TÉCNICAS DE PROGRAMAÇÃO II

Construindo um programa

© Prof. Engº Luiz Antonio Vargas Pinto
Versão 1.0 Julho de 1998

Neste trabalho, vamos fazer uma análise das técnicas de programação. Para isto, em primeiro lugar podemos começar dividindo este trabalho, tornando-o desta maneira mais claro, posto que se tentarmos englobar tudo o que existe sobre técnicas de programação sem que tenhamos uma visão isolada de cada uma das partes que compõe esse extenso e interessante universo é extremamente difícil. Assim, podemos começar, dividindo da seguinte forma:

Técnicas de Programação ...	Básico	Conceito de Programação
		Análise
		Construção
	Avançado	Análise de dados
Técnicas de armazenamento		Portabilidade
		Estrutura
		Entrada de dados
		Algoritmos
		Performance do sistema
Técnicas avançadas		

Técnica de programação básica

Como o próprio nome já diz são as técnicas que podemos usar para programação. Então, em primeira instância, precisamos conceituar o que é programação. Isto não deve envolver nenhuma linguagem de programação, porquê isto a tornaria específica, e nós temos de considerar os tratamentos estruturais e não da codificação de um programa. Então, partimos deste conceito para que entendamos o que é programação, porquê não tem sentido falar em técnicas para programar sem que ainda não se saiba o que é programar.

Análise

Veja bem, uma das dificuldades, talvez a maior que um programador de nível básico encontra, é transformar uma informação que ele tenha a nível de mundo real, isto é, um problema, uma idéia ou uma necessidade, em alguma coisa palpável, em alguma coisa funcional, no nosso caso, um programa de computador. Então, por exemplo, se eu precisasse de uma agenda teremos algumas questões por responder.

Primeiro - Qual o universo que uma agenda abrange ?
Segundo - Como eu poderia fazer uma agenda ?

Ou ainda um problema mais simples:

- Olhe, eu tenho três notas de cada aluno de uma classe com 40 alunos, e eu gostaria de fazer um programa que mediante a passagem das notas de todos os estudantes, calculasse a média da classe, a média de cada aluno, e ainda calculasse a evolução mensal da classe !”

Ou mesmo ainda, aquele problema do indivíduo que quer fazer um controle de estoque. Então ele precisa de um programa que possa fazer isso. Apesar do controle de estoque ser um programa simples em sua concepção, pois necessita muito mais do hardware do que software. Basta que você tenha uma caneta óptica no local e os códigos de barra nos produtos e você já está bem próximo da solução de seu problema. Você poderia gerenciar o seu estoque de forma indireta, por exemplo, com a saída de uma nota fiscal, que obrigatoriamente implica na saída de materiais do estoque, assim como na saída de uma ordem de serviço que implica que o produto que será fabricado obrigatoriamente termine no estoque. Assim es-

sa análise, essa capacidade de pegar um problema e transformá-lo em alguma coisa funcional á nível de máquina é o que nós vamos chamar de análise.

Construção de fluxograma

Em primeiro lugar, vamos esclarecer o seguinte: Por quê fluxograma ?

Muita gente, muitas vezes me pergunta porquê usar fluxograma se existem vários outros métodos para fazer estruturação de um programa a nível gráfico, muito mais funcionais que o fluxograma ?

Bem, existem várias razões. Primeiro porquê o fluxograma foi o primeiro método que apareceu, além disso é o mais simples que existe, e que consiste em mais ou menos 20 blocos e símbolos onde você define tudo aquilo que o computador precisa fazer. Isso facilita bastante o trabalho de ensino e o aprendizado porquê você não tem que se preocupar em aprender muitos conceitos, mas sim usá-los, isto é: Como escolher e ligar blocos ?

E isto é o que vamos ver no transcorrer deste tópico, isto é, como construir um fluxograma. Como transformar uma simples idéia em alguma coisa executável por um computador ou ainda, como transportar a idéia que está na minha mente para o papel sem que eu precise lançar mão de uma linguagem de programação. Caso você já tenha visto tudo isso, obviamente que este nosso primeiro contato esteja sendo bastante maçante, porquê como diz o ditado popular, estamos “chovendo no molhado”, ou tentando “ensinar o Padre Nosso ao vigário”. Chegamos a um momento em que estamos falando uma série de coisas que você, leitor, está saturado de ouvir, e gostaria de alguma coisa mais real, com mais riqueza de detalhes. Portanto, veja bem, essa é a razão pelo qual este trabalho está dividido em dois blocos. Eu tento atender aquele que esta começando e também aquele que já está programando, que também com certeza precisa de um suporte maior sem perder o brilho do trabalho. Eu recomendo que o iniciante leia tudo atentamente desde o começo, e ao experiente que se concentre nos temas que mais lhe interesse (embora a leitura do texto completo facilite bastante o seu aprendizado).

Técnica de programação avançada

No tópico avançado nosso maior objetivo consiste em dar uma base muito sólida, mergulhando em uma gama de assuntos o mais detalhada possível. O perfil técnico deste grupo é mais profissional, provavelmente constituído de um pessoal mais experiente, e evidentemente já com conhecimento básico e portanto com capacidade de organizar e montar seus próprios fluxogramas. Grupo esse, com capacidade de discernir idéias, transporta-las para dentro do computador, e dessa forma é mais razoável começar com a análise de dados.

Mas,... o que é análise de dados ?

Veja só que interessante, eu já havia comentado antes sobre um controle de estoque. Mas, um controle de estoque do quê? As quantidades são inteiras? É preciso guardar valores? Esses dados tem de ser convertidos para outra moeda? Ele vai entrar no meu ativo fixo ou é um ativo móvel? Então essa qualificação do meu ambiente, isto é, da minha idéia seria o segundo passo se fazer em cima do problema. Eu preciso conhecer profundamente o meio que me rodeia para explorá-lo adequadamente. E essa análise de dados é necessária porque é ela que vai me dizer que tipo de programa, que tipo de computador, que tipo de estrutura devo utilizar. Por exemplo, um banco de dados que gere simplesmente dois ou três tipos de relatórios não precisa de um elevado grau de sofisticação para ser criado, pois as rotinas que geram esses relatórios já vem embutidas dentro do próprio programa. Mas suponha que por alguma razão ele precise de um ambiente mais sofisticado, por exemplo, de um QUERY, porquê eu quero tirar relatórios variados, ou relatórios gráficos, ou específicos. Nesse caso eu preciso dar recursos ao sistema, e isso evidentemente depende do grupo de dados que vai ser tratado. Poderemos ver o significado da caracterização dos dados e da influência destes no tipo de programa.

Certo dia um cliente me perguntou sobre a possibilidade de ampliar os relatórios gerados em seu sistema, principalmente porquê ele temia que fosse difícil incluir esse novo rol de relatórios. Então eu lhe respondi que no instante em que nós fizemos a análise de seus dados, eu havia tomado o cuidado de selecionar e identificar todos os grupos existentes e a grande maioria daqueles que viessem a surgir em momentos futuros dentro do contexto da sua empresa. Claro que é impossível prever todos os tipos de modificações que possam vir a ser introduzidas dentro de um sistema já integrado, mas não impossível, desde que tenhamos em mãos os dados para isto.

O pior caso pode ser considerado quando o conjunto disponível de dados não permite a geração de um relatório de forma direta, o que obriga o uso de cálculos, muitas vezes extensos. Desta forma você pode observar a importância dos dados para um sistema, não? Mas, não se preocupe tanto com isto agora, atente apenas para o fato de que estamos seguindo uma lógica natural, e inclusive, no tópico seguinte, veremos as técnicas de armazenamento.

Técnicas de armazenamento

Uma vez que você conhece o meio onde você pode armazenar, e já sabe os dados que vão chegar as suas mãos, você já conhece os limites e características desses dados e portanto já sabe exatamente com o que trabalhar. Por exemplo: valores.

O meu controle de estoque é de arruelas. Tenho mais de 10.000 tipos diferentes de arruelas, mas cada tipo delas não custa mais que Cr$ 2,00 ou Cr$ 3,00 cruzeiros. É evidente que com valores como esses eu dificilmente me preocuparia em reservar muito espaço para armazená-los, até mesmo poderia pensar em reservar um valor inteiro para guardá-los mediante uso de alguma técnica de conversão, de uma maneira semelhante aquela usada pela linguagem FORTH. Mesmo assim é válida a questão de qual seria o tipo ideal para armazenamento: Uma matriz? Registros? Usar variáveis? Trabalhar direto no disco ou a memória é mais recomendável? Qual é a minha abrangência?

Então já estaria incorporada nessa fase as técnicas usadas para armazenar informações. No tópico 6 nós vamos ver as estruturas que existem para fazermos isto. Isto significa que as técnicas me “falam” como fazer, e as estruturas me “dizem” onde fazer. Por exemplo, vou utilizar uma matriz porquê dentro do contexto do meu programa é mais fácil trabalhar com índices e portanto, com indexação numérica. Por outro lado esses índices podem ser substituídos por ponteiros. Ora, então de acordo com a adoção do critério de técnica de armazenamento eu adoto uma estrutura de programa.

E é evidente que são diferentes, pois no primeiro caso, por exemplo, quando eu trato com matrizes, as minhas técnicas e rotinas de programação são voltadas para tratamento com índices numéricos ao passo que se eu utilizar tratamento com ponteiros de memória implica no conceito de tratamento com lista encadeada, e ainda com o conceito próprio de ponteiro, que em última instância é um endereço de memória, logo é claro que são conceitos diferentes. Inclusive, aproveito a ocasião para ressaltar que nesse caso eu normalmente utilizaria uma técnica de registro com ponteiro pois considero este um conceito mais avançado e cheio de recursos. Assim podemos concluir dizendo que as rotinas e consequentemente suas técnicas são diferentes. Mais adiante trataremos com mais detalhes a portabilidade.

Estruturas

Falar sobre estruturas não é algo tão simples quanto se possa pensar. Normalmente envolve o relacionamento entre os dados a serem coletados e o programa que os administrar. A construção de um programa deve obedecer a critérios rígidos de programação modular. Isto normalmente requer paciência na avaliação do meio á ser tratado. Por exemplo, no caso da construção de um banco de dados, podemos e devemos separar as seguintes coisas:

1º)Os dados consistem de informações tratar;

2º)A operação do sistema exige treinamento do usuário e uma interface homem-máquina bem elaborada;

3º)O tratamento dos dados está diretamente relacionado com os relatórios que o sistema gera;

4º)O gerenciamento dos dados é o programa propriamente dito, e cabe aqui, aos seus autores, a escolha da estratégia de tratamento dos dados,por exemplo: “Devemos usar um banco de dados tradicional ou um banco de dados relacional?”; “Usaremos dados normais em ASCII ou dados criptografados e comprimidos, e nesse caso, quais as técnicas de compressão devem ser utilizadas?”.

Portabilidade

Veja você que é muito importante hoje, com tantos sistemas existentes no mercado, que haja possibilidade de transportarmos dados fazendo transferência de informações de um sistema para outro.

Hoje em dia fazer um sistema fechado que não possua recursos, que não seja criativo o para permitir que dados sejam passados para outro sistema, é o mesmo que decretar que o seu sistema “nasceu” obsoleto e o seu tempo de vida num mercado como o de hoje praticamente inexiste. Em outras palavras, eu não compraria um sistema onde eu tenho dados que não posso transferir para um outro sistema. Observe que uma empresa normalmente possui pelo menos dois sistemas diferentes, ou até mais. E veja, mais grave ainda, se “amanhã” a empresa que me vendeu esse equipamento abre falência, o meu banco de dados ficara isolado porquê eu não teria mais suporte.

- Ah ! Mas nós lhe daremos o programa fonte !

Bom, mas o fonte é uma listagem e isto não é suficiente para compreendermos a idéia que o gerou. Porquê veja, um programa nasce de uma idéia e a partir disso ele termina no programa executável. Só que, entre você ter a idéia e conseguir transformar isso em um programa executável, muitas fases são atravessadas, muitas etapas são cumpridas. É difícil você transformar uma idéia em um programa e achar que todo mundo que tem aquela mesma idéia vai chegar naquele mesmo programa. Isso é impossível! Uma idéia pode gerar milhões de programas diferentes que gerem aqueles mesmos resultados, portanto não é assim que se deve pensar. E portanto a portabilidade de meus dados, principalmente no que se refere a um gerenciamento de um banco de dados, é o mais importante. Assim, deve ser possível coletarmos nossos dados, formatá-los e enviá-los para outro sistema. E essa portabilidade esta vinculada a estrutura, porquê existem estruturas onde é inviável o transporte de dados.

Algoritmo

Ainda acompanhando aquele passo de evolução, finalmente vamos chegar em uma coisa que chamamos de algoritmo. O que seria um algoritmo ?

Primeiro vamos tentar colocar as coisas bem claras dentro e fora de um programa. Um programa nasce de uma idéia ou de um conjunto destas.

- Eu gostaria de fazer uma determinada coisa! Eu sei que o computador pode fazer isso. E vamos partir da premissa de que isso seja possível de ser fei-

to. Ora, deve haver um caminho que, seguido naturalmente, me leve ao fim desejado. Porquê muitas vezes um programa se torna inviável por pequenos problemas. Por exemplo, algumas vezes a ficha, o banco de dados, o registro que ia ser montado fica muito grande, ou mau dimensionada. Então, considerando que tudo isso já foi tratado, é fácil nós constatarmos que existe uma seqüência natural entre a idéia e a transformação desta em um programa de computador. É um fim determinado, isto é, o computador processa os dados que eu passar para ele. Todos os caminhos, independente de como se faça isso precisa ser baseado em algoritmos. Algoritmos podem ser definidos como as etapas que se tem que cumprir para se chegar a um determinado fim.

Um algoritmo não precisa necessariamente ser um algoritmo matemático, este pode ser também um algoritmo lógico. Uma idéia tem de ser transportada para um forma de representação gráfica, que se chama fluxograma, porém essa idéia representada por esses gráficos e que faz uma função especifica é denominado algoritmo. Um exemplo: O algoritmo para fazermos uso de um telefone público. O algoritmo para fazermos a troca de um pneu. O algoritmo para fazermos café. No exemplo do telefone, este é o critério a ser seguido:

1º) Você retira o fone do gancho;
2º) Aguarda o tom de discar;
3º) Quando chega, se chegar, disque o número desejado;
4º) Aguarda a chamada ser completada;
5º) Se isto ocorrer, você fala com o destinatário e em seguida desliga;
6º) Se isto ocorrer, você responde o fone no gancho e, se assim desejar, pode repetir o procedimento á partir do item 1º.
7º) Caso o tom de discar não venha, então reponha o fone no gancho e repita o procedimento a partir do item 1. E, conforme você pode observar, existe uma seqüência a ser respeitada, e é sempre a mesma, o que caracteriza um algoritmo. Então um programa é cheio de algoritmos, desde a idéia básica, até a idéia global e inclusive o conjunto todo forma um grande algoritmo.

Performance

A performance do sistema é uma etapa muito importante porquê todo o sistema tem um destino. Se é meu, ele é de ordem particular, e sou eu quem vai determinar o rendimento deste. Por exemplo, na minha casa, uma agenda pode ser até relativamente lenta para organizar porquê eu não dependo daquela informação instantaneamente, no entanto para uma empresa isto já não é muito produtivo.

E todo esse trabalho de partir de uma idéia e chegar em um produto final, passando pela elaboração dos algoritmos, pelos fluxogramas, e pela codificação até chegar a mesa de teste. E, somente sair desta testado não somente em funcionalidade, isto é, com todas as opções do sistema disponíveis funcionando corretamente, assim como também ter um desempenho satisfatório.

Isto é pertinente a tempo de acesso ao banco, quantidade de relatórios, tipos de relatórios, facilidade de comunicação com o usuário, proteção contra uso indevido muitas vezes por usuários destreinados, ou, em outras palavras, do comportamento final.

É evidente que na construção de sistemas é impossível prever todas as intempéries que este pode sofrer, mas existem certos tipos de erros que tem e podem ser evitados a qualquer custo. Podemos citar por exemplo, o fato do usuário ter um acesso fácil a saída do sistema. Eu mesmo já usei alguns tipos de programa onde foi impossível achar a saída deste. Não que estivesse camuflada, mas deveria estar muito mais visível. Aproveito o momento também para lembrar que em certos casos essa camuflagem do comando de saída foi uma das formas para dificultar pirataria de programa, pois o programa atende com presteza o usuário que o comprou legalmente, mas é bastante incômodo para o pirata. De qualquer forma, nessa fase normalmente eu examino rigorosamente o programa e o reavalio.

- Ah, ele ficou lento!

ou
- Ah, ele ficou muito rápido!
- Não, ele é lento em determinadas condições!

Então isto significa dizer que a performance desse sistema é boa, porquê não citei apenas a velocidade como referência. Não se esqueça de outros problemas tal como a limitação de memória. Um programa que ocupa muito espaço de memória só roda em sistemas grandes, e ainda que tenha performance satisfatória exige equipamento sofisticado e caro. Isso restringe sua aplicação, e assim, mesmo que de boa qualidade acaba prestando-se apenas a serviços específicos.

Análise de técnicas avançadas

Finalmente, vamos a análise das técnicas avançadas de programação. É simples falar de algoritmos gerais, pois existem muitos livros que abrangem isso mas é a necessidade de algum tipo específico de algoritmo que impulsiona o programador a pesquisar ou criar o seu próprio. É essa condição que vamos explorar nesse tópico visando fornecer algumas ferramentas que permitam ao programador criar seus próprios algoritmos ou ao menos compreender como faze-lo.

Considerando que a maioria dos algoritmos baseia-se em alguns modelos consagrados, conhecer alguns deles pode ser benéfico nesse trabalho. E ainda mais, podemos chegar mais longe ensinando conceitos básicos de programação avançada associados aos algoritmos. Essas técnicas enfocam muitas vezes até o aspecto físico do computador e a própria ciência da computação, obrigando a reacender velhos conceitos esquecidos por falta de uso. Embora intrínsecos em muitas das mais modernas técnicas utilizadas atualmente, estes passam despercebidos. Por exemplo - o uso do conceito de sistemas operacionais remonta a década de 50, no entanto, ainda hoje se aplicam técnicas, que mesmo mais sofisticadas, ainda apresentam muita semelhança com aquelas. Outro exemplo é o algoritmo de compressão/descompressão de dados, onde recentemente li na revista PC RUNTIME, fevereiro/março de 1992, nas páginas de 2 á 7 uma aplicação do algoritmo de compressão de David Huffman datado de 1952. Não que eu pretenda fazer deste artigo uma sessão de história da computação, mas é interessante saber que certas aplicações de computador tiveram de aguardar até por algumas décadas antes de se tornarem práticas adaptadas a máquinas mais rápidas e com maior capacidade de memória. Entretanto veja que essas idéias são tão antigas quanto o próprio computador e no entanto ainda hoje são utilizadas e muito podemos aprender com elas.

O universo que envolve as idéias humanas é infinito e inesgotável, sendo somente equiparado a constante atração do desafio a vencer. Por essa razão, ao invés de ficarmos citando uma lista extensa de diferentes idéias vamos começar limitando nosso estudo a uma única idéia e estuda-la detalhadamente. Proponho construir um programa para cálculo de previsão orçamentaria.

Aproveito inclusive para frisar algo que já disse antes, relativo ao mundo que nos rodeia. Esta idéia não é tão recente, mas a maneira como surgiu já é motivo de estudo.

Certa tarde, conversando com meu cunhado, eu dizia para ele que pretendia economizar algum dinheiro para talvez no fim do ano poder tirar alguns dias de férias na praia. Claro, não esqueci que nossa economia é instável; nem momentaneamente pensei em guardar o dinheiro em casa, posto que a inflação gira em torno de 23% a/m e isto seria o mesmo que guardar papel velho, pois ao cabo de 12 meses esse dinheiro perderia completamente o valor.

Então ele me respondeu que a idéia parecia interessante, mas lembrou um problema: - Quanto devo depositar mensalmente?
e completou:
- Devo corrigir o valor da mensalidade ou posso depositar a mesma fixa todo mês?
e ainda:

- Será que a diferença entre um modo e outro é significativa? Ao que eu também questionei: - Quanto será que eu teria ao fim de doze meses?

Eu sei que pode parecer estranho, mas qualquer processo a nível de programação é sempre criado ou dessa forma ou de forma semelhante. Então podemos associar esta forma de exposição ao processo de análise de um software. Não existe um modelo padrão de levantamento de dados que possamos adaptar á qualquer problema existente. Geralmente o bom senso ainda é o melhor meio de analisarmos um determinado problema. Para isso, precisamos desenvolver um espírito aguçado e estarmos sempre atentos, mesmo ao menor sinal de evidência. É interessante observar que isto é um fato comum quando conversamos com algum analista, um detetive, um cientista, ou qualquer cidadão que já tenha alguma experiência de vida e bom senso.

Tornando este fato repetitivo, note que ser observador e detalhista caracteriza mesmo um defeito pessoal. Porém estar atento aos mínimos detalhes é o que caracteriza o profissional. Tome como exemplo disso o analista que descobriu a presença do primeiro vírus de computador, o detetive que desvendou um crime através de um pequeno detalhe que somente ele observou, ou do cientista que prestou atenção á atividade celular sangüínea e descobriu o vírus da AIDS. Mas, embora isto seja interessante, voltemos a análise de nosso problema considerando as condições de cálculo para determinar como o programa deveria solucionar as formas de evolução dos valores ao longo de 1 á n meses.

1º) O acumulo de valores é um procedimento bastante simples, basta darmos um valor inicial á variável que vai acumular os valores. Soma:=0; em seguida, dentro de um laço, “alimentamos” essa variável da seguinte forma:
Soma:=Soma + valor_a_ser_adicionado;

O que causará o acréscimo de uma parcela variável “valor_a_ser_adicionado” á cada “loop” executado, resultando em algo do tipo:

```
        :
        :
   Soma:=0;
        :
        :
Para i variando de 1 á 10 faça
inicio
  Leia (valor_a_ser_adicionado);
  Soma:=Soma + valor_a_ser_adicionado;
fim;
     :
     :
```

Com base nisto, passamos a fazer um “loop” de n meses. (Considere “Loop” como um ou mais ciclos fechados). Inicialmente coloquei isto na forma de 12 meses e com saldo inicial=0, isto é, começando com o depósito da primeira parcela sendo o único valor na conta. Nesse ponto meu cunhado lembrou-me de que seria interessante se pudéssemos definir um outro valor inicial que não zero, pois poderíamos ter alguma reserva, e por menor que fosse poderia interferir no resultado final. Tomando-o como um cliente em busca de algum programa específico considerei a observação pertinente. Note como a participação do cliente é importante no desenvolvimento de um sistema específico. A partir daí já passei a ponderar sobre a necessidade de criar um sistema mais aberto, onde o usuário pudesse definir alguns parâmetros com certo grau de liberdade.

O termo parâmetro é largamente utilizado em informática, sendo que sistemas tais como folha de pagamento são totalmente parametrizadas. É o mesmo que dizer que esse programa tem um “corpo” padrão e dados de referência inicial e que podem ser atualizados ou pelo usuário ou pelo próprio programa sendo que o melhor seria que o usuário pudesse definir sozinho qual seriam esses valores. (conside-

re o desconto de INSS ou imposto de renda como um exemplo disto). Analisando o nosso procedimento com bastante calma, pude observar os seguintes elementos que poderiam ser passados como parâmetro:

a) saldo inicial (residual já existente);
b) investimento (valor da parcela mensal);
c) corrige a parcela mensalmente ou não ?;
d) quantos meses de investimento ?;
e) Inflação mensal estimada em porcentagem.

É claro que isto pode eventualmente ter deixado lacunas, uma vez que é somente no momento do teste definitivo que realmente podemos definir se a idéia esta completa ou não.

2º) Tome isto como exemplo: Da primeira vez que experimentamos o sistema, este já apresentou resultados á contento, entretanto, rodava uma única vez e retornava ao DOS. Então adaptei um "loop" externo para questionar o usuário se este desejaria repetir o procedimento novamente.

3º) Conforme você mesmo poderá constatar na listagem seguinte, existe uma diferença muito pequena na rotina que corrige o valor da prestação mensal daquela que executa sem correção, de maneira que uma condicional "IF" foi introduzida para evitar a duplicidade de rotina com uma diferença tão pequena. Considerando ainda a alta velocidade de procedimento das máquinas atuais, mesmo um PC-XT não deixaria transparecer o atraso de tempo na decisão exigida em cada "loop".
4º) Apenas para efeito visual, e isto eu decidi de última hora, seria apresentado na tela os seguintes elementos:

- Valor da mensalidade á ser paga;
- Valor acumulado na poupança até o presente momento (isto significa que para cada mês haveria um valor diferente á ser mostrado);
- Valor da última mensalidade paga, o total acumulado no período de "n" meses, e o total acumulado das parcelas pagas no período de "n" meses.

Mais uma pausa para explicar a diferença entre o total acumulado no período de "n" meses e o total acumulado das parcelas pagas no período de "n" meses. É uma prática bastante difundida e conhecida entre poupadores sem conhecimento técnico do que é poupança, (diga-se de passagem que, descontando-se o desconhecimento técnico, do qual não sou também possuidor, resta o conhecimento de aplicação da matemática ao problema) de que esta valoriza e mantém o valor do dinheiro. Isto associado á outro grave problema, conforme tentamos exemplificar abaixo.

Se uma aplicação for composta de 10 parcelas de mesmo valor iguais de Cr$ 10.000,00 fica evidente que as 10 parcelas somadas implicam em um total de Cr$ 100.000,00. Considerando que a inflação acumulada no período foi de 100% isto conduz á idéia falsa de que teremos na poupança um total de Cr$ 200.000,00.

Veja os cálculos abaixo e conclua por si próprio:

Cálculo tradicional:

Parcela	Valor	Total	% sobre o acumulado
1ª	10.000	10.000	10
2ª	10.000	20.000	10
3ª	10.000	30.000	10
4ª	10.000	40.000	10
5ª	10.000	50.000	10
6ª	10.000	60.000	10
7ª	10.000	70.000	10
8ª	10.000	80.000	10
9ª	10.000	90.000	10
10ª	10.000	100.000	10

de onde 100.000,00 + 100.000,00 * 100% = 200.000,00

Cálculo real utilizado:

Parcela	Valor	Total	% sobre o acumulado
1ª	10.000	10.000,00	0
2ª	10.000	21.000,00	10
3ª	10.000	33.100,00	10
4ª	10.000	46.410,00	10
6ª	10.000	77.156,10	10
7ª	10.000	94.871,71	10
8ª	10.000	114.358,88	10
9ª	10.000	135.794.77	10
10ª	10.000	159.374,25	10

Total final = CR$ 175.311,67

Isto porquê as parcelas não sofreram correção, e observe mais, a porcentagem da correção é e deve ser mensal e sobre o valor em conta corrente e não sobre o valor total depositado. No caso real, no primeiro mês, você deposita Cr$ 10.000,00, mas os juros sobre esse dinheiro é somente depositado no mês consecutivo, e o seu primeiro saldo real seria obtido no ato do pagamento da 2ª parcela e seria obtido por:

10.000 + 10% + 10.000 onde: 10.000 é o saldo da conta;
1.000 é o equivalente de 10% sobre o total da conta
10.000 é o depósito da 2ª parcela

totalizando Cr$ 21.000,00 de saldo.

E é por essa mesma razão que para as duas comparações, teremos para o mesmo período, no modo real Cr$ 175.311,67 e não os Cr$ 200.000,00 esperados.

Foi esta a conclusão, trágica por sinal, á que chegamos, que na verdade representa um déficit de Cr$ 24.688,00 entre um modo e outro. Porém, observe que para o primeiro caso, o poupador pensou que no mês do primeiro depósito ele já receberia os primeiros 10% equivalentes aos juros do primeiro mês, mas se “esqueceu” de que os juros são pagos sobre o dinheiro aplicado por pelo menos 30 dias, e que no ato do pagamento da primeira parcela antes havia Cr$ 0,00 em sua conta mas este gostaria de receber 10% de juros como se já houvesse dinheiro lá aplicado. Por esse motivo coloquei no programa, quando este acaba de rodar, a soma das parcelas pagas e o saldo na conta corrente real para estabelecer essa comparação e expor dados reais muito importantes para a construção de um programa real e funcional. Embora a fantasia de ganhar dinheiro fácil seja sempre atrativa, fabricar um programa que mostra dados irreais é expor o cliente á uma série de problemas graves.

5º) A rotina “Formata” foi desenvolvida com princípios simples de manipulação de Strings para proporcionar uma visualização mais adequada dos valores calculados e apresentados. Neste exemplo, todos os dados tratados são do tipo Real, isto é, em notação de ponto flutuante, e portanto exponenciais para obter a máxima precisão. Entretanto, por motivos óbvios de que o nosso dinheiro trabalha com apenas duas casas decimais, o número real é formatado com duas casas decimais e os locais de inserção de pontos decimais são calculados de forma simples, por faixas de valores, e devidamente inseridos no string. Por fim a vírgula é colocada no local adequado e o string todo é apresentado.

6º)Como falamos de programa, e no caso, programação PASCAL, aproveitamos para esclarecer melhor o mecanismo de funcionamento de rotina leitura. Em PASCAL, atualmente falamos, no mínimo de TURBO-PASCAL, e no mínimo de versão 5.0. Quem trabalhou com TURBO-PASCAL versões 3.xx no mínimo estranhou o tratamento das procedures de entrada de dados, e principalmente no comando READ. Aliás, mais precisamente no caso de READ de Strings. Quando se aprende PASCAL em versões iguais ou inferiores á 3.xx, aprendemos que a diferença entre READ e READLN consiste unicamente no fato de que o comando READ lê o dado e coloca-o na variável correspondente e, o comando READLN lê o dado e também coloca-o na variável correspondente, mas executa um Line Feed colocando o cursor na coluna 1 da linha seguinte. Já, quando tratamos de versões superiores a esta, isto já não pode ser tratado desta maneira tão simples. Houveram algumas mudanças sistemáticas quando partimos de versões 4.xx. Uma delas, o comando READ e o READLN mudaram significativamente. Não é mais a presença do Line Feed apenas, mas toda a estrutura dos comando foi alterada. Assim, quando um String é lido, se tentarmos ler, logo em seguida outro String, o sistema interpreta como se este e os demais consecutivos já tivessem sido lidos e não para ler os dados. Ocorre que já é praxe de programação não lermos dados numéricos de forma direta, uma vez que a tentativa de ler um dado numérico e a respectiva digitação de uma letra qualquer misturada á esses números acarreta um erro grave e o sistema é interrompido e retorna ao sistema operacional. Para evitarmos este tipo de erro, entramos com strings e somente após isto tentamos converter o dado recebido para número. Existe em TURBO-PASCAL uma procedure que faz esta função, e mais, ainda testa se é possível a conversão ou não, fornecendo uma variável de teste indicando o resultado da conversão. Por outro lado, agora temos um outro problema: PASCAL não aceita ler duas ou mais Strings consecutivas, o que é um difícil obstáculo para programação em geral. Então, como contornar este fato? O próprio manual original do Turbo-Pascal da Borland® contém os elementos para solucionar esse problema, inclusive com exemplos simples para isto. Neste caso, em particular, eu desenvolvi a rotina “Leitura” que funciona como um comando READ ou semelhante. Agora, digite o programa exemplo, compile-o em turbo Pascal 5.0 ou maior e bom divertimento:

```
Program poup;
Uses Crt,Turbo3;
Type;
  Str20 = String[20];
Var
 Titulo,Bufer:Str20;
 Saldo,Soma,Prestacao,Parcela,Valor,correcao,Total:Real;
 a,Opcao:Char;
 Test:Boolean;
 meses:Word;
 i:Byte;
 m:Integer;
```

```
Function Leitura(Var flag:Boolean;Bypass:Char;Maximo:Byte):Str20;
Var
 Primeiro:Byte;
 x:Char;
 funcao:Boolean;
 m,n:Integer;
 Buf:Str20;
Begin
  Buf:='';
  Flag:=False;  { Padräo sem mudança }
  Primeiro:=0;
  If Bypass=' ' Then Buf:='' else Buf:=Bypass+Buf;
  Repeat
    x:=Readkey;
    if x<>#0 then funcao:=False else
    Begin
      funcao:=True;
      x:=Readkey;
    end;
    If not funcao Then
    Begin
      m:=length(Buf);
      n:=ord(x);
      Case n of
      32..126:Begin
                If m<Maximo Then
               Begin
                 Flag:=True;
                 Buf:=Buf + x;
                 write(x);
               end;
             end;
      8 :Begin
           If m>0 Then
           Begin
             Flag:=True;
             write(x);
             write(' ');
             write(x);
             Delete(Buf,m,1);
           end;
         end;
      end;
    end;
 Until ((Ord(x)=68) and (funcao)) or ((Ord(x)=13) and (Not funcao));
 Leitura:=Buf;
end;

Procedure Formata(Var formatado:Str20;x:Real);
Var
  forma:LongInt;
Begin
  Str(x:16:2,formatado);
  If x<2147483647 Then
  Begin
    forma:=Trunc(x);
    If forma>999 Then
    Begin
```

```
      insert('.',formatado,11);
      Delete(formatado,1,1);
    end;
    If forma>999999 Then
    Begin
      insert('.',formatado,7);
      Delete(formatado,1,1);
    end;
    If forma>999999999 Then
    Begin
      insert('.',formatado,3);
      Delete(formatado,1,1);
    end;
    Delete(formatado,14,1);
    insert(',',formatado,14);
  end;
end;

Begin
  Repeat
  Soma:=0;
  ClrScr;
  Saldo:=0;
  GoToXY(1,5);
  write('Qual o seu saldo inicial:');
  Repeat
    GoToXY(27,5);
    Test:=True;
    Bufer:=Leitura(Test,' ',20);
    Val(Bufer,Saldo,m);
  Until m=0;
  GoToXY(35,5);
  ClrEol;
  Formata(Bufer,Saldo);
  write(Bufer);
  GoToXY(1,6);
  write('Qual o seu investimento inicial:');
  Repeat
    GoToXY(35,6);
    Test:=True;
    Bufer:=Leitura(Test,' ',20);
    Val(Bufer,valor,m);
  Until m=0;
  GoToXY(35,6);
  ClrEol;
  Formata(Bufer,valor);
  write(Bufer);
  GoToXY(1,7);
  write('Você quer corrigir a parcela mensalmente ou näo ?');
  Repeat
    Opcao:=Readkey;
    opcao:=UpCase(opcao);
  Until opcao in ['N','S'];
  write(opcao);
  GoToXY(1,8);
  write('Quantos mêses:');
  Repeat
    GoToXY(16,8);
```

```
    Bufer:=Leitura(Test,' ',20);
    Val(bufer,meses,m);
  Until m=0;
  GoToXY(1,9);
  write('Taxa mensal estimada (%):');
  Repeat
    GoToXY(27,9);
    Bufer:=Leitura(Test,' ',20);
    Val(bufer,correcao,m);
  Until m=0;
  Prestacao:=Valor;
  Soma:=Prestacao;
  Total:=Saldo + Valor + (Valor*correcao/100); {equivalente ao 1º mês}
  For i:=2 to meses do
  Begin
    If opcao='S' Then Prestacao:=Prestacao + (Prestacao*correcao/100);
    Soma:=Soma+Prestacao;
    GoToXY(1,19);
    write('Valor da ',i,' mensalidade  ser paga = Cr$ ');
    Formata(Bufer,Prestacao);
    write(Bufer);
    GoToXY(1,20);
    write('Total acumulado no ',i-1,' mês = Cr$ ');
    Formata(Bufer,Total);
    write(Bufer);
    a:=ReadKey;
    Total:=(Total + Prestacao) + (Total + Prestacao)*correcao/100;
  end;
  GoToXY(1,19);
  write('Valor da última mensalidade paga = Cr$ ');
  Formata(Bufer,Prestacao);
  write(Bufer);
  GoToXY(1,20);
  write('Total acumulado no período de ',meses,' meses = Cr$ ');
  Formata(Bufer,Total);
  write(Bufer);
  GoToXY(1,21);
  write('Total acumulado das parcelas no período de ',meses,' meses = Cr$ ');
  Formata(Bufer,Soma);
  write(Bufer);
  GoToXY(1,25);
  write('Você gostaria de refazer os cálculos novamente ?');
  Repeat
    a:=Readkey;
    a:=UpCase(a);
  Until a in ['S','N'];
 Until a='N';
end.
```

Uma outra técnica, bastante interessante de estudar são os ponteiros, também tratados como POINTERs. Para isto vamos estudar as variáveis de Alocação Dinâmica. Este tipo de estrutura fornece apoio para grandes ocupações de memória sem a prévia reserva desta que proporciona maior rendimento dos programas que não necessitam pelo menos inicialmente, da totalidade de espaço reservado.

Por exemplo se desejamos arquivar informações sobre produtos é óbvio que este cresce á medida que novos produtos vão sendo incorporados. Ora, se trabalhamos com matrizes é necessário saber o máximo espaço que será usado e este deve ser reservado na sua totalidade mesmo que nenhum seja usado. Já por outro la-

do, se usamos registros (Records) temos a opção de utilizar apenas o espaço dos produtos que foram cadastrados ou seja se um produto requer 200 Bytes de informação e prevemos 1.000 produtos, como uso de matrizes seremos obrigados á reservar:

```
  200
x 1.000
--------
200.000
```

Veja, **200.000 bytes**.

Mesmo que nenhum produto esteja cadastrado. Por outro lado, com o uso de memória de alocação dinâmica se tivermos um produto este consome apenas 20 bytes mas se não tivermos nenhum não teremos nenhum gasto de memória daí a denominação memória de alocação dinâmica. Até mesmo porquê incluir ou excluir dados não altera o tamanho da matriz mas as estruturas dinâmicas “expandem” ou “encolhem” consumindo apenas a memória necessária. Pascal adota dois elementos para este fim:

1. Pointer
2. Record

Fila encadeada:

Um dos procedimentos mais interessantes de estrutura de programação é a lista que consiste tal e qual aquela que usamos no nosso dia-a-dia como no caso de uma lista de compras para supermercado. Por exemplo:

- latas de óleo
- litro de vinagre

Neste caso esta é uma lista simples onde não existe nenhum elo de ligação entre óleo e vinagre. Entretanto podemos ter determinadas listas que podem ter elementos relacionados entre si em uma estrutura onde a consulta de um dado elemento além dos dados intrínsecos deste informa a localização do próximo, e a esta técnica denominamos lista encadeada. A grande vantagem de operar uma lista encadeada é que os elementos de ligação, os elos, são ponteiros os quais fazem com que a primeira conseqüência disto seja a grande velocidade de operação na alterações. Isto é uma decorrência do fato de que os ponteiros são variáveis do tipo Inteiras, porém cujo conteúdo são endereços de memória e para usá-los basta apenas atribuir-lhes valores inteiros. Há muitas aplicações para este tipo de estruturas tais como Organização de registros em um arquivo de acesso aleatório de disco, Cadastro de um banco de dados onde os pointers são usados como elemento de ligação entre os registros, em outras palavras o índice ou ainda como já usamos, o ponteiro para o próximo registro. A título de exemplo, damos a seguir a listagem de um programa didático em Pascal para implementar uma lógica de encadeamento de registros com controle de variáveis com elementos POINTER.

Já falamos bastante sobre estrutura e constantemente mencionamos POINTER sem especificar exatamente o que seria o tipo POINTER. Em programação tradicional é comum falarmos em matrizes ou vetores, que é o momento onde nos referimos ao acesso indireto dos elementos, armazenados ou a armazenar. O índice constitui o meio de acesso á cada um desses elementos matriciais. A nível de programação de alto nível, porém em memória as coisas não podem ser tratadas da mesma forma que na estrutura de alto nível, até mesmo porquê a memória é um elemento linear, isto é os endereços de acesso começam em 0 e vão até o limite desta memória em uma seqüência com passo unitário e cujos valores são números inteiros: 1, 2, 3, 4, etc..

Assim o acesso direto em memória será feito por meio de endereços físicos do computador que é um número inteiro (por exemplo, nos micros de 8 bits são inteiros de 0000 á 65535)e são tratados diretamente em memória os acessos.

E os POINTERS?

Para ligar e acessar as estruturas dos registros (Records) fazemos uso de um tipo de variável denominada POINTER que contém um valor inteiro que é um endereço de memória. Ocorre que para programação não nos é facultado o poder de atribuir valores numéricos á estas variáveis como se fossem varáveis inteiras comuns, mas apenas podendo trocar entre si. Assim um POINTER será constituído de um elemento inteiro, porém restrito a aplicações de endereçamento de memória.

```
Program fila_encadeada;
Const
   Max_Reg := 3;
Type
   Ponta = ^Pointer;
   Pointer = Record
        Indice:Ponta;
         Nome:String[10];
         Code:Integer;
        end;
Var
  Reg,Reg_Buffer:Pointer;
  Prilin,Newlin,Prolin,Freelin:Ponta;
  i:Integer;
Begin
  Prilin:=Nil;
  For i:=1 to Max_reg do
  Begin
    New(Newlin);
    Write('Product:..');
    Readln(Newlin^.Nome);
    write('Code:.....');
    Readln(Newlin^.Code);
    If Prilin = Nil Then
     Prilin:=Newlin
    else
     Prolin^.Indice:=Newlin;
    Newlin^.Indice:=Nil;
    Prolin:=Newlin;
  end;
  { Primeiro método de acesso:
   Observe que eu controlo o número de registros que possuo }
  Freelin:=Prilin;
  For i:=1 to Max_Reg do
  Begin
    write('Name_of_Product:=');
    writeln(FreeLin^.Nome);
    write('Code_of_Product:=');
    writeln(Freelin^.Code);
    Freelin:=Freelin^.Indice;
  end;
  { Segundo método de acesso:
   Observe que eu busco o fim da lista mas não sei onde este ocorre}
  Freelin:=Prilin;
  Repeat
   write('Name_of_Product:=');
   writeln(FreeLin^.Nome);
   write('Code_of_Product:=');
   writeln(Freelin^.Code);
   Freelin:=Freelin^.Indice;
  Until Freelin=Nil;
```

end.

Efetivamente estas são sem sombra de dúvida as ferramentas mais importantes de programação e embora pareça complexa a primeira vista se você analisar cuidadosamente verá que o princípio é bastante simples e fácil de ser compreendido.