"  COMO EH ESPECIFICADO NO COMANDO DI-
C             MENSION DO PROGRAMA PRINCIPAL.
C       N     INTEGER
C             EH A ORDEM DA MATRIZ.
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C  OBSERVE A PRECISAO DA PROXIMA LINHA: PERMANECE UM TIPO VALIDO NA
C  LINGUAGEM DE PROGRAMACAO...
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C&      A     &REAL (NM,N)
C             CONTEM A MATRIZ DE ENTRADA A SER BALANCEADA.
C
C    NO RETORNO
C       A        - CONTEM A MATRIZ BALANCEADA.
C       LOW,IGH  - SAO DOIS INTEIROS TAIS QUE A(I,J)
C                  EH IGUAL A ZERO SE -
C                     (1) I FOR MAIOR QUE J, E
C                     (2) J=1,...,LOW-1 OU I=IGH+1,...,N
C       SCALE    - CONTEM INFORMACAO DETERMINANDO AS PERMUTACOES E
C                  FATORES ESCALADOS USADOS.
C
C    SUPONHA QUE AS LINHAS DA SUBMATRIZ PRINCIPAL TENHAM SIDO BALANCEADAS
```

```
C    DE LOW PARA IGH, QUE P(J) DENOTA O INDICE TROCADO POR J DURANTE O
C    PASSO DE PERMUTACAO, E QUE OS ELEMENTOS DA MATRIZ DIAGONAL USADA
C    SAO DENOTADOS POR D(I,J). ENTAO
C        SCALE(J) = P(J)      PARA J=1,...,LOW-1
C                 = D(J,J)          J=LOW,...,IGH
C                 = P(J)            J=IGH+1,...,N.
C    A ORDEM NA QUAL AS TROCAS SAO EFETUADAS EH DE N PARA IGH+1, OU
C    ENTAO DE 1 PARA LOW-1.
C    NOTE QUE 1 RETORNARAH PARA IGH, SE IGH FOR FORMALMENTE ZERO.
C
C    ----------------------------------------------------------------
C
C               RADIX EH UM PARAMETRO DEPENDENTE DA MAQUINA QUE ESPECI-
C               FICA A BASE DA REPRESENTACAO DE PONTO FLUTUANTE DA MA-
C               QUINA.
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C    VERIFIQUE O QUE OCORRE COM AS DECLARACOES DE TIPO E COM A FUNCAO
C    INTERNA...
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      LOGICAL NOCONV
      REAL B2, C, F, G, R, S, DRADIX
      REAL ABS
      REAL  P95, UM, ZERO
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C  O PROXIMO COMANDO FOI PROPOSITALMENTE MODIFICADO. VEJA O QUE ACONTECE
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DATA         ZERO /0.E 0/,          P95 /.95E0/,              UM /1.
     T E 0/
      DATA DRADIX/Z42100000000000 /
C
      B2 = DRADIX*DRADIX
      K=1
      L=N
      GO TO 100
C     ---------- PROCEDIMENTO IN-LINE PARA
C                TROCA DE LINHA E COLUNA ----------
   20 SCALE(M) = J
      IF (J .EQ. M) GO TO 50
C
         DO 30 I=1,L
            F = A(I,J)
            A(I,J) = A(I,M)
            A(I,M) = F
   30    CONTINUE
C
         DO 40 I=K,N
            F = A(J,I)
            A(J,I) = A(M,I)
            A(M,I) = F
   40    CONTINUE
C
   50 GO TO (80,130), IEXC
C     ---------- PESQUISA LINHAS PARA  ISOLAR UM
C                AUTOVALOR E COLOCA-LO PARA BAIXO ----------
   80 IF (L .EQ. 1) GO TO 280
      L=L-1
C     ---------- PARA J=L DECREMENTANDO J DE 1
```

```
C                  ATE QUE ATINJA O VALOR 1 ----------
  100 DO 120 JJ=1,L
         J=L+1-JJ
C
         DO 110 I=1,L
            IF (I .EQ. J) GO TO 110
            IF (A(J,I) .NE. ZERO) GO TO 120
  110    CONTINUE
C
         M=L
         IEXC=1
         GO TO 20
  120 CONTINUE
C
      GO TO 140
C     ---------- PESQUISA COLUNAS PARA ISOLAR UM
C                AUTOVALOR E COLOCA-LO A ESQUERDA ----------
  130 K = K+1
C
  140 DO 170 J=K,L
C
         DO 150 I=K,L
            IF (I .EQ. J) GO TO 150
            IF (A(I,J) .NE. ZERO) GO TO 170
  150    CONTINUE
C
         M=K
         IEXC=2
         GO TO 20
  170 CONTINUE
C     ---------- BALANCEA A SUBMATRIZ DA LINHA K PARA L ----------
      DO 180 I=K,L
  180    SCALE(I) = UM
C     ---------- LOOP ITERATIVO PARA REDUCAO NORMAL ----------
  190 NOCONV = .FALSE.
C
      DO 270 I=K,L
         C = ZERO
         R = ZERO
         DO 200 J=K,L
            IF (J .EQ. I) GO TO 200
               C = C + ABS(A(J,I))
               R = R + ABS(A(I,J))
  200    CONTINUE
C
         G = R/DRADIX
         F = UM
         S = C+R
  210    IF (C .GE. G) GO TO 220
            F = F*DRADIX
            C = C * B2
            GO TO 210
  220    G = R * DRADIX
  230    IF (C .LT. G) GO TO 240
            F = F / DRADIX
            C = C / B2
            GO TO 230
C     ---------- BALANCEA ----------
```

```
  240     IF ((C + R) / F .GE. P95 + S) GO TO 270
             G = UM / F
             SCALE(I) = SCALE(I) * F
             NOCONV = .TRUE.
C
             DO 250 J = K,N
  250           A(I,J) = A(I,J) * G
C
             DO 260 J = 1,L
  260           A(J,I) = A(J,I) * F
C
  270 CONTINUE
C
      IF (NOCONV) GO TO 190
C
  280 LOW = K
      IGH = L
      RETURN
      END
```

```
      SUBROUTINE DALANC(NM,N,A,LOW,IGH,SCALE)
      DOUBLE PRECISION A(NM,N),SCALE(N)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C    ALGUNS COMENTARIOS FORAM ADICIONADOS NUMA TENTATIVA DE MOSTRAR
C    O FUNCIONAMENTO DA ROTINA DE CONVERSAO DA DOCUMENTACAO INTERNA:
C    OBSERVE O QUE ACONTECE COM AS INFORMACOES DAS PROXIMAS LINHAS
C    QUANDO ANTECEDIDAS POR AMPERSAND...
C&& ESTE COMENTARIO POSSUI A CONSTANTE 3. QUE NAO MUDARA, TEM
C&& A CONSTANTE &3.D O QUE MUDARA E A VARIAVEL &DMATRZ QUE MUDARA.
C&& A FUNCAO &DABS EM E PRECISAO DUPLA SERA ALTERADO JUNTAMENTE COM
C    O NOME DA PRECISAO, ENQUANTO QUE AS PALAVRAS PRECISAO DUPLA NAO
C&& MUDAM E AS PALAVRAS E PRECISAO DUPLA DEVEM MUDAR.
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C
C       ESTA SUBROTINA BALANCEA UMA MATRIZ REAL E ISOLA
C    AUTOVALORES SEMPRE QUE POSSIVEL.
C
C                  UNIVERSIDADE FEDERAL DA PARAIBA
C                  DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                  GRUPO DE ANALISE NUMERICA
C                  MILCIADES MENTOR DE ARAUJO
C                  VERSAO SET/85 (REVISAO POR HATTORI)
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C    OBSERVE QUE NA VERSAO ORIGINAL AS LINHAS QUE IDENTIFICAM A VERSAO
C    ESTAO EM BRANCO, ENQUANTO QUE NAS DEMAIS VERSOES ELAS ESTAO PREEN-
C    CHIDAS...
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C&   VERSAO:
C&   EQUIPAMENTO:IBM-4341
C&   PRECISAO:PRECISAO DUPLA
C    PARAMETROS
C
C    NA CHAMADA
C       NM    INTEGER
C             EH UMA VARIAVEL INTEIRA DE ENTRADA QUE INDICA O NUMERO DE
C             LINHAS DA MATRIZ "A"  COMO EH ESPECIFICADO NO COMANDO DI-
C             MENSION DO PROGRAMA PRINCIPAL.
C       N     INTEGER
C             EH A ORDEM DA MATRIZ.
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C  OBSERVE A PRECISAO DA PROXIMA LINHA: PERMANECE UM TIPO VALIDO NA
C  LINGUAGEM DE PROGRAMACAO...
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C&      A     &DOUBLE PRECISION (NM,N)
C .           CONTEM A MATRIZ DE ENTRADA A SER BALANCEADA.
C
C    NO RETORNO
C       A        - CONTEM A MATRIZ BALANCEADA.
C       LOW,IGH  - SAO DOIS INTEIROS TAIS QUE A(I,J)
C                  EH IGUAL A ZERO SE -
C                     (1) I FOR MAIOR QUE J, E
C                     (2) J=1,...,LOW-1 OU I=IGH+1,...,N
C       SCALE    - CONTEM INFORMACAO DETERMINANDO AS PERMUTACOES E
C                  FATORES ESCALADOS USADOS.
C
C    SUPONHA QUE AS LINHAS DA SUBMATRIZ PRINCIPAL TENHAM SIDO BALANCEADAS
```

```
C    DE LOW PARA IGH, QUE P(J) DENOTA O INDICE TROCADO POR J DURANTE O
C    PASSO DE PERMUTACAO, E QUE OS ELEMENTOS DA MATRIZ DIAGONAL USADA
C    SAO DENOTADOS POR D(I,J). ENTAO
C        SCALE(J) = P(J)      PARA J=1,...,LOW-1
C                 = D(J,J)          J=LOW,...,IGH
C                 = P(J)            J=IGH+1,...,N.
C    A ORDEM NA QUAL AS TROCAS SAO EFETUADAS EH DE N PARA IGH+1, OU
C    ENTAO DE 1 PARA LOW-1.
C    NOTE QUE 1 RETORNARAH PARA IGH, SE IGH FOR FORMALMENTE ZERO.
C
C    ----------------------------------------------------------------
C
C                RADIX EH UM PARAMETRO DEPENDENTE DA MAQUINA QUE ESPECI-
C                FICA A BASE DA REPRESENTACAO DE PONTO FLUTUANTE DA MA-
C                QUINA.
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C    VERIFIQUE O QUE OCORRE COM AS DECLARACOES DE TIPO E COM A FUNCAO
C    INTERNA...
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      LOGICAL NOCONV
      DOUBLE PRECISION B2, C, F, G, R, S, DRADIX
      DOUBLE PRECISION DABS
      DOUBLE PRECISION  P95, UM, ZERO
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C  O PROXIMO COMANDO FOI PROPOSITALMENTE MODIFICADO. VEJA O QUE ACONTECE
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DATA         ZERO /0.D 0/,          P95 /.95D0/,              UM /1.
     T D 0/
      DATA DRADIX/Z42100000000000 /
C
      B2 = DRADIX*DRADIX
      K=1
      L=N
      GO TO 100
C     ---------- PROCEDIMENTO IN-LINE PARA
C                TROCA DE LINHA E COLUNA ----------
   20 SCALE(M) = J
      IF (J .EQ. M) GO TO 50
C
         DO 30 I=1,L
            F = A(I,J)
            A(I,J) = A(I,M)
            A(I,M) = F
   30    CONTINUE
C
         DO 40 I=K,N
            F = A(J,I)
            A(J,I) = A(M,I)
            A(M,I) = F
   40    CONTINUE
C
   50 GO TO (80,130), IEXC
C     ---------- PESQUISA LINHAS PARA  ISOLAR UM
C                AUTOVALOR E COLOCA-LO PARA BAIXO ----------
   80 IF (L .EQ. 1) GO TO 280
      L=L-1
C     ---------- PARA J=L DECREMENTANDO J DE 1
```

```
C                  ATE QUE ATINJA O VALOR 1 ----------
  100 DO 120 JJ=1,L
         J=L+1-JJ
C
         DO 110 I=1,L
            IF (I .EQ. J) GO TO 110
            IF (A(J,I) .NE. ZERO) GO TO 120
  110    CONTINUE
C
         M=L
         IEXC=1
         GO TO 20
  120 CONTINUE
C
      GO TO 140
C     ---------- PESQUISA COLUNAS PARA ISOLAR UM
C                AUTOVALOR E COLOCA-LO A ESQUERDA ----------
  130 K = K+1
C
  140 DO 170 J=K,L
C
         DO 150 I=K,L
            IF (I .EQ. J) GO TO 150
            IF (A(I,J) .NE. ZERO) GO TO 170
  150    CONTINUE
C
         M=K
         IEXC=2
         GO TO 20
  170 CONTINUE
C     ---------- BALANCEA A SUBMATRIZ DA LINHA K PARA L ----------
      DO 180 I=K,L
  180    SCALE(I) = UM
C     ---------- LOOP ITERATIVO PARA REDUCAO NORMAL ----------
  190 NOCONV = .FALSE.
C
      DO 270 I=K,L
         C = ZERO
         R = ZERO
         DO 200 J=K,L
            IF (J .EQ. I) GO TO 200
               C = C + DABS(A(J,I))
               R = R + DABS(A(I,J))
  200    CONTINUE
C
         G = R/DRADIX
         F = UM
         S = C+R
  210    IF (C .GE. G) GO TO 220
            F = F*DRADIX
            C = C * B2
            GO TO 210
  220    G = R * DRADIX
  230    IF (C .LT. G) GO TO 240
            F = F / DRADIX
            C = C / B2
            GO TO 230
C     ---------- BALANCEA ----------
```

```
  240     IF ((C + R) / F .GE. P95 + S) GO TO 270
             G = UM / F
             SCALE(I) = SCALE(I) * F
             NOCONV = .TRUE.
C
             DO 250 J = K,N
  250           A(I,J) = A(I,J) * G
C
             DO 260 J = 1,L
  260           A(J,I) = A(J,I) * F
C
  270 CONTINUE
C
      IF (NOCONV) GO TO 190
C
  280 LOW = K
      IGH = L
      RETURN
      END
```

```
      SUBROUTINE SALANC(NM,N,A,LOW,IGH,SCALE)
      REAL A(NM,N),SCALE(N)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C   ALGUNS COMENTARIOS FORAM ADICIONADOS NUMA TENTATIVA DE MOSTRAR
C   O FUNCIONAMENTO DA ROTINA DE CONVERSAO DA DOCUMENTACAO INTERNA:
C   OBSERVE O QUE ACONTECE COM AS INFORMACOES DAS PROXIMAS LINHAS
C   QUANDO ANTECEDIDAS POR AMPERSAND...
C&& ESTE COMENTARIO POSSUI A CONSTANTE 3. QUE NAO MUDARA, TEM
C&& A CONSTANTE &3.E O QUE MUDARA E A VARIAVEL &SMATRZ QUE MUDARA.
C&& A FUNCAO &ABS EM & PRECISAO SIMPLES SERA ALTERADO JUNTAMENTE COM
C   O NOME DA PRECISAO, ENQUANTO QUE AS PALAVRAS PRECISAO DUPLA NAO
C&& MUDAM E AS PALAVRAS & PRECISAO SIMPLES DEVEM MUDAR.
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C
C      ESTA SUBROTINA BALANCEA UMA MATRIZ REAL E ISOLA
C   AUTOVALORES SEMPRE QUE POSSIVEL.
C
C                UNIVERSIDADE FEDERAL DA PARAIBA
C                DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                GRUPO DE ANALISE NUMERICA
C                MILCIADES MENTOR DE ARAUJO
C                VERSAO SET/85 (REVISAO POR HATTORI)
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C   OBSERVE QUE NA VERSAO ORIGINAL AS LINHAS QUE IDENTIFICAM A VERSAO
C   ESTAO EM BRANCO, ENQUANTO QUE NAS DEMAIS VERSOES ELAS ESTAO PREEN-
C   CHIDAS...
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C&  VERSAO:
C&  EQUIPAMENTO:IBM-4341
C&  PRECISAO:PRECISAO SIMPLES
C   PARAMETROS
C
C   NA CHAMADA
C      NM    INTEGER
C            EH UMA VARIAVEL INTEIRA DE ENTRADA QUE INDICA O NUMERO DE
C            LINHAS DA MATRIZ "A"  COMO EH ESPECIFICADO NO COMANDO DI-
C            MENSION DO PROGRAMA PRINCIPAL.
C      N     INTEGER
C            EH A ORDEM DA MATRIZ.
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C  OBSERVE A PRECISAO DA PROXIMA LINHA: PERMANECE UM TIPO VALIDO NA
C  LINGUAGEM DE PROGRAMACAO...
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C&     A     &REAL (NM,N)
C            CONTEM A MATRIZ DE ENTRADA A SER BALANCEADA.
C
C   NO RETORNO
C      A        - CONTEM A MATRIZ BALANCEADA.
C      LOW,IGH  - SAO DOIS INTEIROS TAIS QUE A(I,J)
C                 EH IGUAL A ZERO SE -
C                    (1) I FOR MAIOR QUE J, E
C                    (2) J=1,...,LOW-1 OU I=IGH+1,...,N
C      SCALE    - CONTEM INFORMACAO DETERMINANDO AS PERMUTACOES E
C                 FATORES ESCALADOS USADOS.
C
C   SUPONHA QUE AS LINHAS DA SUBMATRIZ PRINCIPAL TENHAM SIDO BALANCEADAS
```

```
C    DE LOW PARA IGH, QUE P(J) DENOTA O INDICE TROCADO POR J DURANTE O
C    PASSO DE PERMUTACAO, E QUE OS ELEMENTOS DA MATRIZ DIAGONAL USADA
C    SAO DENOTADOS POR D(I,J). ENTAO
C        SCALE(J) = P(J)      PARA J=1,...,LOW-1
C                 = D(J,J)          J=LOW,...,IGH
C                 = P(J)            J=IGH+1,...,N.
C    A ORDEM NA QUAL AS TROCAS SAO EFETUADAS EH DE N PARA IGH+1, OU
C    ENTAO DE 1 PARA LOW-1.
C    NOTE QUE 1 RETORNARAH PARA IGH, SE IGH FOR FORMALMENTE ZERO.
C
C    ----------------------------------------------------------------
C
C               RADIX EH UM PARAMETRO DEPENDENTE DA MAQUINA QUE ESPECI-
C               FICA A BASE DA REPRESENTACAO DE PONTO FLUTUANTE DA MA-
C               QUINA.
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C    VERIFIQUE O QUE OCORRE COM AS DECLARACOES DE TIPO E COM A FUNCAO
C    INTERNA...
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      LOGICAL NOCONV
      REAL B2, C, F, G, R, S, DRADIX
      REAL ABS
      REAL  P95, UM, ZERO
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C  O PROXIMO COMANDO FOI PROPOSITALMENTE MODIFICADO. VEJA O QUE ACONTECE
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DATA         ZERO /0.E 0/,          P95 /.95E0/,              UM /1.
     T E 0/
      DATA DRADIX/Z42100000000000 /
C
      B2 = DRADIX*DRADIX
      K=1
      L=N
      GO TO 100
C     ---------- PROCEDIMENTO IN-LINE PARA
C                TROCA DE LINHA E COLUNA ----------
   20 SCALE(M) = J
      IF (J .EQ. M) GO TO 50
C
         DO 30 I=1,L
            F = A(I,J)
            A(I,J) = A(I,M)
            A(I,M) = F
   30    CONTINUE
C
         DO 40 I=K,N
            F = A(J,I)
            A(J,I) = A(M,I)
            A(M,I) = F
   40    CONTINUE
C
   50 GO TO (80,130), IEXC
C     ---------- PESQUISA LINHAS PARA  ISOLAR UM
C                AUTOVALOR E COLOCA-LO PARA BAIXO ----------
   80 IF (L .EQ. 1) GO TO 280
      L=L-1
C     ---------- PARA J=L DECREMENTANDO J DE 1
```

```
C                  ATE QUE ATINJA O VALOR 1 ----------
  100 DO 120 JJ=1,L
          J=L+1-JJ
C
          DO 110 I=1,L
             IF (I .EQ. J) GO TO 110
             IF (A(J,I) .NE. ZERO) GO TO 120
  110     CONTINUE
C
          M=L
          IEXC=1
          GO TO 20
  120 CONTINUE
C
      GO TO 140
C     ---------- PESQUISA COLUNAS PARA ISOLAR UM
C                AUTOVALOR E COLOCA-LO A ESQUERDA ----------
  130 K = K+1
C
  140 DO 170 J=K,L
C
          DO 150 I=K,L
             IF (I .EQ. J) GO TO 150
             IF (A(I,J) .NE. ZERO) GO TO 170
  150     CONTINUE
C
          M=K
          IEXC=2
          GO TO 20
  170 CONTINUE
C     ---------- BALANCEA A SUBMATRIZ DA LINHA K PARA L ----------
      DO 180 I=K,L
  180     SCALE(I) = UM
C     ---------- LOOP ITERATIVO PARA REDUCAO NORMAL ----------
  190 NOCONV = .FALSE.
C
      DO 270 I=K,L
         C = ZERO
         R = ZERO
         DO 200 J=K,L
            IF (J .EQ. I) GO TO 200
               C = C + ABS(A(J,I))
               R = R + ABS(A(I,J))
  200    CONTINUE
C
         G = R/DRADIX
         F = UM
         S = C+R
  210    IF (C .GE. G) GO TO 220
            F = F*DRADIX
            C = C * B2
            GO TO 210
  220    G = R * DRADIX
  230    IF (C .LT. G) GO TO 240
            F = F / DRADIX
            C = C / B2
            GO TO 230
C     ---------- BALANCEA ----------
```

```
  240    IF ((C + R) / F .GE. P95 + S) GO TO 270
            G = UM / F
            SCALE(I) = SCALE(I) * F
            NOCONV = .TRUE.
C
            DO 250 J = K,N
  250          A(I,J) = A(I,J) * G
C
            DO 260 J = 1,L
  260          A(J,I) = A(J,I) * F
C
  270 CONTINUE
C
      IF (NOCONV) GO TO 190
C
  280 LOW = K
      IGH = L
      RETURN
      END
```

```
      SUBROUTINE DALANC(NM,N,A,LOW,IGH,SCALE)
      DOUBLE PRECISION A(NM,N),SCALE(N)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C    ALGUNS COMENTARIOS FORAM ADICIONADOS NUMA TENTATIVA DE MOSTRAR
C    O FUNCIONAMENTO DA ROTINA DE CONVERSAO DA DOCUMENTACAO INTERNA:
C    OBSERVE O QUE ACONTECE COM AS INFORMACOES DAS PROXIMAS LINHAS
C    QUANDO ANTECEDIDAS POR AMPERSAND...
C&& ESTE COMENTARIO POSSUI A CONSTANTE 3. QUE NAO MUDARA, TEM
C&& A CONSTANTE &3.D O QUE MUDARA E A VARIAVEL &DMATRZ QUE MUDARA.
C&& A FUNCAO &DABS EM & PRECISAO DUPLA SERA ALTERADO JUNTAMENTE COM
C    O NOME DA PRECISAO, ENQUANTO QUE AS PALAVRAS PRECISAO DUPLA NAO
C&& MUDAM E AS PALAVRAS & PRECISAO DUPLA DEVEM MUDAR.
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C
C       ESTA SUBROTINA BALANCEA UMA MATRIZ REAL E ISOLA
C    AUTOVALORES SEMPRE QUE POSSIVEL.
C
C                  UNIVERSIDADE FEDERAL DA PARAIBA
C                  DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                  GRUPO DE ANALISE NUMERICA
C                  MILCIADES MENTOR DE ARAUJO
C                  VERSAO SET/85 (REVISAO POR HATTORI)
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C    OBSERVE QUE NA VERSAO ORIGINAL AS LINHAS QUE IDENTIFICAM A VERSAO
C    ESTAO EM BRANCO, ENQUANTO QUE NAS DEMAIS VERSOES ELAS ESTAO PREEN-
C    CHIDAS...
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C&   VERSAO:
C&   EQUIPAMENTO:IBM-4341
C&   PRECISAO:PRECISAO DUPLA
C    PARAMETROS
C
C    NA CHAMADA
C       NM    INTEGER
C             EH UMA VARIAVEL INTEIRA DE ENTRADA QUE INDICA O NUMERO DE
C             LINHAS DA MATRIZ 'A'  COMO EH ESPECIFICADO NO COMANDO DI-
C             MENSION DO PROGRAMA PRINCIPAL.
C       N     INTEGER
C             EH A ORDEM DA MATRIZ.
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C  OBSERVE A PRECISAO DA PROXIMA LINHA: PERMANECE UM TIPO VALIDO NA
C  LINGUAGEM DE PROGRAMACAO...
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C&      A     &DOUBLE PRECISION (NM,N)
C             CONTEM A MATRIZ DE ENTRADA A SER BALANCEADA.
C
C    NO RETORNO
C       A         - CONTEM A MATRIZ BALANCEADA.
C       LOW,IGH   - SAO DOIS INTEIROS TAIS QUE A(I,J)
C                   EH IGUAL A ZERO SE -
C                      (1) I FOR MAIOR QUE J, E
C                      (2) J=1,...,LOW-1 OU I=IGH+1,...,N
C       SCALE     - CONTEM INFORMACAO DETERMINANDO AS PERMUTACOES E
C                   FATORES ESCALADOS USADOS.
C
C    SUPONHA QUE AS LINHAS DA SUBMATRIZ PRINCIPAL TENHAM SIDO BALANCEADAS
```

```
C    DE LOW PARA IGH, QUE P(J) DENOTA O INDICE TROCADO POR J DURANTE O
C    PASSO DE PERMUTACAO, E QUE OS ELEMENTOS DA MATRIZ DIAGONAL USADA
C    SAO DENOTADOS POR D(I,J). ENTAO
C        SCALE(J) = P(J)      PARA J=1,...,LOW-1
C                 = D(J,J)          J=LOW,...,IGH
C                 = P(J)            J=IGH+1,...,N.
C    A ORDEM NA QUAL AS TROCAS SAO EFETUADAS EH DE N PARA IGH+1, OU
C    ENTAO DE 1 PARA LOW-1.
C    NOTE QUE 1 RETORNARAH PARA IGH, SE IGH FOR FORMALMENTE ZERO.
C
C    ----------------------------------------------------------------
C
C               RADIX EH UM PARAMETRO DEPENDENTE DA MAQUINA QUE ESPECI-
C               FICA A BASE DA REPRESENTACAO DE PONTO FLUTUANTE DA MA-
C               QUINA.
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C    VERIFIQUE O QUE OCORRE COM AS DECLARACOES DE TIPO E COM A FUNCAO
C    INTERNA...
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      LOGICAL NOCONV
      DOUBLE PRECISION B2, C, F, G, R, S, DRADIX
      DOUBLE PRECISION DABS
      DOUBLE PRECISION  P95, UM, ZERO
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C  O PROXIMO COMANDO FOI PROPOSITALMENTE MODIFICADO. VEJA O QUE ACONTECE
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DATA         ZERO /0.D 0/,          P95 /.95D0/,              UM /1.
     T D 0/
      DATA DRADIX/Z42100000000000 /
C
      B2 = DRADIX*DRADIX
      K=1
      L=N
      GO TO 100
C     ---------- PROCEDIMENTO IN-LINE PARA
C                TROCA DE LINHA E COLUNA ----------
   20 SCALE(M) = J
      IF (J .EQ. M) GO TO 50
C
         DO 30 I=1,L
            F = A(I,J)
            A(I,J) = A(I,M)
            A(I,M) = F
   30    CONTINUE
C
         DO 40 I=K,N
            F = A(J,I)
            A(J,I) = A(M,I)
            A(M,I) = F
   40    CONTINUE
C
   50 GO TO (80,130), IEXC
C     ---------- PESQUISA LINHAS PARA  ISOLAR UM
C                AUTOVALOR E COLOCA-LO PARA BAIXO ----------
   80 IF (L .EQ. 1) GO TO 280
      L=L-1
C     ---------- PARA J=L DECREMENTANDO J DE 1
```

```
C                  ATE QUE ATINJA O VALOR 1 ----------
  100 DO 120 JJ=1,L
         J=L+1-JJ
C
         DO 110 I=1,L
            IF (I .EQ. J) GO TO 110
            IF (A(J,I) .NE. ZERO) GO TO 120
  110    CONTINUE
C
         M=L
         IEXC=1
         GO TO 20
  120 CONTINUE
C
      GO TO 140
C     ---------- PESQUISA COLUNAS PARA ISOLAR UM
C                AUTOVALOR E COLOCA-LO A ESQUERDA ----------
  130 K = K+1
C
  140 DO 170 J=K,L
C
         DO 150 I=K,L
            IF (I .EQ. J) GO TO 150
            IF (A(I,J) .NE. ZERO) GO TO 170
  150    CONTINUE
C
         M=K
         IEXC=2
         GO TO 20
  170 CONTINUE
C     ---------- BALANCEA A SUBMATRIZ DA LINHA K PARA L ----------
      DO 180 I=K,L
  180    SCALE(I) = UM
C     ---------- LOOP ITERATIVO PARA REDUCAO NORMAL ----------
  190 NOCONV = .FALSE.
C
      DO 270 I=K,L
         C = ZERO
         R = ZERO
         DO 200 J=K,L
            IF (J .EQ. I) GO TO 200
               C = C + DABS(A(J,I))
               R = R + DABS(A(I,J))
  200    CONTINUE
C
         G = R/DRADIX
         F = UM
         S = C+R
  210    IF (C .GE. G) GO TO 220
            F = F*DRADIX
            C = C * B2
            GO TO 210
  220    G = R * DRADIX
  230    IF (C .LT. G) GO TO 240
            F = F / DRADIX
            C = C / B2
            GO TO 230
C     ---------- BALANCEA ----------
```

```
  240     IF ((C + R) / F .GE. P95 + S) GO TO 270
             G = UM / F
             SCALE(I) = SCALE(I) * F
             NOCONV = .TRUE.
C
             DO 250 J = K,N
  250           A(I,J) = A(I,J) * G
C
             DO 260 J = 1,L
  260           A(J,I) = A(J,I) * F
C
  270 CONTINUE
C
      IF (NOCONV) GO TO 190
C
  280 LOW = K
      IGH = L
      RETURN
      END
```

```
      SUBROUTINE BALANC(NM,N,A,LOW,IGH,SCALE)
      DOUBLE PRECISION A(NM,N),SCALE(N)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C   ALGUNS COMENTARIOS FORAM ADICIONADOS NUMA TENTATIVA DE MOSTRAR
C   O FUNCIONAMENTO DA ROTINA DE CONVERSAO DA DOCUMENTACAO INTERNA:
C   OBSERVE O QUE ACONTECE COM AS INFORMACOES DAS PROXIMAS LINHAS
C   QUANDO ANTECEDIDAS POR AMPERSAND...
C   ESTE COMENTARIO POSSUI A CONSTANTE 3. QUE NAO MUDARA, TEM
C   A CONSTANTE  3.D 0 QUE MUDARA E A VARIAVEL  DMATRZ QUE MUDARA.
C   A FUNCAO  DABS EM   PRECISAO DUPLA SERA ALTERADO JUNTAMENTE COM
C   O NOME DA PRECISAO, ENQUANTO QUE AS PALAVRAS PRECISAO DUPLA NAO
C   MUDAM E AS PALAVRAS   PRECISAO DUPLA DEVEM MUDAR.
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C
C      ESTA SUBROTINA BALANCEA UMA MATRIZ REAL E ISOLA
C   AUTOVALORES SEMPRE QUE POSSIVEL.
C
C               UNIVERSIDADE FEDERAL DA PARAIBA
C               DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C               GRUPO DE ANALISE NUMERICA
C               MILCIADES MENTOR DE ARAUJO
C               VERSAO SET/85 (REVISAO POR HATTORI)
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C   OBSERVE QUE NA VERSAO ORIGINAL AS LINHAS QUE IDENTIFICAM A VERSAO
C   ESTAO EM BRANCO, ENQUANTO QUE NAS DEMAIS VERSOES ELAS ESTAO PREEN-
C   CHIDAS...
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C   VERSAO:
C   EQUIPAMENTO:IBM-4341
C   PRECISAO:PRECISAO DUPLA
C   PARAMETROS
C
C   NA CHAMADA
C      NM    INTEGER
C            EH UMA VARIAVEL INTEIRA DE ENTRADA QUE INDICA O NUMERO DE
C            LINHAS DA MATRIZ "A"  COMO EH ESPECIFICADO NO COMANDO DI-
C            MENSION DO PROGRAMA PRINCIPAL.
C      N     INTEGER
C            EH A ORDEM DA MATRIZ.
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C  OBSERVE A PRECISAO DA PROXIMA LINHA: PERMANECE UM TIPO VALIDO NA
C  LINGUAGEM DE PROGRAMACAO...
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C      A     DOUBLE PRECISION (NM,N)
C            CONTEM A MATRIZ DE ENTRADA A SER BALANCEADA.
C
C   NO RETORNO
C      A       - CONTEM A MATRIZ BALANCEADA.
C      LOW,IGH - SAO DOIS INTEIROS TAIS QUE A(I,J)
C                EH IGUAL A ZERO SE -
C                   (1) I FOR MAIOR QUE J, E
C                   (2) J=1,...,LOW-1 OU I=IGH+1,...,N
C      SCALE   - CONTEM INFORMACAO DETERMINANDO AS PERMUTACOES E
C                FATORES ESCALADOS USADOS.
C
C   SUPONHA QUE AS LINHAS DA SUBMATRIZ PRINCIPAL TENHAM SIDO BALANCEADAS
```

```
C    DE LOW PARA IGH, QUE P(J) DENOTA O INDICE TROCADO POR J DURANTE O
C    PASSO DE PERMUTACAO, E QUE OS ELEMENTOS DA MATRIZ DIAGONAL USADA
C    SAO DENOTADOS POR D(I,J). ENTAO
C        SCALE(J) = P(J)      PARA J=1,...,LOW-1
C                 = D(J,J)          J=LOW,...,IGH
C                 = P(J)            J=IGH+1,...,N.
C    A ORDEM NA QUAL AS TROCAS SAO EFETUADAS EH DE N PARA IGH+1, OU
C    ENTAO DE 1 PARA LOW-1.
C    NOTE QUE 1 RETORNARAH PARA IGH, SE IGH FOR FORMALMENTE ZERO.
C
C    ----------------------------------------------------------------
C
C             RADIX EH UM PARAMETRO DEPENDENTE DA MAQUINA QUE ESPECI-
C             FICA A BASE DA REPRESENTACAO DE PONTO FLUTUANTE DA MA-
C             QUINA.
C
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C    VERIFIQUE O QUE OCORRE COM AS DECLARACOES DE TIPO E COM A FUNCAO
C    INTERNA...
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      LOGICAL NOCONV
      DOUBLE PRECISION B2, C, F, G, R, S, DRADIX
      DOUBLE PRECISION DABS
      DOUBLE PRECISION  P95, UM, ZERO
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C  O PROXIMO COMANDO FOI PROPOSITALMENTE MODIFICADO. VEJA O QUE ACONTECE
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DATA         ZERO /0.D 0/,          P95 /.95D0/,              UM /1.
     T D 0/
      DATA DRADIX/Z42100000000000 /
C
      B2 = DRADIX*DRADIX
      K=1
      L=N
      GO TO 100
C     ---------- PROCEDIMENTO IN-LINE PARA
C                TROCA DE LINHA E COLUNA ----------
   20 SCALE(M) = J
      IF (J .EQ. M) GO TO 50
C
         DO 30 I=1,L
            F = A(I,J)
            A(I,J) = A(I,M)
            A(I,M) = F
   30    CONTINUE
C
         DO 40 I=K,N
            F = A(J,I)
            A(J,I) = A(M,I)
            A(M,I) = F
   40    CONTINUE
C
   50 GO TO (80,130), IEXC
C     ---------- PESQUISA LINHAS PARA  ISOLAR UM
C                AUTOVALOR E COLOCA-LO PARA BAIXO ----------
   80 IF (L .EQ. 1) GO TO 280
      L=L-1
C     ---------- PARA J=L DECREMENTANDO J DE 1
```

```
C                  ATE QUE ATINJA O VALOR 1 ----------
  100 DO 120 JJ=1,L
         J=L+1-JJ
C
         DO 110 I=1,L
            IF (I .EQ. J) GO TO 110
            IF (A(J,I) .NE. ZERO) GO TO 120
  110    CONTINUE
C
         M=L
         IEXC=1
         GO TO 20
  120 CONTINUE
C
      GO TO 140
C     ---------- PESQUISA COLUNAS PARA ISOLAR UM
C                AUTOVALOR E COLOCA-LO A ESQUERDA ----------
  130 K = K+1
C
  140 DO 170 J=K,L
C
         DO 150 I=K,L
            IF (I .EQ. J) GO TO 150
            IF (A(I,J) .NE. ZERO) GO TO 170
  150    CONTINUE
C
         M=K
         IEXC=2
         GO TO 20
  170 CONTINUE
C     ---------- BALANCEA A SUBMATRIZ DA LINHA K PARA L ----------
      DO 180 I=K,L
  180    SCALE(I) = UM
C     ---------- LOOP ITERATIVO PARA REDUCAO NORMAL ----------
  190 NOCONV = .FALSE.
C
      DO 270 I=K,L
         C = ZERO
         R = ZERO
         DO 200 J=K,L
            IF (J .EQ. I) GO TO 200
               C = C + DABS(A(J,I))
               R = R + DABS(A(I,J))
  200    CONTINUE
C
         G = R/DRADIX
         F = UM
         S = C+R
  210    IF (C .GE. G) GO TO 220
            F = F*DRADIX
            C = C * B2
            GO TO 210
  220    G = R * DRADIX
  230    IF (C .LT. G) GO TO 240
            F = F / DRADIX
            C = C / B2
            GO TO 230
C     ---------- BALANCEA ----------
```

```
  240     IF ((C + R) / F .GE. P95 * S) GO TO 270
             G = UM / F
             SCALE(I) = SCALE(I) * F
             NOCONV = .TRUE.
C
             DO 250 J = K,N
  250           A(I,J) = A(I,J) * G
C
             DO 260 J = 1,L
  260           A(J,I) = A(J,I) * F
C
  270 CONTINUE
C
      IF (NOCONV) GO TO 190
C
  280 LOW = K
      IGH = L
      RETURN
      END
```

```
R; T=0.01/0.01 09:40:31
trapo
Campina Grande, 15/01/86.

Sr. Usuario, bom dia.

       Sao 09:40 e estamos iniciando uma
nova Sessao de Trabalho do  T R A P O.


Informe o seu CODIGO DE USUARIO:
sarney
ERRO  1:
Usuario nao Autorizado a Requerer Transformacoes


Informe o seu CODIGO DE USUARIO:
hattori

Informe a sua SENHA:
Funaro
ERRO  2:
Senha Invalida


Deseja Continuar (S = Sim, N = Nao)?
quero
ERRO 10:
Codigo de Continuacao Invalido


Deseja Continuar (S = Sim, N = Nao)?
s

Informe o seu CODIGO DE USUARIO:
camilo

Informe a sua SENHA:
camilo

Informe a UNIDADE DE PROGRAMA:
Todas
ERRO  3:
Unidade de Programa Inexistente


Informe a UNIDADE DE PROGRAMA:
BALANC

Informe o CODIGO DO EQUIPAMENTO ATUAL:
01 - IBM-4341
02 - COBRA-530
03 - DEC-10
 ? - Mostre a Situacao Corrente
 ! - Anule esta Sessao de Trabalho
CODIGO ESCOLHIDO:
01
```

```
Informe o CODIGO DA PRECISAO ATUAL:
1 - Precisao Simples
2 - Precisao Dupla
3 - Complexo de Precisao Simples
4 - Complexo de Precisao Dupla
? - Mostre a Situacao Corrente
! - Anule esta Sessao de Trabalho
CODIGO ESCOLHIDO:
1

Informe o CODIGO DO EQUIPAMENTO REQUERIDO:
01 - IBM-4341
02 - COBRA-530
03 - DEC-10
 ? - Mostre a Situacao Corrente
 ! - Anule esta Sessao de Trabalho
CODIGO ESCOLHIDO:
01

Informe o CODIGO DA PRECISAO REQUERIDA:
1 - Precisao Simples
2 - Precisao Dupla
3 - Complexo de Precisao Simples
4 - Complexo de Precisao Dupla
? - Mostre a Situacao Corrente
! - Anule esta Sessao de Trabalho
CODIGO ESCOLHIDO:
1

Informe se Deseja DECK DE DISTRIBUICAO (S = Sim, N = Nao):
talvez
ERRO  7:
Codigo de Distribuicao Invalido


Informe se Deseja DECK DE DISTRIBUICAO (S = Sim, N = Nao):
tambem
ERRO  7:
Codigo de Distribuicao Invalido


Informe se Deseja DECK DE DISTRIBUICAO (S = Sim, N = Nao):
n

Informe se Deseja OTIMIZACAO (S = Sim, N = Nao):
quero
ERRO  8:
Codigo de Otimizacao Invalido


Informe se Deseja OTIMIZACAO (S = Sim, N = Nao):
n
```

```
-------------------------------------------------
!  SISTEMA TRANSFORMADOR DE PROGRAMAS FORTRAN.  !
!  Usuario:  camilo                             !
!  Unidade de Programa:BALANC                   !
!  VERSAO ATUAL.                                !
!     Equipamento:                              !
!        01 - IBM-4341       02 - COBRA-530     !
!        03 - DEC-10                            !
!        CODIGO:01                              !
!     PRECISAO:                                 !
!        1 - Precisao Simples                   !
!        2 - Precisao Dupla                     !
!        3 - Complexo de Precisao Simples       !
!        4 - Complexo de Precisao Dupla         !
!        CODIGO:1                               !
!  VERSAO REQUERIDA.                            !
!     Codigo do Equipamento:01                  !
!     Codigo da Precisao   :1                   !
!     Distribuicao (S?, N?):n                   !
!     Otimizacao (S?, N?)  :n                   !
-------------------------------------------------

NAO FOI REQUERIDA CONVERSAO DE PRECISAO
NAO FOI REQUERIDA CONVERSAO DE AMBIENTES
NAO FOI REQUERIDA NENHUMA CONVERSAO NESTA SESSAO DE TRABALHO

FINAL DA SESSAO DE TRABALHO...
R  T=0.39/1.22 09 45 02
SP CON STOP
```