# PROGRAMAÇÃO SUPER-ESTRUTURADA E O GERADOR DE PROGRAMAS I-M-E

V.W.Setzer*


## RESUMO

Neste trabalho mostramos inicialmente porque a Programação Estruturada (PE) não resolveu os graves problemas da programação de computadores. Apresentamos uma nova estruturação de algoritmos, que denominamos Programação Super-Estruturada (PSE), de um nível de abstração muito mais alto que a PE, e que elimina a necessidade de se descrever detalhes necessários para o computador, mas que estão longe do problema que se quer resolver. A PSE pode ser usada tanto manualmente, como técnica de desenvolvimento "top-down" de programas e documentação, como automaticamente. Apresentamos características do gerador de programas I-M-E, que emprega as estruturas da PSE e gera programas em Pascal, e mostramos também alguns resultados do uso manual do novo paradigma de programação em desenvolvimento de software básico. Como subproduto do projeto foi também implementado o pre-processador de Tabelas de Decisão DECIS, que gera Pascal e C, usado no caso manual. Ambos os produtos estão no domínio público.



## ABSTRACT

In this paper we initially describe why Structured Programming (SP) has not solved the problems of computer programming. We present a new algorithm structure, which we called Superstructured Programming (SSP), of a much higher abstraction level than SP. It eliminates the need of describing in the program the details required by the computer, which are far apart from the data processing problem being solved. SSP may be used both manually as a top-down program development and documentation method, or automaticaly. We present an outline of the I-M-E program generator, which accepts the SSP structures and generates Pascal code. We also mention some results of the manual use of the SSP paradigm in the development of systems software. As a subproduct of the I-M-E project, we have also implemented DECIS, a Decision Table generator producing Pascal and C code, which has been used in the manual applications. Both products are in the public domain.


*Professor do Depto. de Ciência da Computação do Instituto de Matemática e Estatística da USP. Áreas de interesse: Compilação, Bancos de Dados, Linguagens de Programação, Desenvolvimento de Software, Educação em Computação, Computadores na Educação e Impacto Social e Individual dos Computadores. Endereço: Cx. Postal 20.570, 01498 São Paulo, tel. (011) 813-6199, FAX: (011) 814-4135. E-mail: VWSetzer@ime.usp.br

Este artigo é um resumo de trabalho escrito pelo autor [referência bibliográfica 14] enquanto estagiava na Phillips Universität Marburg, Alemanha, com auxílio do programa BID-USP (projeto 30.01).



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

## 1. Introdução.

A Programação Estruturada (PE), introduzida no fim da década de 60, não resolveu os problemas principais de se projetar, implementar e testar programas. A principal razão para isso é o fato de que a programação é feita em linguagens de um nível muito baixo. Isto se aplica tanto às linguagens algorítmicas mais empregadas (Fortran, Cobol, Basic, Pascal e C), como às assim chamadas "Linguagens de 4ª geração" (Natural/ADABAS, dBase, ZIM etc). Por exemplo, nestas últimas talvez o mais simples processamento "on line", que usa um arquivo seqüencial produzindo-se a exibição de um registro na tela, e passando-se para o próximo após apertar-se alguma tecla, exige o emprego de comandos algorítmicos.

A PE pode ser caracterizada como o emprego de três conceitos: desenvolvimento "top-down" ou por refinamentos sucessivos [3], modularização e o uso das três estruturas básicas - seqüência de comandos (**begin...end**), alternativas (**if...then...else**... e suas variações), e iteração (**while...do**... e variações). Böhm e Jacopini [1] provaram que qualquer algoritmo pode ser formulado usando essas estruturas básicas. Algumas linguagens forçaram o uso dessas estruturas, eliminando o **goto** de sua definição, como as citadas Natural e ZIM (até a sua versão 2). Empregando-se refinamentos sucessivos ou modularização, chega-se sempre a um nível em qûe se deve usar as estruturas básicas. A questão é que estas refletem não a estrutura dos problemas de processamento de dados, mas a do computador, isto é, de sua Linguagem de Máquina (LM). De fato, a menos de uma instrução de desvio, as instruções em LM são executadas em seqüência; os desvios condicionais e incondicionais da LM praticamente refletem a estrutura de controle do **if**. A iteração é uma das estruturas de controle mais típicas de LM: um desvio incondicional ou condicional para trás cria uma malha de repetição, um controle absolutamente essencial para que um programa finito possa resolver um número infinito de problemas de mesma classe.

Assim, podemos dizer que as três estruturas básicas da PE são aderentes à estrutura interna da máquina e não aos problemas de processamento de dados. Elas constituem apenas um "embelezamento" dessa estrutura simplificando, obviamente, a sintaxe e aumentando sua legibilidade, mas sem deixar de refletir a estrutura interna. É por isso que programas podem ser muito bem estruturados (dentro do esquema da PE) e ao mesmo tempo ser ininteligíveis - até pelo seu autor! Um programador obrigado a formular seu algoritmo nas três estruturas está sendo forçado a encaixar seu pensamento nos padrões apresentados pela estrutura interna do computador.

A Programação Super-estruturada (PSE), ao contrário, apresenta estruturas que fogem totalmente da estrutura interna da LM. Além disso, essas novas estruturas podem ser empregadas na estruturação de qualquer processo físico executado por seres humanos [11,14]. Ainda mais, seu uso pode servir de base para uma especificação de programas que permite a geração automática de código; a especificação pode servir de documentação do programa, que permanece então sempre atual.

Neste trabalho descrevemos os princípios da PSE e como foram usados na linguagem de especificação do Gerador de Aplicações I-M-E. Eles remontam a idéias de E.Marussi, que ele implementou em outro gerador, o IDT [11], que produz código em Cobol para ambientes de "mainframes" IBM. Esse gerador já foi exaustivamente testado em ambientes de processamento de dados administrativos, tendo a prática demonstrado que resolve praticamente todos os problemas provenientes da programação algorítmica naqueles ambientes. O I-M-E, que tem sintaxe diferente do IDT e contém conceitos não presentes neste último, foi desenvolvido para testar as idéias da PSE em ambiente de



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

programação científica e computacional, e gera programas em Pascal. Ele amplia e uniformiza vários conceitos do LDT. Infelizmente, conhecemos apenas parcialmente o LDT como estava em Setembro de 1988, quando sua equipe deixou de ter contato conosco, e somente através de algumas listagens de exemplos que conseguimos e dos contatos pessoais com E.Marussi, principalmente durante a confecção de [11].

A PSE pode também ser empregada manualmente na fase da programação por refinamentos sucessivos imediatamente anterior à da elaboração do código. Relataremos experiências bem sucedidas nesse sentido.

Mais detalhes sobre a PSE e o gerador de programas I-M-E podem ser encontrados em [14], que é bem mais extenso que este artigo. A implementação do gerador I-M-E e sua sintaxe completa podem ser encontrados em [6]. Uma discussão sobre os problemas causados pela programação algorítmica tradicional e algumas características do gerador LDT estão em [11].

**2. As super-estruturas.**

Dados em computadores apresentam-se sempre em conjuntos ordenados, ou melhor, em *seqüências*. Em alguns casos, a ordem dos elementos de uma seqüência é irrelevante, podendo-se ter então uma visão de conjunto no sentido da Matemática. Vamos denominar de *processo* a uma especificação de tratamento de uma seqüência abstrata de dados, derivada eventualmente de partes de diversas seqüências físicas, tais como telas, arquivos, tabelas, enumerações de valores ou mesmo seqüências unitárias como uma variável. Um *sub-processo* é um processo referenciado por outro. Uma *tarefa* de processamento de dados é composta por um processo e por vários sub-processos; estes devem ser referenciados ("chamados") pelo processo ou por sub-processos.

A especificação de um processo ou sub-processo é estruturada em dois níveis: 1) Tratamento *global* de uma seqüência (eventualmente abstrata) de dados; 2) Tratamento *elementar* dessa seqüência. O 1º nível especifica o que deve ser feito com a seqüência como um todo; o 2º como deve ser processado cada elemento da seqüência. Denominaremos esses níveis de *global* e *elementar*, respectivamente.

2.1. Nível global.

Este nível é estruturado nas seguintes *etapas*: *preparação*, *principal* e *encerramento*. Elas são indicadas na especificação pelas palavras reservadas **Initialization**, **Main** e **Ending**, que podem ser abreviadas por **I**, **M** e **E** (correspondendo a I, P e F do LDT). Elas são executadas nessa ordem, respectivamente *1* vez, *n* vezes e *1* vez, onde *n* é o número de elementos da seqüência abstrata de dados. A interpretação dessa estruturação é que qualquer processamento de uma seqüência de dados deve inicialmente sofrer uma preparação de seu uso, em seguida deve-se processar seus elementos, e finalmente encerrar o processamento colocando as coisas em ordem. Um exemplo seria a produção de um relatório: na etapa **I**, deve-se emitir o título do relatório, na **M** indicar como processar cada elemento da seqüência de dados que gerará o corpo do relatório, e finalmente na etapa **E** emitir o rodapé. Na etapa **I** deve-se ainda especificar a entrada e seleção de parâmetros globais necessários. Por exemplo, se o relatório trata dos dados dos funcionários de um departamento, nessa etapa pode-se entrar a sua identificação e consultar um arquivo onde estão os dados gerais como nome, endereço, etc. O processo de uma tarefa precisa ter pelo menos a etapa **M**; as outras duas etapas são opcionais. Cada subprocesso deve conter pelo menos uma etapa.

Note-se que esse nível de estruturação não é novo: proposta semelhante pode ser encontrada em trabalhos de Yourdon [16] e Warnier [15]. A diferença é conceitual: em nosso caso, essa organização deve ser aplicada a cada seqüência abstrata de dados a ser



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

processada, ao passo que eles a propõe para cada programa. Em outras palavras, o exame das estruturas dos dados que serão processados e o reconhecimento das seqüências que deverão ser tratadas leva à estrutura do programa, de modo que esta é conseqüência das estruturas das seqüências. Isso pode lembrar o método de Jackson [4], mas no nosso caso o nível de estruturação é muito mais alto, como veremos na subseção 7.6.

### 2.2. Nível elementar.

Cada etapa do nível global pode ser estruturada nas seguintes *partes: seleção, escolha* e *transformação/transporte*. Elas são indicadas na especificação pelas palavras reservadas **Selection, Choice** e **Transformations**, podendo ser abreviadas por **S, C** e **T**, correspondendo a L, D e T do LDT [11]. Elas são executadas iniciando-se pela seleção e passando-se em seguida para a escolha. Esta especifica quais transformaçoes da última parte devem ser executadas, e em que ordem.

A interpretação que deve ser dada a esse nível é o seguinte. Nele deve ser especificado como cada elemento da seqüência abstrata de dados deve ser processado. A parte **S** especifica quais elementos da seqüência devem ser tratados, e controla a varredura da seqüência. Cada vez que ela produz a seleção de um elemento, entrega-o às partes seguintes; quando o último elemento da seqüência foi selecionado e processado, esta parte produz o encerramento da etapa em que está contida. A parte **C** recebe um elemento da seqüência abstrata de dados, e testa condiçoes que ele satisfaz; dependendo da combinação dessas condições, executa uma seqüência ordenada de ações especificadas na parte **T**.

A estruturação deste nível é um ponto fundamental do método de PSF, e é nova, tendo sido introduzida pela primeira vez por E.Marussi no gerador LDT. Idealmente, a parte **C** deve conter exclusivamente condiçoes e referências a transformações/transportes de dados, e não ações correspondentes a estes últimos. Com isso, separam-se física e totalmente três diferentes conceitos: a seleção de um dado, a escolha das açoes que devem ser feitas com ele, e finalmente a descrição destas ações.

## 3. Esquema de uma tarefa e comparação com a PE.

O esquema de uma tarefa pode ser resumido como abaixo. Usaremos a notação [x] para indicar que x é opcional; {x}{y} indica que pelo menos um dentre x e y deve estar presente; {x}* indica zero ou mais ocorrências de x.

```
Process NomeDoProcesso
   [ I
        { S }
      { [ C ]
        T } ]
    M
        [ S ]
        [ C ]
          T
   [ E
        [ S ]
        [ C ]
          T ]
{ Subprocess NomeDoSubprocesso
   { I }
   { M }
   { E } }*
```



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

Deixamos de detalhar as etapas do subprocesso pois são exatamente as mesmas do processo. Vários subprocessos podem compor uma tarefa.

É importante constatar que essas estruturas não têm nada a ver com as estruturas internas do computador, como apresentadas pela sua linguagem de máquina e refletidas nas estruturas empregadas na PF. Uma destas últimas, a malha de repetição, foi totalmente eliminada. Na verdade existem malhas implícitas, que são geradas para varrer as seqüências de dados. Assim, dá-se uma noção conceitual a uma repetição; ela ocorre sempre que houver uma seqüência de dados a ser varrida. No nosso caso, o programador não especifica essa varredura: a própria estrutura das seqüências deve permitir ao gerador produzir o código com a repetição. Por exemplo, se a seleção é feita sobre um arquivo, haverá a geração automática de instruções para "ler" os registros, entregando um a um às partes **C** e **T**, bem como instruções para repetir essa "leitura" até que não haja mais registro a ser selecionado. O programador pode, portanto, dedicar-se à descrição, na parte **M**, do que o processo deve fazer com cada elemento da seqüência, sem preocupar-se em descrever a varredura em si. Por outro lado, vê-se a implementação de um conceito importante em programação: qualquer repetição deriva do processamento de uma seqüência de dados (dada ou a ser gerada). A eliminação das malhas de repetição explícitas e a geração de malhas implícitas para o processamento de seqüências de dados pode significar uma simplificação nas provas formais de programas. A dificuldade destas reside justamente na dedução dos denominados "invariantes das malhas". Aqui eles talvez possam ser deduzidos diretamente das estruturas de dados.

Vamos insistir: quando o programador estrutura seu algoritmo seguindo a PSE ele está impondo estruturas de controle ao computador. Se ele usa a PF a estrutura interna do computador impõe as estruturas básicas de controle.

**4. Uso de tabelas de decisão.**

Para implementar a parte **C** foi necessário escolher uma estrutura diferente do que se denomina "árvores de decisão" decorrentes da estrutura **if...then...else...** (ou **case**). A razão disso é que nessas árvores existe em geral uma mistura das condições de escolha com as ações escolhidas. De fato, em geral na seção do **then** aparecem alguns comandos de transformação e alguns comandos **if** com novas escolhas. Além disso, essas árvores de decisão têm um defeito essencial: elas requerem repetições de condições e de ações, o que dificulta a legibilidade e a compreensão do programa.

Tabelas de Decisão (TD - usaremos essa abreviatura tanto no singular como no plural) resolvem esses dois problemas. A solução do primeiro é absolutamente esssencial na PSE, pois deseja-se separar totalmente as partes **C** e **T**, tanto física como conceitualmente. Na verdade, as TD servem também para separar a parte **S** da parte **T**, tornando-se o recheio do sanduíche entre estas duas.

De acordo com Pollack et al. [8], TD têm sido usadas desde 1957, quando foram introduzidas no Integrated Systems Projects da General Electric. Muitos processadores de TD foram implementados; o último que achamos na literatura foi o de J.McCarthy [5]. Uma extensa resenha sobre TD encontra-se na primeira parte de [6], em que estão citados mais de 80 (!) artigos e livros sobre o assunto datando até o fim de 1990. Poder-se-ia perguntar: porque TD são tão pouco empregadas? Algumas das razões citadas na literatura são as seguintes: os artigos sobre TD são demasiadamente técnicos; elas requerem uma mudança de hábitos de pensar; processadores não eram disponíveis facilmente; há uma impressão falha de que TD não são suficientemente poderosas; há uma falta de ferramentas especiais para construir-se TD e testá-las. Cremos que as seguintes razões podem ser adicionadas a essa lista.



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

**R1**: Geradores de TD não são fáceis de usar. Por exemplo, o gerador GEM [2,5] processa uma tabela e gera código que é guardado em um arquivo. Posteriormente esse arquivo deve ser inserido por cópia no programa do usuário.

**R2**: TD não têm sido apresentadas como uma parte essencial de um método de programação.

**R3**: TD não tem constituído parte integrante de um gerador "universal" (isto é, que elimina a necessidade de se dar algum código em linguagem algorítmica) como o I-M-E ou o LDT.

O projeto do gerador I-M-E começou pelo projeto e implementação de um pre-processador de tabelas de decisão, que denominamos de DECIS, implementado por S.Nagayama [6] que acabou tornando-se um produto independente. Ele aceita um programa escrito em C ou em Pascal contendo TD no meio do código, e traduz as tabelas para comandos de uma dessas linguagens. As TD originais são convertidas em comentários da linguagem com a anotação de nível de abstração 2 do método de documentação descrito em [12], inserindo imediatamente abaixo delas, no programa do usuário, o código gerado pela tradução. Resultados do uso desse pre-processador em estruturação de programas segundo a PSE, sem o uso do gerador I-M-E e com tradução manual das estruturas **I**, **M**, **E** e **S**, **C**, **T** são apresentados na seção 11. Essas experiências vieram comprovar que a razão **R2** dada acima justifica por si só o uso de TD.

**5. Exemplo de especificação para o gerador I-M-E.**

Antes de descrevermos com algum detalhe a sintaxe dos comandos de seleção e as TD do gerador I-M-E, vamos dar um exemplo simples mostrando como se apresenta uma especificação completa. Após essa especificação tecemos alguns comentários para torná-la mais compreensível.

Suponhamos que se tenha um arquivo de uma certa disciplina, em que cada registro contém o nome de um aluno e 3 notas de provas que ele fez. Queremos produzir uma estatística de quantos alunos foram aprovados e reprovados na nota final, que deve ser calculada como a média aritmética das notas das provas. Esta estatística deve ser exibida em uma tela, com o nome da disciplina e o número de alunos da mesma. O nome da disciplina consta do primeiro registro do arquivo, no campo onde ficam os nomes dos alunos. A seguinte especificação resolve o problema. Ela foi testada devidamente, usando-se para isso o gerador I-M-E, tendo-se compilado e executado o programa resultante.

```
%Declarations

  %File Notas (Nome: string[20];
        Nota1, Nota2, Nota3: real) on 'Notas1.Ent' (text)
  %Screen Tela_Estat (Const 'Nome da disciplina: ': 1,10;  %-  1: linha ...
                                                           %- 10: coluna
        Var Nome: string, 1,30,20;                         %- 20: tamanho
        Const 'Numero total de alunos: ': 4,10;
        Var NumTotal: int, 4,34,3;
        Const 'Numero de aprovados: ': 6,10;
        Var NumAprovados: int, 6,31,3;
        Const 'Numero de reprovados: ': 8,10;
        Var NumReprovados: int, 8,32,3;
        Var Mensagem: string, 20,10,25)
  %Variable Media: real
  %Variable ContaNumTotal, ContaNumAprovados, ContaNumReprovados: int
```



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

```
%Process EstatisticaDeDisciplina

  %Initialization step
    %Selection part
      %Select Notas                          %- pega o nome da disciplina
    %Transformations part
      %Change Tela_Estat with Notas,         %- exibe nome da disciplina
          Tela_Estat.Mensagem with 'processando...'
      %Change ContaNumTotal with 0
      %Change ContaNumReprovados with 0
      %Change ContaNumAprovados with 0

  %Main step
    %S
      %Select Notas starting_at 2
    %C
      %Decision Table TestaMedia
        %Preparation
          %Exec CalculaMedia
        %Sets
          Passou    | >=5.0
          NaoPassou | <5.0
        %Conditions                |
          Media                    | Passou  NaoPassou
        %Actions
          %Exec ContaTotal         | 2       2
          %Exec ContaAprovados     | 1
          %Exec ContaReprovados    |         1
       %End Table
    %T
      %Transf ContaTotal
        %Inc ContaNumTotal
      %End ContaTotal
      %Transf ContaAprovados
        %Inc ContaNumAprovados
      %End ContaAprovados
      %Transf CalculaMedia
        %Change Media with (Nota1 + Nota2 + Nota3)/3.0
      %End CalculaMedia
      %Transf ContaReprovados
        %Inc ContaNumReprovados
      %End ContaReprovados

  %Ending step
    %T
      %Change Tela_Estat.NumAprovados with ContaNumAprovados,
          NumReprovados with ContaNumReprovados,
          NumTotal with ContaNumTotal,
          Mensagem with 'Acabou'

%End process
```



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

Faremos algumas breves observações para esclarecer o exemplo.

1) O gerador I-M-E ainda não conta com um diretório/dicionário de dados. Daí a necessidade de haver uma seção de declarações. Nesta são declaradas todas as estruturas, sendo que a visão de arquivos, telas e tabelas residentes é relacional. Como o código gerado está em Pascal, empregamos os tipos dessa linguagem, que são copiados tais e quais para código gerado (a menos da abreviatura **int**, em lugar de "integer").

2) As palavras reservadas indicadoras de comandos são em geral precedidas de '%'. Se em uma linha não aparecer inicialmente uma palavra precedida de '%', o gerador simplesmente copia essa linha para o código gerado; com isso pode-se inserir comandos em Pascal no meio das especificações. Os comentários são precedidos por %- valendo até o fim da linha. A razão do uso do caractere % foi de que o projeto foi incremental: aos poucos foram sendo cada vez mais eliminadas dos testes as linhas de código em Pascal. Estamos agora na posição de poder inverter essa notação.

3) Depois das palavras reservadas **Initialization**; **Main**, etc podem vir comentários ("step", "part", etc como usado no exemplo). Usamos também a forma abreviada dessas palavras reservadas para exemplificar.

4) Existe um único comando de entrada de dados, que é o **select**, sobre o qual discorreremos na seção 7. Quando ele atua sobre arquivos, provoca a abertura e o fechamento destes em cada etapa do processo ou subprocesso onde se encontra. Daí termos que especificar na etapa **M** que a seqüência de registros de alunos começa efetivamente no 2º registro. Quando esse comando aparece em etapas **I** ou **E** é feita uma única "leitura"; em etapas **M** são lidos todos os elementos da sequência de dados referenciada (é gerada u'a malha de repetição).

5) O comando **change** é o único que provoca a mudança de valor de variáveis, campos de registros de arquivos, campos de tabelas ou de telas. Quando tanto a origem do dado quanto o destino forem estruturados, é feita automaticamente a atribuição aos elementos de mesmo nome, numa ação semelhante à dos comandos **move corresponding** do Cobol ou **change** do sistema ZIM [17]. No exemplo, a primeira atribuição é feita usando os campos Nome.

6) As tabelas de decisão do gerador I-M-E seguem a estrutura introduzida no gerador LDT [11], compostas das 4 seções que se vê no exemplo, sendo as 2 primeiras opcionais (parece-nos que em LDT a 2ª é obrigatória). A seção de preparação deve estar presente quando os dados, obtidos na parte de seleção, não são adequados para a escolha de que ações devem ser feitas com eles. No caso, necessitamos da média das notas, mas ela não se encontra nos registros dos alunos. Uma transformação ("CalculaMedia") é executada antes do processamento passar à escolha propriamente dita. A segunda seção permite definir conjuntos de valores. Não encontramos algo semelhante na literatura de TD; ela aparece pela primeira vez no gerador LDT. Discorreremos um pouco mais sobre as TD do nosso gerador na seção 8.

**6. Visão unificada de dados.**

Introduzimos no gerador I-M-E um conceito unificado de dados. Do ponto de vista do programador, tanto faz qual a natureza do dado e onde ele se encontra fisicamente: os comandos de manipulação são sempre os mesmos. Em contraste, nas linguagens algorítmicas e de "4ª geração", em geral há comandos diferentes para, por exemplo, fazer acesso a uma tabela residente, um arquivo ou uma tela. Apenas a linguagem Mumps tem um tratamento um pouco mais uniforme, pelo menos não havendo distinção entre dado residente e não-residente. Parece-nos um absurdo ter que sobrecarregar o programador com uma preocupação que advém das instruções e estrututuras diferenciadas da



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

linguagem de máquina. No caso do I-M-E, um dado é simplesmente um dado, e ponto final. Isso significa que qualquer dado é tratado pelos mesmos comandos, independente de sua origem ou destino. Assim, há um só comando para modificar um dado, o **change**, como se pode ver em nosso exemplo na seção anterior: esse comando foi usado para mudar campos de tela e para atribuir novos valores a variáveis. O mesmo comando produz a alteração de registros de arquivos ou de campos de linhas de tabelas.

O comando que seleciona dados é o **select**, e também se aplica a qualquer classe de dados. Para localizar uma linha de uma tabela, é necessário usar uma visão relacional, e dar uma condição sobre um dos campos, como veremos na próxima seção. No I-M-E não há variáveis indexadas: os "arrays" devem ser sempre tratados como relações do modelo relacional de dados.

**7. O comando de seleção. Subprocessos.**

Uma sintaxe parcial do comando de seleção é a seguinte:

**select** {[( < número de elementos a serem selecionados > )]
< nome da seqüência de dados >
[**starting_at** < número de ordem do elemento inicial >]
[**where** < condição de seleção >] [**when** < condição de acesso >]
[**grouping by** { < nome de atributo >
**exec subprocess** < nome de subprocesso > || } $^{+}$ ] | |,} $^{+}$

onde [x] significa ocorrência opcional de x e {x||y} $^{+}$ pelo menos uma ocorrência de x, seguida por um número qualquer de yx, isto é, representa as cadeias x, xyx, xyxyx...

A seqüência de dados pode ser o nome de um arquivo, de uma tabela residente, de uma tela, de uma seqüência de valores (subintervalo, como 1..10 em Pascal) ou uma enumeração de valores. Na verdade, ainda pode ser um o nome de uma variável simples, mas nesse caso o comando não gera nada. Não tivemos coragem de levar até a última conseqüência nossa idéia de que qualquer dado, antes de ser usado, deve ser previamente selecionado, como se estivesse em uma unidade externa, da mesma maneira que um processo físico com algum material requer que se "pegue" esse material previamente. Nesse caso, uma variável simples teria que constar explicitamente de um **select** para que seu valor pudesse ser empregado.

A "condição de seleção" estabelece o critério de escolha do elemento da seqüência, como em linguagens de "4ª geração". A "condição de acesso" determina se o comando deve ser executado: se ela for falsa, o comando é ignorado.

Vejamos o significado desse comando em algumas situações típicas.

7.1. Seleção de elementos de uma seqüência.

O comando

**Select** S **where** CS **when** CA

gera u'a malha de repetição, que será abandonada quando não houver mais nenhum elemento da seqüência que pode ser selecionado, isto e, que satisfaz a CS. A cláusula CA pode também encerrar a varredura, se assumir o valor "falso".

7.2. Seleção 0-1:0-1.

O comando

**Select** $S_1$ **where** $CS_1$ **when** $CA_1$, $S_2$ **where** $CS_2$ **when** $CA_2$

produz uma única malha de repetição, com comandos de acesso a $S_1$ e $S_2$. Por exemplo,



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

se houver ainda uma outra seqüência $S_3$, $S_1$ poderia ser um registro de um arquivo, e $S_2$ e $S_3$ duas tabelas; dependendo do elemento selecionado de $S_1$, seria feito o acesso a apenas uma das tabelas. As CAs chaveariam qual tabela deveria ser usada. Um outro uso típico seria na intercalação de dois arquivos; um exemplo dessa aplicação encontra-se em [6].

7.3. Seleções 1:N.

A sequência de comandos

**Select** $S_1$ **where** $CS_1$
**Select** $S_2$ **where** $CS_2$ **when** $CA_2$

provê o resto da etapa do processo com um elemento abstrato de dados formado pela concatenação de um elemento de $S_1$ com um elemento de $S_2$; depois de executada a etapa, ela provê o resto da etapa com o mesmo elemento de $S_1$ concatenado com o próximo de $S_2$, e assim por diante até que se esgotem os elementos selecionáveis de $S_2$. Nesse ponto, é selecionado o próximo elemento de $S_1$, repetindo-se tudo outra vez. Em outras palavras, são geradas duas malhas de repetição encaixadas, com os acessos a $S_1$ e $S_2$. A cláusula $CA_2$ permite a formação de um elemento abstrato contendo apenas um elemento de $CS_1$. A seqüência de comandos acima, se colocada em uma etapa **I** ou **E** produz apenas um único elemento abstrato de dados.

Normalmente, $CS_2$ contém uma equação de junção, como no Modelo Relacional de dados [13]. Em uma versão para o gerenciador de bancos de dados ZIM que estamos orientando, os dois comandos **select** acima serão fundidos em um só. Este conterá a menção à ligação entre as duas seqüências, que provavelmente implementa um relacionamento conceitual, armazenada no dicionário de dados do ZIM [17].

7.4. Seleções 1:1(N)

Suponhamos que a parte S de uma etapa p contenha a seqüência

**Select** $S_1$ **where** $CS_1$
**Select** $S_2$ **thru subprocess** SP

Neste caso, para cada elemento selecionado de $S_1$ é chamado o subprocesso de nome SP, que é executado. Após a finalização de SP, é executado o resto de p. Isto produz uma junção entre os elementos de $S_1$ e o *resultado* da execução de SP. Todos os objetos de um processo e de seu subprocessos são globais. Em [11] dehominamos essa construção de "superjunção". O correspondente no LDT era diferente; nossa solução parece conceitualmente mais coerente.

Um exemplo de uso dessa construção seria a emissão de um relatório por departamento contendo dados deste (nome, endereço, etc) constantes de um arquivo (que faz o papel da seqüência $S_1$) concatenados à soma dos salários de seus funcionários, extraídos de um outro arquivo (que faz o papel de $S_2$). O subprocesso SP é que varre os elementos do arquivo de funcionários, calculando a soma de seus salários.

7.5. Agrupamentos.

Suponha uma seqüência de dados estruturados $S_1$, ordenada segundo o valor de um atributo A. O comando

**Select S where CS when CA grouping by A exec subprocess SP**

varre S e, sempre que existe uma "quebra" dos valores de A, o subprocesso SP é chamado. Sua etapa **E** é executada com o valor prévio de A (isto é, com o último elemento de S antes da quebra) e aí é executada a sua etapa **I** com o novo valor de A. Em SP podem ser executadas funções do que se denomina em Bancos de Dados de "funções de agregação",



PDF compression, OCR, web optimization using a watermarked

tais como somatórias, contagens, médias, etc, aplicadas aos dados dentro de cada subseqüência com mesmos valores de A.

Essa forma de seleção, que pode empregar um número qualquer de agrupamentos, foi usada (com outra sintaxe) no LDT. Ela é também semelhante ao que se encontra em qualquer gerador de relatórios, mas através dela pode-se entender conceitualmente certos comandos destes últimos. No caso do ZIM, por exemplo, o comando **report from** pode conter uma cláusula **break**, dentro da qual pode ser colocada uma seção **break heading** e outra **break footing**. A primeira corresponde à etapa **I** e a segunda à etapa **E** do subprocesso que trata dos agrupamentos. O corpo do relatório, o **detail line**, corresponde à etapa **M** do processo que contém o **select** e que chama o subprocesso.

Note-se que o subprocesso SP não deve conter uma etapa **M**, pois os elementos de S já são tratados na etapa **M** do processo chamador. O identificador S colocado após o **select** é tratado como um comentário.

7.6. Projeto de processos.

Apresentamos nesta seção alguns esquemas estruturais de comandos de seleção e de subprocessos. No projeto de especificações usando esse método, podemos sugerir os seguintes passos:

1) Reconhecimento das seqüências de dados envolvidas na aplicação.
2) Exame das ligações entre as seqüências (ou os relacionamentos conceituais), e o seu tipo (1:1, 1:N - também N:N se o Diretório-dicionário de dados contém informações sobre as ligações). Note-se que essas ligações podem ocorrer entre vários tipos de seqüências, como por exemplo entre telas e tabelas. O caso N:N, se implementado com uma relação auxiliar [13], deve ser tratado com a concatenação dos casos 1:N e 1:1.
3) Reconhecimento da maneira como as seqüências devem ser processadas (1:1, 1:N, 1:1(N)).
4) Projeto dos processos e subprocessos na estrutura I-M-E.
5) Projeto das estruturas S-C-T.

Dessa maneira, as estruturas de dados levam às estruturas dos processos. Aqui pode-se ver a diferença para com o método de Jackson [4]: este deriva das estruturas de dados as estruturas básicas da Programação Estruturada, ao passo que na PSE derivam-se estruturas de controle de muito maior nível. Além disso, temos uma estruturação que depende também da maneira como os dados são processados (item 3 acima).

**8. A parte de escolhas lógicas e as tabelas de decisão.**

A parte **C** é constituída exclusivamente de TD. Como afirmamos na seção 5, elas podem constar de 4 partes. A segunda, definição de conjuntos de valores, não foi encontrada na literatura, tendo sido originalmente introduzida por E.Marussi no LDT. A nossa é mais geral, pois permite a definição de conjuntos empregando todos os operadores usuais em qualquer número de níveis parentéticos. Como no LDT, o gerador I-M-E também permite definição de conjuntos de n-plas ordenadas, o que dá um enorme poder de expressão. Por exemplo, suponhamos que se queira testar a validade de datas entre os anos de 1981 a 1983 (estamos evitando os bissextos de propósito...). A seguinte declaração define o conjunto Data_valida com todas as datas desejadas:

**Sets**

Data_valida | ((1;3;5;7;8;10;12) : 1..31;(4;6;9;11) : 1..30; 2 : 1..29) : 1983..1985

O símbolo ';' indica união e ':' indica produto cartesiano. Um teste de validação de uma data representada pelas variáveis Dia, Mes e Ano seria então feita da seguinte maneira, na próxima parte da TD ('~' indica complementação):



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

**Conditions**
Mes:Dia:Ano | Data_valida -Data_valida

Só isso. O leitor deveria tentar escrever os comandos **if** correspondentes e comparar a legibilidade e concisão da notação acima.

**Def.** Um conjunto de valores é:

a) Uma constante ou variável (conjunto unitário);
b) Um símbolo relacional aritmético (<, <=, =, >=, >, <>) seguido de uma constante ou variável;
c) Um subintervalo de constantes ou variáves, na notação de Pascal ('..');
d) Uma expressão de conjuntos de valores. Sejam $C_1$ e $C_2$ conjuntos válidos de valores. Então uma expressão de conjuntos de valores é, na ordem de precedência decrescente:

$d_1$) $(C_1)$;
$d_2$) $-C_1$ (complementação);
$d_3$) $C_1 \& C_2$ (intersecção);
$d_4$) $C_1 \backslash C_2$ (diferença, complementação relativa)
$d_5$) $C_1 : C_2$ (produto cartesiano)
$d_6$) $C_1 ; C_2$ (união)

A geração de código transforma a pertinência a conjuntos em testes de comandos **if** sobre os intervalos ou enumeração de valores que definem os conjuntos.

Como se pode ver no exemplo da seção 5, a parte de condições das TD é um conjunto de pares ordenados, denominados de *linhas de condição*; os elementos de cada par são separados por '|', denominado de *separador*. O primeiro elemento de cada par é denominado de *descrição* da condição e o segundo de *entrada* da mesma [6]. As entradas podem conter uma ou mais colunas. Dizemos que uma linha de condição é *satisfeita* ou *não-satisfeita* por uma certa coluna, nos termos especificados adiante. O separador tem que estar sempre na mesma coluna, para todas as linhas de condição; essa coluna é determinada pela posição em que ele é colocado ao lado da palavra reservada **Conditions**.

Tanto o gerador DECIS (V. seção 4) quanto o I-M-E permitem o que se denomina na literatura de TD de *condições limitadas* e *estendidas*. Nas primeiras, as entradas das condições são exclusivamente 'Y', 'N', '-', '*' e '$', e aplicam-se a descrições de condições cujo valor seja booleano (comparações, variáveis, expressões, etc). Em uma condição limitada, se a sua descrição assume valor verdadeiro (falso), então as colunas com 'Y', '$' e '-' ('N', '*' e '-') são as que satisfazem a condição. O símbolo '-' é denominado de *indiferente*; '$' e '*' de *indiferente assumindo verdadeiro* e *falso*, respectivamente. Estes últimos permitem a otimização do código gerado, indicando-se que não é necessário fazer-se a verificação de verdade ou falsidade sabendo-se de antemão que o resulatado é verdadeiro ou falso, respectivamente. No caso de '-' não se sabe qual valor o resultado assume.

Uma outra extensão que introduzimos nos geradores DECIS e I-M-E foi a de permitir o que se denomina na literatura de TD de *condições estendidas*. Neste caso, as descrições são exclusivamente variáveis ou campos de dados estruturados (como tabelas, arquivos e telas), e as entradas podem assumir os seguintes casos:

1) Uma constante, variável ou campo. Uma coluna é satisfeita se ela assume o mesmo valor que a condição (teste de igualdade).
2) Um símbolo relacional aritmético ou alfabético (<, <=, etc) seguido de uma constante ou variável. Uma coluna é satisfeita se a expressão formada pela concatenação da descrição com a entrada dessa coluna assumir o valor verdadeiro.



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

3) O nome de um conjunto de valores declarado na parte **Sets**. Neste caso uma coluna é satisfeita se o valor da descrição pertencer ao conjunto especificado nessa coluna.
4) O sinal "indiferente" ('-'); a coluna é sempre satisfeita.

Especificamos acima os casos em que uma coluna assume a situação de ser satisfeita; o complemento da união dos mesmos dá a condição de ser não-satisfeita.

As entradas de uma TD devem ser agrupadas em colunas verticais, determinadas pelos símbolos mais à esquerda dessas entradas, denominadas de *colunas de entradas de condição*. Uma TD é *consistente* se e somente se existe apenas uma tal coluna para a qual todas as entradas são satisfeitas, para qulquer valor das decrições das condições.

Passemos às ações. O conjunto de ações é um conjunto de pares ordenados, chamados de *linhas de ação*. O primeiro elemento de cada par é chamado de *descrição da ação* e o segundo de *entrada da ação*. Como no caso das linhas de condição, os elementos de cada par são separados por um '|', colocado exatamente na mesma coluna do separador das condições.

Uma entrada de ação pode ser ou vazia ou um número natural, cujo dígito mais significativo deve ser escrito em uma coluna de condição. Em cada uma dessas colunas, as entradas de ação devem formar uma seqüência de números começando em 1. Uma coluna extra, escrita à direita da coluna de condição mais à direita, pode conter entradas de ações. Denominá-la-emos de *coluna alternativa*. Se nenhuma das colunas de condição é satisfeita, são executadas as ações especificadas na coluna alternativa.

A execução de uma TD é feita da seguinte maneira. Primeiramente é executada a ação especificada na parte de preparação, se existir. Em seguida são testadas as condições, em busca da coluna que satisfaz as mesmas. São então executadas as ações correspondentes às entradas de ação não vazia dessa coluna, na ordem da seqüência numérica.

No gerador I-M-E implementado por S.Nagayama, para gerar código para uma TD inicialmente todas as condições estendidas são transformadas em condições limitadas. É então escolhida uma linha de condição lc usando-se a heurística proposta por Pollack [7], que tenta minimizar o número de comparações geradas. Ela emprega uma estratégia denominada "quick rule method" [9] que tenta isolar uma condição o mais cedo possível. Uma vez escolhida a linha lc, a TD é fatorada em duas subtabelas onde lc não está mais presente, uma com as entradas de condição contendo 'Y', '$' e '-' e a outra contendo as entradas 'N', '*' e '-'. É gerado um comando **if** com a descrição da condição de lc, desviando para o código gerado recursivamente a partir das duas subtabelas.

TD podem ser encadeadas, isto é, uma descrição de ação pode especificar a execução de outra TD N, como numa chamada de procedimento, por meio da ação **exec table** N. Com isso pode-se fatorar manualmente condições, aumentando a ligibilidade. No exemplo da seção 5, as ações constituem chamadas de transformações especificadas na parte **T** da etapa do processo. O código gerado para TD, trtansformações e subprocessos emprega uma pilha combinada com um comando **case** para retornar à posição adequada depois da execução do trecho chamado. Não empregamos procedimentos porque a intenção original era gerar código tanto em Pascal como em C. Nesta, há um só nível léxico de funções, dificultando muito o uso de objetos globais. Por outro lado, procedimentos em Pascal têm que ser declarados no cabeçalho do programa principal ou dos procedimentos (não há uma estrutura de blocos no sentido de Algol), o que significaria a necessidade de inserir código em trechos já gerados. É interessante notar que as chamadas de TD, de transformações e de subprocessos teria sido enormemente simplificada se em Pascal (ou C) houvesse um comando semelhante aos *perform* e *gosub*



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

de Cobol e Basic. Por outro lado, não teríamos podido pensar em gerar código para o ZIM se não tivesse sido introduzido na linguagem o comando *goto*. Nota-se como os projetistas de linguagens não pensaram que elas pudessem servir de código objeto produzido por geradores automáticos. Gostaríamos de sugerir que projetistas de linguagens tivessem também essa meta em mente.

**9. Telas**

Telas de entrada (referenciadas em comandos *select*) são tratadas como elementos de seqüências de dados. Como tal, é necessário ao usuário indicar o fim de cada elemento (tela) e o fim de uma seqüência (de telas). Inspirados no ZIM [17], empregamos para isso as declarações **transmit_key** e **end_key** (**exitkey** no ZIM) mrespectivamente. Elas permitem ao usuário assinalar quais teclas ele deseja empregar para indicar esses dois términos. O emprego de **transmit_key all** permite a implementação de uma situação bem comum: qualquer tecla digitada pelo usuário indica que o processo deve continuar, após uma pausa de exibição de uma tela.

**10. Transformações**

Os seguintes comandos foram implementados no I-M-E para a parte T das etapas:

1) **Change**. Altera valores de variáveis, campos de tabelas, registros de arquivos, campos de telas e trechos de cadeias de caracteres.
2) **Insert**. Adiciona uma nova linha no fim de uma tabela, um novo registro no fim de um arquivo ou exibe uma tela.
3) **Exchange**. Troca duas variáveis, campos, registros, telas ou linhas de tabelas.
4) **Inc (dec)**. Incrementa (decrementa) uma variável ou campo com o valor 1 ou com o valor de uma expressão.
5) **Clear**. Limpa uma tela e inicializa seus campos variáveis.
6) **Exit**. Usado apenas em uma parte **M**; força o processo ou subprocesso a encerrar essa parte.
7) **Exec subprocess**. Chama um subprocesso.
8) **Remove left blanks from**. Remove espaços em branco mais à esquerda do valor de uma variável ou campo contendo caracteres. Pode-se usar **right** em lugar de **left**.

Note-se que algumas dessas transformações foram introduzidas por S.Nagayama para testar algum tipo especial de processamento, como o caso de (1) e (8) com cadeias de caracteres. Esses comandos deveriam ser estendidos conforme as necessidades detetadas em aplicações de qualquer tipo de processamento, principalmente simbólico para software básico.

As seguintes funções de agregação podem ser usadas nos subprocessos de agrupamento: **count**, **max**, **min**, **avg**, **total**, cujo significado é óbvio.

**11. Aplicabilidade e conclusões.**

Em termos de aplicabilidade, é necessário distinguir o uso do método I-M-E (que é praticamente o mesmo que o método LDT) no projeto, desenvolvimento e implementação de programas, e o uso do gerador de aplicações. O uso manual foi por nós testado através de projetos de implementação de compiladores para Pascal ou C, escritos em C, por nossos alunos de Compilação, combinando a PSE com o sistema de documentação de vários níveis de abstração que desenvolvemos [12] e o gerador de tabelas de decisão DECIS [6]. Como a linguagem fonte é C, pode-se colocar nas TD, nas descrições de condições, macros dessa linguagem, com nome sugestivo em termos dos problemas, e não os detalhes de programação. Estes ficam para as definições dessas macros, colocadas em geral imediatamente antes das TD. As descrições de ações têm



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

sido feitas com o comando **exec** como no exemplo da seção 5 referenciando transformações (disponíveis como rótulos no DECIS), ou também com uso de macros. O resultado tem sido excepcional, apesar de não se poder aproveitar em todo o desenvolvimento a facilidade do ambiente integrado de compilação e execução apresentado pelos compiladores modernos para micros, devido à necessidade de um passo de pre-processamento das TD. A clareza resultante da estruturação da PSE e do uso das TD é enorme. Os alunos aprendem a estruturar seus programas e documentá-los decentemente, principalmente ao nível do problema e não das linguagens algorítmicas de computação e de seus truques. Neste 1º semestre de 1991 estamos experimentando o uso da PSE também com alunos de uma disciplina de introdução à computação; temos obtido bons resultados com o nível mais alto (**I**, **M** e **E**), dado logo no início. Ainda não pudemos introduzir o esquema S-C-T (estes exigiriam tradução manual usando o **goto**) e as TD mas cremos que isso poderá ser feito um pouco antes do fim do semestre letivo, com o emprego do gerador I-M-E.

Nossa experiência com o uso manual do método da PSE na construção de compiladores leva-nos à conclusão de que o método é realmente universal, como já havia afirmado E.Marussi. Através de seu gerador LDT, instalado em muitos CPDs de grande porte, ele mostrou que o método pode abarcar praticamente 100% dos problemas de processamento de dados administrativos, eliminando completamente a programação algorítmica tradicional, e possibilitando que pessoas sem esse conhecimento gerem programas. Resta empregar o I-M-E em vários tipos de aplicações e espandi-lo adaptando-o às necessidades. Um aspecto interessante do LDT é um sistema de especificações por meio de menus, que elimina a sintaxe a menos, obviamente, de expressões. O gerador I-M-E teve como intenção servir de base para idéias e experiências de especificações e estruturações, daí termos desenvolvido uma linguagem para ele. A sua sintaxe completa, bem como detalhes da implementação, que usou nosso analisador sintático com correção automática de erros [10], e vários exemplos de especificações e código gerado podem ser encontrados em [6].

O leitor poderá ter achado que as especificações exemplificadas na seção 5 são decepcionantemente longas. Certamente um programa em Pascal que utilizasse as funções básicas de uso de telas seria bem maior. Além disso, nosso exemplo foi simples demais. Quanto mais complexa a lógica de um programa, tanto maior será o ganho das especificações da PSE em concisão. Note-se que várias linhas são na realidade supérfluas, como as que contém o **end** entre uma trasformação e outra. Pretendemos reduzir este e outros pontos em futuras versões. Em nossa experiência, uma especificação para o gerador I-M-E tem em geral 10 a 20 vezes menos linhas que o código gerado. Cremos que somente um programador com muita experiência poderia desenvolver um programa substancialmente menor que o gerado. De qualquer modo, é importantante considerar a clareza de nossa descrição em comparação com qualquer programa, e lembrar que ela constitui uma documentação sempre atualizada desde que não se mexa diretamente no código gerado, o que iria totalmente contra a idéia de se usar um gerador que não necessita de código adicional na linguagem gerada.

Os geradores DECIS e I-M-E estão disponíveis para cópia, requerendo um total de 3 diskettes de 5 1/4", bastando enviá-los ao autor. Infelizmente não contamos com um manual de uso do sistema, mas cremos que os exemplos e o grafo sintático são suficientes.



PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

**Bibliografia**


[1] Böhm, C. & Jacopini, G. Flow diagrams, Turing machines and languages with only two formation rules. *Comm. ACM* 9, 5 (May 1966), 366-371.

[2] Dlugosz, J.M. Logic GEM V.1.0. *Computer Languages* 6, 5 (May 1989), 131-132.

[3] Wirth, N. Program development by stepwise refinement. *Comm. ACM* 14, 4 (April 1971), 221-227.

[4] Jackson, M. *Software System Development*. Prentice-Hall, Englewood Cliffs (1983).

[5] McCarthy, J. Sowing seeds of productivity. *Computer Languages* 6, 6 (June 1989), 34-43.

[6] Nagayama, S. *Tabelas de Decisão e a Implementação do Gerador I-M-E*. Diss. de Mestrado, Instituto de Matemática e Estatística da USP (Dez. 1990).

[7] Pollack, S.L. Coversion of limited-entry decision tables to computer programs. *Comm. ACM* 8, 11 (Nov. 1965), 677-682.

[8] Pollack, S.L., Hicks, H.T.Jr. & Harrison, W.J. *Decision Tables: Theory and Practice*. John Wiley & Sons, N.York (1971).

[9] Pooch, U.W. Translation of decision tables. *ACM Computing Surveys* 6,2 (June 1974), 125-151.

[10] Setzer, V.W. & Melo, I.S.H. *A construção de um compilador* (5ª ed.). Ed. Campus, R. de Janeiro (1990).

[11] Setzer, V.W. e Marussi, E. *LDT - um gerador universal de aplicações para processamento de dados*. RT-MAC-8806, Depto. de Ciência da Computação, IME-USP, São Paulo (Junho de 1988).

[12] Setzer, V.W. *Um sistema simples para documentação semi automática de programas*. RT-MAC-8808, Depto. de Ciência da Computação, IME-USP, São Paulo (Set. 1988).

[13] Setzer, V.W. *Bancos de Dados - Conceitos, Modelos, Gerenciadores, Projeto Lógico e Projeto Físico* (3ª edição). Ed. E.Blücher, São Paulo (1989).

[14] Setzer, V.W. *Superstructured programming and the I-M-E program generator*. Bericht Nr.7, Fachbereich Mathematik, Phillips-Universität Marburg (1991), 34 pgs.

[15] Warnier, J.D. *Computers and Human Intelligence*. Prentice-Hall, Englewood Cliffs (1986).

[16] Yourdon, E. & Constantine, L. *Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice-Hall, Englewood Cliffs (1979).

[17] *ZIM* (trad. de *NIH's Guide to ZIM* por E.Cecassi). People Computação, Campinas, e Livros Técnicos e Científicos Editora, R. de Janeiro (1990).




PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor