Edwar Saliba Júnior
Novembro de 2007

Estruturas de Dados

Notas de Aula

Faculdade de Tecnologia INED
Belo Horizonte - MG

Sumário

1. Nota:

Os códigos em Linguagem C apresentados neste trabalho foram criados utilizando a ferramenta conhecida por “Dev-C++” versão “4.9.9.2”. Esta ferramenta poderá ser adquirida gratuitamente no site abaixo:

http://www.bloodshed.net/devcpp.html

2. Tipos de Dados e Tipo Abstrato de Dados

Segundo Mizrahi (1990) e o Centro Tecnológico de Mecatrônica de Caxias do Sul (1997), a Linguagem C possui cinco tipos básicos de dados. Seus tipos, tamanho, intervalos e uso podem ser vistos abaixo:

2.1. Comuns:

Tipo	Tamanho	Intervalo	Uso
char	1 byte	-128 a 127	Número muito pequeno e caracter ASCII
int	2 bytes	-32768 a 32767	Contador, controle de laço
float	4 bytes	3.4e-38 a 3.4e38	Real (precisão de 7 dígitos)
double	8 bytes	1.7e-308 a 1.7e308	Científico (precisão de 15 dígitos)
void	0 bytes	Sem Valor	Ponteiro para nulo

2.2. Modificados:

Tipo	Tamanho (bytes)	Intervalo
unsigned char	1	0 a 255
unsigned int	2	0 a 65 535
long int	4	-2.147.483.648 a 2.147.483.647
unsigned long int	4	0 a 4.294.967.295
long double	10	3.4e-4932 a 1.1e4932

2.3. Tipo Abstrato de Dados (TDA):

De acordo com Tenenbaum et al. (1995), o mecanismo conhecido como TDA, é uma ferramenta muito útil para especificar as propriedades lógicas de um tipo de dado. Fundamentalmente, um tipo de dado significa um conjunto de valores e uma seqüência de operações que podem ser realizadas sobre estes valores.

Para deixar mais claro esse assunto será tratado mais adiante, em nosso curso, quando estivermos definindo, com o auxílio de ponteiros, os TDA's: Pilha, lista e fila e suas respectivas operações.

2.4. Tipo Enumerado:

O tipo enumerado é muito utilizado para facilitar a criação e, principalmente o entendimento do programa. Vamos supor a seguinte situação: Um cliente pediu-nos que desenvolvêssemos um programa onde o usuário digite um número entre 1 e 7 (inclusive) e, o software retorne o dia da semana correspondente. Apesar de muito simples, didaticamente falando este programa é excelente para o que queremos demonstrar.

No código a seguir, você poderá observar que, para atender a solicitação de nosso cliente, foi criado um software que solicita a entrada de um número de 1 a 7 e logo em seguida, através de uma estrutura “switch”, o programa devolve o dia da semana ao usuário, de acordo com o número digitado. Por exemplo, se o usuário digitar o número 3 (três), automaticamente aparecerá na tela a palavra “terça-feira”.

Agora, se observarmos atentamente a estrutura “switch”, veremos que ao invés de testarmos a variável “num” com números (1, 2, 3 e etc.), estamos testando com as iniciais dos dias da semana (seg, ter, qua e etc.).

Isto foi possível devido à criação de um tipo enumerado chamado “dias_semana” e, logicamente, uma variável chamada “dia” do tipo “dias_semana”.

Ao criarmos um tipo enumerado na Linguagem C, temos que ter em mente que todas as palavras que comporem o tipo corresponderão a um número inteiro e seqüencial, iniciado em zero. No exemplo abaixo, “dom” vale zero, “seg” vale um, “ter” vale dois e assim por diante.

```
#include <cstdlib>
#include <iostream>

using namespace std;

typedef enum { dom, seg, ter, qua, qui, sex, sab } dias_semana;

int main(int argc, char *argv[])
{
    dias_semana dia;
    int num;

    cout << "Digite um número de 1 a 7: ";
    cin >> num;

    cout << "\n\n";

    switch (num - 1) {
           case dom:
                cout << "Domingo";
                break;
           case seg:
                cout << "Segunda-feira";
                break;
           case ter:
                cout << "Terça-feira";
                break;
           case qua:
```

```
                cout << "Quarta-feira";
                break;
           case qui:
                cout << "Quinta-feira";
                break;
           case sex:
                cout << "Sexta-feira";
                break;
           case sab:
                cout << "Sábado";
    }

    cout << "\n\n";

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

2.5. Constantes:

A declaração de valores constantes pode ser efetuada de duas formas. A primeira delas é usando a diretiva do pré-processador conhecida como “#define”. E a segunda é usando a palavra reservada “const”.

No exemplo a seguir, poderemos verificar que serão criadas duas constantes através do uso da diretiva “#define”. Uma constante se chama INICIO e tem seu valor igual a 1 (um) e a outra constante tem o nome FIM e tem seu valor igual a 1000 (mil).

O software apresentado a seguir imprime no vídeo os números de 1 a 1000:

```
#include <cstdlib>
#include <iostream>

#define INICIO 1
#define FIM 1000

using namespace std;

int main(int argc, char *argv[])
{
    int i;

    for (i = INICIO; i <= FIM; i++)
      cout << "\n" << i;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

No exemplo a seguir, poderemos verificar que serão criadas duas constantes através do uso da palavra reservada “const”. Como no software anterior, uma constante se chama INICIO e tem seu valor igual a 1 (um) e a outra constante tem o nome FIM e tem seu valor igual a 1000 (mil).

Contudo, podemos observar no código abaixo que, diferentemente de quando usamos define, agora, para criar uma constante usando a palavra reservada “const”, faz-se necessário também a declaração do tipo da constante. No exemplo a seguir nossas constantes são do tipo inteiro (int).

O software abaixo apresenta no vídeo os números de 1 a 1000:

```
#include <cstdlib>
#include <iostream>

const int INICIO = 1,
          FIM = 1000;

using namespace std;

int main(int argc, char *argv[])
{
    int i;

    for (i = INICIO; i <= FIM; i++)
      cout << "\n" << i;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

2.6. Variáveis (Locais e Globais):

Uma variável pode ser declarada localmente a uma função, ou global a todas as funções de um módulo do programa.

Quando declarada dentro da função, a variável tem seu escopo limitado à própria função onde foi declarada.

No software a seguir, veremos que serão declaradas duas variáveis “A” e “B”, ambas do tipo inteiro. O objetivo deste software é simplesmente: Receber dois valores inteiros via teclado, somar os dois valores recebidos e imprimir o total da soma no monitor.

As variáveis “A” e “B”, têm seus escopos iniciados após o sinal de ponto-e-vírgula que completam suas declarações, até o final da função “main”.

Vejamos um exemplo de declaração de variáveis locais.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    int A = 0,
        B = 0;

    cout << "Digite o primeiro valor: ";
    cin >> A;
```

```
    cout << "\nDigite o segundo valor: ";
    cin >> B;

    cout << "\n\nO resultado da soma: " << A + B << "\n\n\n";

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Quando declarada fora da função, a variável tem seu escopo limitado ao módulo onde foi declarada, podendo se estender a outros módulos, desde que estes referenciem o módulo onde a variável global está declarada, através da diretiva “#include”, da mesma forma que fazemos com as bibliotecas de funções.

Se observarmos atentamente o software a seguir, veremos que ele faz exatamente o que o software anterior fazia.

Contudo, poderemos notar também, que neste novo software, não temos apenas uma única função declarada. Temos duas funções. A primeira função se chama “soma” e a segunda função se chama “main” (que é obrigatória para todos os programas construídos em Linguagem C).

Poderemos observar também que, as variáveis “A” e “B” (ambas do tipo inteiro), são declaradas antes da declaração das funções, o que as torna variáveis globais. E como veremos no exemplo a seguir, ambas as funções declaradas utilizam as mesmas variáveis globais, ou seja, “A” e “B”.

Vejamos um exemplo de declaração de variáveis globais.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int A = 0,
    B = 0;

int soma () {
  return (A + B);
}

int main(int argc, char *argv[])
{
    cout << "Digite o primeiro valor: ";
    cin >> A;

    cout << "\nDigite o segundo valor: ";
    cin >> B;

    cout << "\n\nO resultado da soma é: " << soma () << "\n\n\n";

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Note que no software acima, nem a função “soma” e tampouco a função “main” declaram variáveis locais.

IMPORTANTE:

Apesar de prático, o uso de variáveis globais não é considerado uma boa prática de programação, pois, dificulta extremamente a análise e o entendimento do software. Portanto, evite ao máximo a utilização desta prática.

Saiba que 99,9% das variáveis Globais, podem ser facilmente substituídas por variáveis locais.

Para provarmos o que acabamos de dizer, vamos refazer o software anterior eliminando as variáveis globais.

Veja no exemplo a seguir que a função “soma” não deixou de existir, contudo, agora ela possui duas variáveis inteiras “PrimeiroValor” e “SegundoValor” (que por estarem declaradas na assinatura da função levam o nome de “parâmetros”) que receberão os valores coletados pela função principal (main) através das variáveis “A” e “B”.

Em seguida a função “soma” fará a soma dos valores a ela passados e devolverá o resultado à função que a chamou (main).

Vejamos:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int soma (int PrimeiroValor, int SegundoValor) {
  return (PrimeiroValor + SegundoValor);
}

int main(int argc, char *argv[])
{
    int A = 0,
        B = 0;

    cout << "Digite o primeiro valor: ";
    cin >> A;

    cout << "\nDigite o segundo valor: ";
    cin >> B;

    cout << "\n\nO resultado da soma é: " << soma (A,B) << "\n\n\n";

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

3. Estruturas de Dados (Registros)

De acordo com Mizrahi (1990), uma estrutura de dados é um conjunto de variáveis, possivelmente de tipos diferentes, agrupadas sob um único nome. Estas estruturas também são conhecidas por “registros”.

Uma estrutura é um tipo de dado cujo formato é definido pelo programador.

Exemplos de Estruturas Simples:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
      struct endereco {
            char logradouro[50];
            int numero;
            char complemento[15];
            char bairro[30];
            char cidade[100];
            char estado[40];
            char cep[8];
      };

      struct endereco end_cliente1;
      struct endereco end_cliente2;

      strcpy (end_cliente1.logradouro, "Rua Antônio de Souza");
      end_cliente1.numero = 18;
      strcpy (end_cliente1.complemento, "Apto 102");
      strcpy (end_cliente1.bairro, "Bela Vista");
      strcpy (end_cliente1.cidade, "Belo Horizonte");
      strcpy (end_cliente1.estado, "Minas Gerais");
      strcpy (end_cliente1.cep, "30810540");

      strcpy (end_cliente2.logradouro, "Rua Alfredo Baleia");
      end_cliente2.numero = 83;
      strcpy (end_cliente2.complemento, "Apto 303");
      strcpy (end_cliente2.bairro, "Linda Vista");
      strcpy (end_cliente2.cidade, "Belo Horizonte");
      strcpy (end_cliente2.estado, "Minas Gerais");
      strcpy (end_cliente2.cep, "30830503");

      cout << "Logradouro: " << end_cliente1.logradouro;
      cout << "\nNúmero: " << end_cliente1.numero;
      cout << "\nComplemento: " << end_cliente1.complemento;
      cout << "\nBairro: " << end_cliente1.bairro;
      cout << "\nCidade: " << end_cliente1.cidade;
      cout << "\nEstado: " << end_cliente1.estado;
      cout << "\nCEP: " << end_cliente1.cep;

      cout << "\n\n";

      cout << "Logradouro: " << end_cliente2.logradouro;
```

```
	cout << "\nNúmero: " << end_cliente2.numero;
	cout << "\nComplemento: " << end_cliente2.complemento;
	cout << "\nBairro: " << end_cliente2.bairro;
	cout << "\nCidade: " << end_cliente2.cidade;
	cout << "\nEstado: " << end_cliente2.estado;
	cout << "\nCEP: " << end_cliente2.cep;

	cout << "\n\n\n";

	system("PAUSE");
	return EXIT_SUCCESS;
}
```

No código acima, nota-se claramente que, para declararmos uma estrutura ou registro em Linguagem C, utilizamos a palavra reservada “struct”, seguido do nome que queremos dar para a estrutura, este, muitas vezes chamado de “etiqueta” e que em nosso exemplo é “endereco”.

O exemplo apresentado mostra uma das formas de se declarar e utilizar uma estrutura dentro de um programa escrito em Linguagem C, contudo, o leitor poderá encontrar no dia-a-dia, códigos como este com algumas diferenças. O que não alterará o resultado final.

Veja a seguir outros exemplos de maneiras diferentes de se fazer a mesma coisa:

- No código abaixo, podemos notar que a palavra reservada “struct” não foi utilizada na declaração das variáveis “end_cliente1” e “end_cliente2”.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
	struct endereco {
		char logradouro[50];
		int numero;
		char complemento[15];
		char bairro[30];
		char cidade[100];
		char estado[40];
		char cep[8];
	};

	endereco end_cliente1, end_cliente2;

	strcpy (end_cliente1.logradouro, "Rua Antônio de Souza");
	end_cliente1.numero = 18;
	strcpy (end_cliente1.complemento, "Apto 102");

	.
	.
	.

	system("PAUSE");
	return EXIT_SUCCESS;
}
```

- No código a seguir, veremos que as variáveis “end_cliente1” e “end_cliente2”, foram declaradas antes de finalizarmos a declaração da estrutura, e ainda, podemos notar também que esta estrutura não tem “etiqueta”, ou seja, foi omitido o nome da estrutura; apenas foram criadas duas variáveis “end_cliente1” e “end_cliente2” do tipo desta estrutura.

Não há necessidade de você dar um nome para a estrutura, a não ser que você tenha que declarar, em outras partes do código, outras variáveis desta estrutura.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
      struct {
            char logradouro[50];
            int numero;
            char complemento[15];
            char bairro[30];
            char cidade[100];
            char estado[40];
            char cep[8];
      } end_cliente1, end_cliente2;

      strcpy (end_cliente1.logradouro, "Rua Antônio de Souza");
      end_cliente1.numero = 18;
      strcpy (end_cliente1.complemento, "Apto 102");

      .
      .
      .

      system("PAUSE");
      return EXIT_SUCCESS;
}
```

3.1. Inicializando Estruturas

Uma estrutura poderá ser inicializada facilmente. Basta seguir o exemplo abaixo:

```
#include <cstdlib>
#include <iostream>

using namespace std;


int main(int argc, char *argv[])
{
    struct livro {
           int codigo;
           char nome[40];
    };
```

```
    livro liv1 = { 1, "Algoritmos e Técnicas de Programação" };
    livro liv2 = { 2, "Estruturas de Dados Usando C" };

    cout << "Código: " << liv1.codigo;
    cout << "\nNome: " << liv1.nome;

    cout << "\n\n";

    cout << "Código: " << liv2.codigo;
    cout << "\nNome: " << liv2.nome;

    cout << "\n\n";

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Observe que a disposição dos valores passados para inicialização das variáveis “liv1” e “liv2”, seguem a seqüência de variáveis que compõem a estrutura “livro”, ou seja, primeiro um valor inteiro e depois um valor string.

4. Exercícios

1) Escreva um software onde seja solicitado do usuário, que entre com 30 valores reais quaisquer. Armazene estes valores num vetor. Em seguida, aplique um aumento de 20% a todos os valores digitados pelo usuário, armazenando estes novos valores num segundo vetor, na ordem inversa à qual foi digitada. Crie um terceiro vetor para receber, em suas posições, o somatório do valor contido nas respectivas posições dos dois vetores anteriores. Apresente os resultados obtidos nas posições do terceiro vetor. Faça uso de constantes para delimitar o valor máximo e mínimo dos vetores, para a aplicação do percentual de reajuste e em tudo mais que você julgar ser útil.

2) Modifique as constantes criadas no software anterior para que o programa anterior solicite do usuário, que entre com 100 valores reais quaisquer. Armazene estes valores num vetor. Em seguida, aplique um aumento de 23,7% a todos os valores digitados pelo usuário, armazenando este novo resultado num segundo vetor, na ordem inversa a qual foi digitada. Crie um terceiro vetor para receber, em suas posições, o somatório do valor contido nas respectivas posições dos dois vetores anteriores. Apresente os resultados obtidos nas posições do terceiro vetor.

   Atenção! Para resolução deste problema é extremamente importante que você só modifique as constantes criadas no exercício 1. Se você tiver que modificar algo mais além das constantes, altere a solução do exercício 1 para que ela passe a aceitar o tipo de alterações propostas neste exercício.

3) Construa um software que simule uma calculadora com as seguintes operações: Soma, subtração, divisão, multiplicação e potenciação. O usuário da calculadora deverá entrar sempre com dois valores. Em seguida o usuário deverá escolher a operação a ser efetuada nos valores que foram digitados e o programa deverá apresentar: os números digitados, a operação escolhida e o resultado da operação.

   a. Resolva o exercício 3 utilizando uma única função (main).

   b. Resolva o exercício 3 utilizando uma função para cada operação criada e também variáveis globais.

   c. Resolva o exercício 3 utilizando uma função para cada operação criada, porém, sem utilização de variáveis globais.

4) Construa um software para cadastrar livros. Os dados que deverão ser armazenados de cada livro são: Código, título do livro, nome do autor, data de publicação, edição, cidade e editora. Utilize a estrutura de dados conhecida como

"registro" para definir os dados que serão armazenados. Seu software deverá permitir que o usuário cadastre 3 livros.

5) Construa um software para cadastro de veículos. Os dados que deverão ser armazenados sobre veículos são: Marca, modelo, ano de fabricação, cor e placa. Use a estrutura conhecida como registro para compor os dados sobre o veículo. Seu software deverá ser capaz de armazenar os dados de 50 veículos. Para isto, use a estrutura registro combinada com um vetor. O software deverá permitir a entrada de quantos veículos o usuário quiser cadastrar. Crie um flag para que o usuário possa interromper o cadastro a qualquer momento (desde que não esteja no meio de um cadastro de veículo). Crie uma opção para o usuário visualizar os veículos cadastrados. Não se esqueça de usar constantes para números que se repetem (em um mesmo contexto).

6) Construa um software para cadastro de alunos. Os dados que deverão ser armazenados sobre alunos são: Matrícula, nome, idade, sexo e curso. Use a estrutura conhecida como registro para compor os dados sobre o aluno. Seu software deverá ser capaz de armazenar os dados de 100 alunos. Para isto, use a estrutura registro combinada com uma matriz 10 x 10. O software deverá permitir a entrada de quantos alunos o usuário quiser cadastrar. Crie um flag para que o usuário possa interromper o cadastro a qualquer momento (desde que não esteja no meio de um cadastro de aluno). Crie uma opção para o usuário visualizar os alunos cadastrados.

7) Crie um outro programa baseado no anterior (Exercício 6), onde se possa armazenar, além dos dados do aluno, também o seu endereço. Para isto crie uma nova estrutura chamada "endereco" que conterá os campos: Logradouro, número, bairro e cidade. Feito isto, dentro da estrutura criada para armazenar os dados do aluno crie uma nova variável chamada "ender" do tipo "endereco". O restante do programa deverá respeitar o que pede o exercício 6. Fazendo isto você estará utilizando estruturas aninhadas.

5. Ponteiros (ou Apontadores)

Segundo Mizrahi (1990), uma das características mais poderosas oferecidas pela Linguagem C, é a utilização de ponteiros. Os ponteiros são vistos pela maioria das pessoas como um dos tópicos mais difíceis de qualquer linguagem de programação.

Um ponteiro, ainda segundo Mizrahi (1990), proporciona um modo de acesso as variáveis sem referenciá-las diretamente. O mecanismo usado para isto é o endereço da variável na memória do computador. De fato, o endereço age como intermediário entre a variável e o programa que a acessa.

Mizrahi (1990) simplifica a explicação de ponteiros dizendo que: Um ponteiro pode ser entendido como sendo uma representação simbólica de um endereço da memória do computador.

5.1. Por que ponteiros são usados?

Ponteiros são usados em situações em que a passagem de valores é difícil ou indesejável. A seguir, Mizrahi (1990) lista algumas razões para o uso de ponteiros:

a) Fornecem maneiras com as quais as funções podem realmente modificar os argumentos que recebem.
b) Para passar matrizes e strings mais convenientemente de uma função para outra, isso é, usa-los ao invés de matrizes.
c) Para manipular matrizes mais facilmente através da movimentação de ponteiros para elas (ou parte delas), em vez de a própria matriz;
d) Para criar estruturas de dados complexas, como listas encadeadas e árvores binárias, onde uma estrutura de dados deve conter referências sobre outra.
e) Para comunicar informações sobre memória, como na função **malloc()** que retorna a localização de memória livre através do uso de ponteiro.
f) Uma outra razão importante para o uso de ponteiros é que notações de ponteiros compilam mais rapidamente tornando o código mais eficiente.

5.2. Por que evitar o uso de ponteiros?

O uso de ponteiros pelas linguagens de programação foi muito debatido e questionado por muitos que eram contra e a favor de seu uso.

Contudo, podemos citar alguns bons motivos para se evitar o uso de ponteiros mesmo que a linguagem de programação que você está utilizando, o provenha com recursos para utilização dos mesmos.

O uso de ponteiros:

a) Torna a programação de um código qualquer muito mais complexa.
b) Dificulta o entendimento de um código.
c) Dificulta a manutenção de um código, principalmente se não foi você quem o escreveu, pois, ao tornar o entendimento mais difícil, consequentemente, o uso de ponteiros torna a manutenção do código, também, mais difícil.

d) A eficiência que se ganha no código com o uso de ponteiros, torna-se irrelevante diante da velocidade das máquinas atuais. E também, é claro, se o código for bem escrito usando as estruturas alternativas.
e) As linguagens de programação mais recentes no mercado atualmente não provêem recursos para a utilização de ponteiros. (Ex.: “Java” e “.Net”).

5.3. Utilizando Ponteiros

Sabemos como fazer para declaramos uma variável normal, do tipo inteiro por exemplo, ou seja:

int NUM;

O código acima é uma declaração típica de uma variável de nome NUM do tipo inteiro em Linguagem C.

A declaração de ponteiros tem, segundo Mizrahi (1990), um sentido diferente da de uma variável normal, pois, a declaração:

Int *PNUM;

Declara que *PNUM é do tipo inteiro e é um ponteiro, isto é PNUM conterá endereços de variáveis do tipo inteiro, e não valores armazenados (como nas variáveis normais).

5.4. Exemplo de Utilização de Ponteiros

O software abaixo, adiciona 3 ao valor da variável X, sem a sua utilização direta.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    int X = 10;
    int *PX;

    PX = &X;

    cout << "O valor da variável X é: " << X;

    *PX = *PX + 3;

    cout << "\n\nO valor da variável X é: " << X << "\n\n";
```

```
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Uma rápida explicação sobre o software acima:

Foram criadas duas variáveis: X e PX, ambas do tipo inteiro.

Contudo, quando fizemos a seguinte declaração:

Int *PX;

Na verdade fizemos a declaração de um ponteiro para uma variável do tipo inteiro, e não uma declaração de uma variável do tipo inteiro.

Em seguida veio a declaração de atribuição de valores:

PX = &X;

Nesta declaração, não estamos passando para PX o valor armazenado em X (que até então era 10). Ao contrário, estamos passando o endereço da memória da máquina (computador) ocupado pela variável X. Ou seja, acabamos de criar um ponteiro, de nome PX, para a variável X.

Em seguida foi impresso no vídeo o valor da variável X (até aqui, 10).

Logo é feita a seguinte declaração:

***PX = *PX + 3;**

A atribuição acima parece estar adicionando ao valor de PX o valor 3. Contudo, PX não é uma variável, e sim um ponteiro, que aponta para o endereço da variável X. Neste caso, é como se estivéssemos atribuindo o valor 3 para a variável X, pois, apesar de não estarmos trabalhando diretamente com a variável X, estamos trabalhando com o valor contido na posição de memória que a variável X ocupa.

E para finalizar, ao imprimirmos novamente o valor da variável X no vídeo, veremos que este será igual a 13, pois, seu valor foi acrescido de 3 através da utilização de uma operação de adição utilizando o ponteiro PX.

5.5. Outros Exemplos de Utilização de Ponteiros

O software a seguir faz a utilização de ponteiros, através de variáveis globais, para simular uma calculadora de quatro operações. Passando os valores através de ponteiros para a respectiva função solicitada pelo usuário.

```
#include <cstdlib>
```

```
#include <iostream>

using namespace std;

int *P1Valor, *P2Valor;

int Adicao () {
    return (*P1Valor + *P2Valor);
}

int Subtracao () {
    return (*P1Valor - *P2Valor);
}

float Divisao () {
    return (*P1Valor / *P2Valor);
}

int Multiplicacao () {
    return (*P1Valor * *P2Valor);
}

int main(int argc, char *argv[])
{
    int PrimeiroValor, SegundoValor;
    int opcao;

    P1Valor = &PrimeiroValor;
    P2Valor = &SegundoValor;

    cout << "Entre com o primeiro valor: ";
    cin >> PrimeiroValor;
    cout << "\nEntre com o segundo valor: ";
    cin >> SegundoValor;

    cout << "\n\nEscolha a operação a ser efetuada com os valores
digitados:";
    cout << "\n 1 - Adição";
    cout << "\n 2 - Subtração";
    cout << "\n 3 - Divisão";
    cout << "\n 4 - Multiplicação";
    cout << "\n\n\n Opção: ";
    cin >> opcao;

    switch (opcao) {
      case 1:
           cout << "\n\nO valor da Adição é: " << Adicao ();
           break;
      case 2:
           cout << "\n\nO valor da Subtração é: " << Subtracao ();
           break;
      case 3:
           cout << "\n\nO valor da Divisão é: " << Divisao ();
           break;
      case 4:
           cout << "\n\nO valor da Multiplicação é: " << Multiplicacao ();
    }

    cout << "\n\n\n";
```

```
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

O software a seguir cria um vetor de quatro posições, preenche as quatro posições de memória e depois, através de ponteiros, faz a inversão do mesmo e imprime os valores no vídeo.

Se observarmos atentamente o código abaixo, notaremos a falta do caracter “*” (asterisco), que representa “ponteiro” em Linguagem C. Contudo é válido explicar que o nome de uma matriz, segundo Mizrahi (1990), é um ponteiro constante, ou seja, diferente dos ponteiros que criamos para acessar outras variáveis e que podem ter o endereço apontado modificado, o nome da matriz aponta sempre para um mesmo endereço, por isto o nome de ponteiro constante. Resumindo, ao criarmos uma matriz, na verdade estamos criando um ponteiro para um endereço fixo na memória (vale lembrar que é fixo em quanto a matriz existir dentro do programa), assim sendo, quando passamos uma matriz como argumento em uma função, não precisamos na função, especificar o argumento com o caracter “*”.

```
#include <cstdlib>
#include <iostream>

#define MAX 4

using namespace std;

void InverteValoresVetor (int VX[]) {
     VX [0] = 4;
     VX [1] = 3;
     VX [2] = 2;
     VX [3] = 1;
}

void ImprimeVetor (int VX[]) {
    cout << "\n\n";
    for (int P = 0; P < MAX; P++) {
        cout << "\nValor Vetor Posição: " << P << "  =  " << VX [P];
    }
    cout << "\n\n";
}

int main(int argc, char *argv[])
{
    int Vetor [MAX];

    Vetor [0] = 1;
    Vetor [1] = 2;
    Vetor [2] = 3;
    Vetor [3] = 4;

    ImprimeVetor (Vetor);

    InverteValoresVetor (Vetor);

    ImprimeVetor (Vetor);
```

```
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

A seguir temos um software que cria uma matriz 2 x 2 de números inteiros. As posições da matriz são preenchidas com os números de 1 a 4. Temos também uma função de inversão dos valores e uma de impressão da matriz. Repare que neste caso, os parâmetros das funções “InverteValoresMatriz” e “ImprimeMatriz” são ponteiros mas, sem o uso do caracter “*” (asterisco). Essa é uma particularidade dos vetores e das matrizes. Observe também que nos primeiros colchetes dos parâmetros de ambas as funções, não existe um valor fixo delimitando a quantidade de colunas da matriz, contudo, existe um valor fixo delimitando a quantidade de linhas da matriz. Essa é outra particularidade das matrizes, ou seja, da segunda dimensão em diante é necessário especificar o valor da dimensão. Para vetores, passados como parâmetros em funções, não é necessário especificar valor algum. Por terem somente uma dimensão, somente a presença dos colchetes já é o suficiente para que o compilador entenda que receberá um vetor como parâmetro.

```
#include <cstdlib>
#include <iostream>

#define MAXCOL 2
#define MAXLIN 2

using namespace std;

void InverteValoresMatriz (int MX[][MAXLIN]) {
     MX [0][0] = 4;
     MX [0][1] = 3;
     MX [1][0] = 2;
     MX [1][1] = 1;
}

void ImprimeMatriz (int MX[][MAXLIN]) {
    cout << "\n\n";
    for (int C = 0; C < MAXCOL; C++) {
        for (int L = 0; L < MAXLIN; L++) {
            cout << "\nValor Matriz Coluna: " << C << " Linha: " << L << "
=  " << MX [C][L];
        }
    }
    cout << "\n\n";
}

int main(int argc, char *argv[])
{
    int Matriz [MAXCOL][MAXLIN];

    Matriz [0][0] = 1;
    Matriz [0][1] = 2;
    Matriz [1][0] = 3;
    Matriz [1][1] = 4;

    ImprimeMatriz (Matriz);
```

```
    InverteValoresMatriz (Matriz);

    ImprimeMatriz (Matriz);

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

5.6. Exercícios

1) Construa um software em que um valor qualquer seja digitado e armazenado em uma variável inteira. Através de ponteiros para esta variável, você deverá adicionar ao valor contido na variável, seu próprio valor elevado a 5 potência.

2) Desenvolva um software que crie um vetor de 5 posições do tipo “aluno”. Onde aluno deve ser um registro contendo “matrícula”, “nome”, “curso”, “idade”. Permita que estes dados possam ser cadastrados pelo usuário do software. Construa uma função que receba o vetor criado como parâmetro e que inverta a ordem dos registros no vetor. Por final, apresente a posição e o registro nela alocado.

3) Crie um software onde se tenha uma matriz de estruturas “veículo” 3x3. Onde a estrutura veículo deve armazenar os seguintes dados: Placa, marca, modelo, cor e ano. Possibilite que o usuário do software cadastre e consulte cada veículo (ou todos simultaneamente). Crie uma função que receba essa matriz através de parâmetros e que troque os registros da matriz seguindo a seguinte ordem: Se o registro estiver na posição (0,1) da matriz, então ele deverá ser trocado com o registro que estiver na posição (1,0), e assim por diante. Desta forma, somente os registros que estiverem em posições onde tanto a linha quanto a coluna possuem o mesmo número, não serão trocados. Apresente os dados após a troca.

4) Modifique o exercício número 2 para que o mesmo passe a aceitar o cadastro de 20 alunos. Se você utilizou constantes no desenvolvimento do exercício 2, então a alteração será simples.

6. Listas, Filas e Pilhas através de Arranjos

As listas, filas e pilhas, são tipos abstratos de dados os quais nos criamos, em grande parte dos casos, para gerenciamento de estruturas de dados (registros).

Cada qual dos tipos acima citados tem sua particularidade, as quais serão descritas separadamente, a seguir:

6.1. Lista

De acordo com Ziviani (1996:35), uma das formas mais simples de interligar os elementos de um conjunto é através de uma lista. Lista é uma estrutura onde as operações inserir, retirar e localizar são definidas.

Para criarmos uma Lista, faz-se necessário a definição de um conjunto de operações sobre os objetos do tipo Lista. Estas operações dependerão de cada aplicação, não existindo um conjunto de operações que seja adequado a todas aplicações.

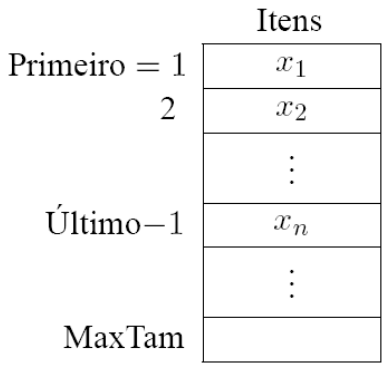

Fig. 01 – Implementação de uma lista através de arranjo
Fonte: Ziviani (1996:37)

Ziviani (1996:36) apresenta um conjunto de operações, atribuídos ao tipo Lista, encontradas na maioria das aplicações que fazem uso deste tipo. São elas:

1) Criar uma lista linear vazia.
2) Inserir um novo item imediatamente após o um determinado item.
3) Retirar um determinado item.
4) Localizar um determinado item para examinar e/ou alterar o conteúdo de seus componentes.
5) Combinar duas ou mais listas lineares em uma lista única.
6) Partir uma lista linear em duas ou mais listas.
7) Fazer uma cópia da lista linear.
8) Ordenar os itens da lista em ordem ascendente ou descendente, de acordo com alguns de seus componentes.
9) Pesquisar a ocorrência de um item com um valor particular em alguns componentes.

Para a implementação das operações citadas anteriormente, são implementadas cinco operações que resolverão a maior parte dos problemas que poderão ser encontrados nas diversas aplicações existentes. Estas operações deverão ser implementadas através das funções:

- FazListaVazia (Lista) – Faz uma lista ficar vazia.
- InsereNaLista (X, Lista) – Insere o item X após o último item da Lista.
- RetiraDaLista (P, Lista, X) – Retorna o item X que está na posição P da Lista, retirando-o da lista e deslocando os itens a partir da posição P + 1 para as posições anteriores.
- ListaVazia (Lista) – Retorna VERDADEIRO se Lista estiver vazia e FALSO caso contrário.
- ImprimeLista (Lista) – Imprime os itens da lista na ordem de ocorrência.

A seguir, um exemplo de software para inserir, excluir, alterar e imprimir uma estrutura contendo o código e o nome de pessoas, implementada em uma lista através de arranjo:

```
#include <cstdlib>
#include <iostream>
#include <conio.h>

#define InicioArranjo 0
#define MaxTam 100

using namespace std;

typedef int Apontador;

typedef struct TipoItem {
        int Codigo;
        char Nome[10];
};

typedef struct TipoLista {
        TipoItem Item [MaxTam];
        Apontador Primeiro;
        Apontador Ultimo;
};

void FazListaVazia (TipoLista &Lista) {
     Lista.Primeiro = InicioArranjo;
     Lista.Ultimo = InicioArranjo;
}

void ConsultaItem (TipoLista &Lista, TipoItem &Item, Apontador P) {
     Item.Codigo = Lista.Item [P].Codigo;
     strcpy (Item.Nome, Lista.Item [P].Nome);
}

int ListaVazia (TipoLista &Lista) {
    return (Lista.Primeiro == Lista.Ultimo);
}
```

```
void InserirNaLista (TipoItem X, TipoLista &Lista) {
     if (Lista.Ultimo > MaxTam) {
          cout << "Lista cheia!";
     }
     else {
          Lista.Item [Lista.Ultimo] = X;
          Lista.Ultimo++;
     }
}

void RetiraDaLista (Apontador P, TipoLista &Lista, TipoItem &Item) {
     if (ListaVazia (Lista) || (P >= Lista.Ultimo)) {
          cout << "Erro: Posição não existe!";
     }
     else {
          Item = Lista.Item [P];
          Lista.Ultimo--;

          for (int Aux = P; Aux < Lista.Ultimo; Aux++)
              Lista.Item [Aux] = Lista.Item [Aux + 1];
     }
}

void InserirListaPosicao (Apontador P, TipoLista &Lista, TipoItem &Item) {
     if (ListaVazia (Lista) || (P >= Lista.Ultimo)) {
          cout << "Erro: Posição não existe!";
     }
     else {
          Lista.Item[P].Codigo = Item.Codigo;
          strcpy (Lista.Item[P].Nome, Item.Nome);
     }
}

void ImprimeLista (TipoLista &Lista) {
     for (int Aux = Lista.Primeiro; Aux < Lista.Ultimo; Aux++) {
         cout << "\nCódigo: " << Lista.Item [Aux].Codigo;
         cout << "\nNome  : " << Lista.Item [Aux].Nome << "\n";
     }
}

int Menu () {
    int Opcao = 5;

    do {
       system ("CLS");
       cout << "Escolha uma opção\n";
       cout << "=================\n";
       cout << "\n 1 - Inserir ";
       cout << "\n 2 - Excluir ";
       cout << "\n 3 - Alterar ";
       cout << "\n 4 - Imprimir ";
       cout << "\n 0 - Sair\n";
       cout << "\nOpção: ";
       cin >> Opcao;
    } while ((Opcao < 0) || (Opcao > 4));

    return (Opcao);
}

int main(int argc, char *argv[])
```

```
{
    TipoLista Lista; /* Cria uma lista. */
    TipoItem Aux;    /* Auxiliar para entrada de dados. */
    int Opcao,
        Posicao;

    /* Faz a lista ficar vazia. */
    FazListaVazia (Lista);

    Opcao = Menu ();
    while (Opcao != 0) {
         switch (Opcao) {
              case 1: /* Inserir */
                   system ("CLS");
                   cout << "Inserir Item";
                   cout << "\n============\n\n";
                   cout << "\nDigite um Código: ";
                   cin >> Aux.Codigo;
                   cout << "\nDigite um Nome: ";
                   cin >> Aux.Nome;

                   InserirNaLista (Aux, Lista);
                   break;

              case 2: /* Excluir */
                   system ("CLS");
                   cout << "Excluir Item";
                   cout << "\n============\n\n";
                   cout << "\nDigite a posição a ser removida: ";
                   cin >> Posicao;

                   RetiraDaLista (Posicao, Lista, Aux);

                   cout << "\nO item a seguir foi removido da lista:\n\n";
                   cout << "\nCódigo: " << Aux.Codigo;
                   cout << "\nNome  : " << Aux.Nome;
                   break;

              case 3: /* Alterar */
                   system ("CLS");
                   cout << "Alterar Item";
                   cout << "\n============\n\n";
                   cout << "\nDigite a posição a ser alterada: ";
                   cin >> Posicao;
                   cout << "\n\n";

                   ConsultaItem (Lista, Aux, Posicao);

                   cout << "Valores atuais";
                   cout << "\n==============\n\n";
                   cout << "Código: " << Aux.Codigo;
                   cout << "\nNome  : " << Aux.Nome << "\n";

                   cout << "\nDigite o novo Código: ";
                   cin >> Aux.Codigo;
                   cout << "\nDigite o novo Nome  : ";
                   cin >> Aux.Nome;

                   InserirListaPosicao (Posicao, Lista, Aux);
                   break;
```

```
                case 4: /* Imprimir */
                     system ("CLS");
                     cout << "Itens da Lista";
                     cout << "\n==============\n\n";

                     ImprimeLista (Lista);

                     fflush(stdin);
                     getchar();
          }
          Opcao = Menu ();
     }

     system("PAUSE");
     return EXIT_SUCCESS;
}
```

6.2. Pilha

A pilha, de acordo com Tenenbaum et al. (1995:86) e Ziviani (1996:48), quando implementada em arranjos, é um conjunto ordenado de itens no qual se aplicam basicamente, duas operações para manutenção dos dados e/ou estruturas nela armazenada.

Devido às suas características, na pilha, a operação de inserção e retirada de itens devem ocorrer sempre no topo.

A pilha também é conhecida como LIFO (Last In, First Out, do inglês, Último que entra é o primeiro que sai).

Um exemplo típico para facilitar o entendimento da estrutura Pilha, é o de uma pilha de pratos.

Numa pilha de pratos, nunca retiramos o primeiro prato (de baixo para cima) e tampouco qualquer posição do meio da pilha. A retirada ou inserção pratos se dá sempre no topo da pilha.

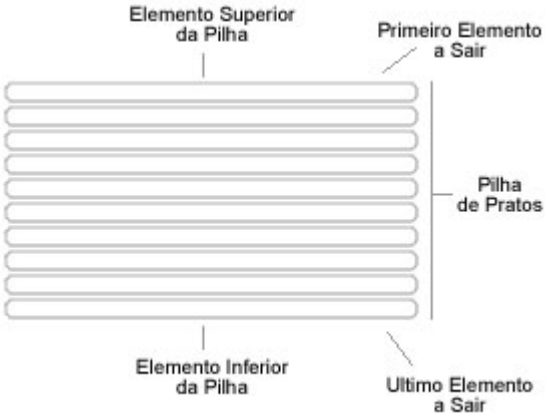

Fig. 02 – Pilha de Pratos
Fonte: <http://www.linhadecodigo.com.br/ArtigoImpressao.aspx?id=975> Acesso em: 26 nov. 2007

A seguir, foram feitas alterações no software apresentado na estrutura Lista, para que este passasse a armazenar as pessoas numa estrutura de pilha.

Poderemos observar também que as operações utilizadas na estrutura de Pilha são em número menor e bem mais simples de serem desenvolvidas do que na estrutura de Lista. Primeiramente vamos às operações realizadas sobre a estrutura de Pilha:

- FazPilhaVazia (Pilha) – Faz uma Pilha ficar vazia.
- PilhaVazia (Pilha) – Retorna VERDADEIRO se Pilha estiver vazia e FALSO caso contrário.
- Empilhar (X, Pilha) – Insere o item X no topo da Pilha.
- Desempilhar (Pilha, X) – Retorna o item X que está no topo da Pilha, removendo-o da Pilha.
- TamanhoPilha (Pilha) – Retorna quantos itens existem armazenados na Pilha.

Vejamos um exemplo do software já citado anteriormente:

```
#include <cstdlib>
#include <iostream>
#include <conio.h>

#define MaxTam 100

using namespace std;

typedef int Apontador;

typedef struct TipoItem {
        int Codigo;
        char Nome[10];
};

typedef struct TipoPilha {
        TipoItem Item [MaxTam];
        Apontador Topo;
};

void FazPilhaVazia (TipoPilha &Pilha) {
     Pilha.Topo = -1;
}

int PilhaVazia (TipoPilha &Pilha) {
    return (Pilha.Topo == -1);
}

void Empilhar (TipoItem X, TipoPilha &Pilha) {
     if (Pilha.Topo == MaxTam) {
          cout << "Pilha cheia!";
     }
     else {
          Pilha.Topo++;
          Pilha.Item [Pilha.Topo] = X;
     }
}

void Desempilhar (TipoPilha &Pilha, TipoItem &Item) {
     system("CLS");
     if (PilhaVazia (Pilha)) {
          cout << "Erro: Pilha está vazia!";
          Item.Codigo = -1;
```

```
     }
     else {
          Item = Pilha.Item [Pilha.Topo];
          Pilha.Topo--;
     }
}

int TamanhoPilha (TipoPilha &Pilha) {
    return (Pilha.Topo + 1);
}

int Menu () {
    int Opcao = 5;

    do {
       system ("CLS");
       cout << "Escolha uma opção\n";
       cout << "=================\n";
       cout << "\n 1 - Empilhar ";
       cout << "\n 2 - Desempilhar ";
       cout << "\n 3 - Verificar Tamanho da Pilha";
       cout << "\n 0 - Sair\n";
       cout << "\nOpção: ";
       cin >> Opcao;
    } while ((Opcao < 0) || (Opcao > 3));

    return (Opcao);
}

int main(int argc, char *argv[])
{
    TipoPilha Pilha; /* Cria uma Pilha. */
    TipoItem Aux;    /* Auxiliar para entrada de dados. */
    int Opcao;

    /* Faz a Pilha ficar vazia. */
    FazPilhaVazia (Pilha);

    Opcao = Menu ();
    while (Opcao != 0) {
         switch (Opcao) {
              case 1: /* Inserir */
                   system ("CLS");
                   cout << "Inserir Item";
                   cout << "\n============\n\n";
                   cout << "\nDigite um Código: ";
                   cin >> Aux.Codigo;
                   fflush(stdin);
                   cout << "\nDigite um Nome: ";
                   gets (Aux.Nome);

                   Empilhar (Aux, Pilha);
                   break;

              case 2: /* Excluir */
                   system ("CLS");
                   cout << "Excluir Item";
                   cout << "\n============\n\n";

                   Desempilhar (Pilha, Aux);
```

```
                    if (Aux.Codigo > 0) {
                       cout << "\nO item a seguir foi removido da
Pilha:\n\n";
                       cout << "\nCódigo: " << Aux.Codigo;
                       cout << "\nNome  : " << Aux.Nome;
                    }

                    fflush(stdin);
                    getchar();
                    break;

               case 3: /* Tamanho da Pilha */
                    system ("CLS");
                    cout << "Pilha";
                    cout << "\n=====\n\n";
                    cout << "Quantidade de itens na Pilha: " << TamanhoPilha
(Pilha);

                    fflush(stdin);
                    getchar();
          }
          Opcao = Menu ();
     }

     system("PAUSE");
     return EXIT_SUCCESS;
}
```

6.3. Fila

As filas, também conhecidas como listas FIFO (First In, First Out, do inglês: Primeiro que Entra, Primeiro que Sai) simulam as filas do mundo real. Como exemplo podemos citar uma fila de banco. Onde a primeira pessoa que chegar, será a primeira pessoa a ser atendida; a segunda pessoa que chegar será a segunda a ser atendida e assim por diante.

Quando manipulamos dados através de arranjos, os itens inseridos são armazenados em posições contíguas de memória. E esta estrutura deve ser utilizada quando desejamos processar dados de acordo com a ordem de chegada.

Segundo Ziviani (1996:55), um possível conjunto de operações definido para este tipo é:

- FazFilaVazia (Fila) – Faz uma fila ficar vazia.
- FilaVazia (Fila) – Retorna VERDADEIRO se fila estiver vazia e FALSO caso contrário.
- InserirNaFila (X, Fila) – Insere o item X no final da fila.
- RetirarDaFila (Fila, X) – Retorna o item X que está no início da fila, retirando-o desta.

Como a fila nada mais é do que uma lista com ordem de entrada e saída, se quisermos, podemos ainda implementar uma outra operação: Imprimir fila. Esta operação vai nos mostrar os itens que estão na fila.

- ImprimeFila (Fila) – Imprime os itens da fila na ordem de ocorrência.

Conforme veremos a seguir, uma fila na informática, ao contrário da realidade, não é representado por um conjunto de itens ordenados um após o outro em uma linha reta. Na informática, as filas são representadas por um conjunto de itens ordenados uma após o outro, porém em forma circular. Isto quando estamos tratando de implementação de Filas em arranjos, pois, desta forma não precisaremos remanejar os itens dentro do vetor à medida que entram e saem da fila. Para sabermos quem é o primeiro e o último item da fila, criaremos dois apontadores que farão este controle. Como podemos observar na figura 03, abaixo:

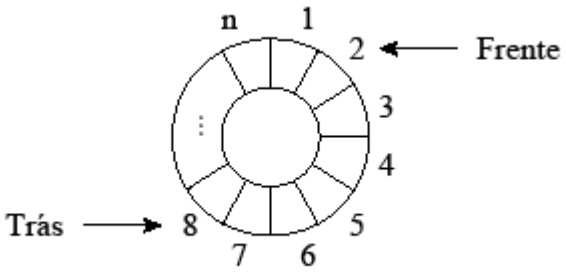

Fig. 03 – Implementação Circular para Filas
Fonte: < http://www.dcc.ufmg.br/algoritmos/transparencias.php> Acesso em: 19 nov. 2007

Por ser implementada em forma circular, de acordo com Ziviani (1996:56), existe um pequeno problema: não há uma forma de distinguir uma fila cheia de uma vazia, pois nos dois casos os apontadores Frente e Trás apontam para a mesma posição do círculo. Uma possível solução para este problema, utilizada por Aho, Hopcroft e Ullman apud Ziviani (1996:56), é não utilizar todo o espaço da fila (vetor ou array), deixando uma posição vazia. Assim sendo, a fila estará cheia quando o apontador Trás + 1 for igual ao apontador Frente. Isto significa que existirá uma célula vazia entre o fim e o início da fila.

Apesar de existir a perda de uma posição de memória, esse controle faz-se necessário para sabermos quando uma fila estará cheia ou não.

A seguir um exemplo de software que simula uma fila para atendimento de pessoas. Este software foi construído utilizando a estrutura de fila circular apresentada.

```
#include <cstdlib>
#include <iostream>
#include <conio.h>

#define MaxTam 5

using namespace std;

typedef int Apontador;

typedef struct TipoItem {
        int Codigo;
        char Nome[10];
```

```
};

typedef struct TipoFila {
        TipoItem Item [MaxTam];
        Apontador Frente;
        Apontador Tras;
};

void FazFilaVazia (TipoFila &Fila) {
     Fila.Tras = Fila.Frente = 1;
}

int FilaVazia (TipoFila &Fila) {
    return (Fila.Frente == Fila.Tras);
}

void InserirNaFila (TipoItem X, TipoFila &Fila) {
     Fila.Item [Fila.Tras - 1] = X;
     Fila.Tras = (Fila.Tras % MaxTam + 1);
}

void RetiraDaFila (TipoFila &Fila, TipoItem &Item) {
     if (FilaVazia (Fila)) {
          cout << "\n\nFila vazia!";
          getchar();
     }
     else {
          Item = Fila.Item [Fila.Frente - 1];
          Fila.Frente = (Fila.Frente % MaxTam + 1);
     }
}

int Menu () {
    int Opcao = 5;

    do {
       system ("CLS");
       cout << "Escolha uma opção\n";
       cout << "=================\n";
       cout << "\n 1 - Inserir ";
       cout << "\n 2 - Excluir ";
       cout << "\n 0 - Sair\n";
       cout << "\nOpção: ";
       cin >> Opcao;
    } while ((Opcao < 0) || (Opcao > 2));

    return (Opcao);
}

int main(int argc, char *argv[])
{
    TipoFila Fila; /* Cria uma Fila. */
    TipoItem Aux;    /* Auxiliar para entrada de dados. */
    int Opcao;

    for (int I = 0; I < MaxTam; I++) {
        Fila.Item [I].Codigo = 0;
        strcpy (Fila.Item [I].Nome, "");
    }
```

```
    /* Faz a Fila ficar vazia. */
    FazFilaVazia (Fila);

    Opcao = Menu ();
    while (Opcao != 0) {
         switch (Opcao) {
              case 1: /* Inserir */
                   system ("CLS");

                   if ((Fila.Tras % MaxTam + 1) == Fila.Frente) {
                        cout << "\n\nFila cheia!";
                        fflush(stdin);
                        getchar();
                   }
                   else {
                        cout << "Inserir Item";
                        cout << "\n============\n\n";
                        cout << "\nDigite um Código: ";
                        cin >> Aux.Codigo;
                        cout << "\nDigite um Nome: ";
                        cin >> Aux.Nome;

                        InserirNaFila (Aux, Fila);
                   }
                   break;

              case 2: /* Excluir */
                   system ("CLS");
                   cout << "Excluir Item";
                   cout << "\n============\n\n";

                   RetiraDaFila (Fila, Aux);

                   if (Aux.Codigo != -1) {
                        cout << "\nO item a seguir foi removido da
Fila:\n\n";
                        cout << "\nCódigo: " << Aux.Codigo;
                        cout << "\nNome  : " << Aux.Nome;
                   }
                   Aux.Codigo = -1;

                   fflush(stdin);
                   getchar();
         }
         Opcao = Menu ();
    }

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

6.4. Exercícios

1) Implemente um software em que o usuário digite valores aleatórios numa lista de tamanho igual a 50. Dê a opção do usuário imprimir somente os números ímpares, somente os números pares ou a lista por completo.

2) Construa um programa onde o usuário entre com 20 valores que deverão ser armazenados numa pilha. Feito isto, o software deverá apresentar os seguintes resultados: Somatório de todos os valores ímpares, somatório de todos os valores pares, somatório de todos os valores que estão nas posições ímpares da pilha, somatório de todos os valores que estão nas posições pares.

   Observação: Seu software só poderá percorrer os valores contidos na pilha através das operações de “Empilhar” e “Desempilhar”.

3) Escreva um software cuja finalidade é administrar uma agenda de atendimento bancário, onde todas as pessoas têm que marcar hora nesta agenda para que possam ser atendidas. O banco quer que seja implementado também, um mecanismo de procura para que uma mesma pessoa não possa fazer dois agendamentos. Para que possa evitar tal problema, antes de ser colocado na agenda, o software deverá pesquisar a fila existente para averiguar se já há ou não algum agendamento para a pessoa que está sendo cadastrada. Além de usar o TAD fila, use também estrutura de dados para criar um tipo agenda que contenha os dados da pessoa, do horário reservado e o sexo. Sua agenda deverá suportar no mínimo 50 pessoas.

4) Implemente um procedimento para inverter a ordem dos elementos de uma lista, usando uma pilha. Onde o usuário deverá digitar 20 valores quaisquer em uma lista, e o software deverá dar a opção de inversão e impressão dos valores na lista.

5) Em um único vetor de 20 posições, crie duas pilhas que crescem do centro para as pontas do vetor. Estas pilhas deverão ser utilizadas num software que solicitará do usuário que entre com 20 números quaisquer, limitados a 10 números impares e 10 números pares. Os números poderão ser digitados aleatoriamente. O software deverá armazenar os números pares do centro para a esquerda, e os números ímpares do centro para a direita. Use constantes para limitar os tamanhos das pilhas e do vetor, pois essas poderão ser alteradas a cada compilação. Implemente também um mecanismo de impressão dos valores digitados: Todos, só os ímpares ou só os pares.

6) Implemente um software onde o usuário entre com diversos números em uma fila, limitado a 50 posições. Implemente uma função para separar em duas outras filas, os números pares dos números ímpares. Construa também uma função para imprimir qualquer uma das duas novas filas criadas utilizando as operações, à fila, aplicadas.

7) Dadas as seguintes estruturas de dados:

```
typedef struct TipoPessoa {
        int Codigo;
        char Nome[50];
        char Logradouro[40];
        int Numero;
        char Bairro[30];
        char Cidade[50];
        char Estado[19];
```

```
        char CEP[8];
        char Telefone[20];
};

typedef struct TipoVeiculo {
        char Placa[7];
        char Marca[50];
        char Modelo[30];
        int Ano_Fabric;
        int Ano_Modelo;
};

typedef struct TipoProprietario {
        TipoPessoa Pessoa;
        TipoVeiculo Veiculo;
};
```

a) Escreva um software utilizando a estrutura de Lista implementada em Arranjo, onde seja possível cadastrar 50 pessoas. Implemente além das principais operações que são aplicadas sobre as listas, uma outra operação de Procura. Utilize-a para que em seu software não seja permitido o cadastro de duas pessoas com códigos e nomes idênticos.

b) Uma concessionária vende veículos novos e usados. Diariamente chegam veículos novos e usados para repor o estoque. Sabe-se que o pátio desta concessionária suporta no máximo 20 veículos novos e 20 veículos usados. Construa um software utilizando um único vetor de 40 posições onde, as primeiras 20 posições serão preenchidas com veículos novos, e as demais com veículos usados. Implemente além das operações básicas do TAD lista, uma operação para imprimir uma lista de veículos Novos ou Usados, à escolha do usuário.

c) Uma empresa governamental possui os direitos de operar uma balsa que faz a travessia, no canal de Bertioga, de carros que estão em Bertioga para Guarujá, no estado de São Paulo. A balsa utilizada pela empresa é estreita, sendo que os carros entram e saem dela por uma única passagem, ou seja, o primeiro carro que entrar na balsa será o último a sair. A empresa necessita manter um cadastro de todos os veículos e seus respectivos proprietários que entraram na balsa, pois a checagem de veículo e dono será feita no momento de desembarque. Esta medida visa implementar maior segurança aos proprietários de veículos nas viagens da balsa. Implemente um software que possibilite tal operação.

d) Construa um software que possua uma matriz de 10 linhas e 26 colunas. Onde cada coluna da matriz deverá funcionar como uma pilha. O software deverá solicitar o cadastro de pessoas e armazena-los de acordo com a letra inicial de cada nome, por exemplo, André, Alexandre e Antônio deverão ficar todos na coluna 0 (zero) da matriz, que está destinada a letra “A”. Bruno, Bernardo e Baltazar deverão ser empilhados na coluna 1 (um) da matriz, pois esta é a segunda coluna e está destinada à letra “B”, e assim por diante. O software deverá aceitar no máximo 10 pessoas com uma determinada letra inicial. O software deverá dar condições do usuário imprimir as colunas de

acordo com suas necessidades, ou seja, todas de uma única vez ou letra por letra. Para a manipulação dos dados na matriz, deverão ser utilizadas, unicamente, as operações aplicadas ao TAD pilha.

7. Lista, Pilha e Fila através de Ponteiros

Como já foi dito no item 6 deste trabalho, as listas, filas e pilhas, são tipos abstratos de dados os quais nos criamos, em grande parte dos casos, para gerenciamento de estruturas de dados (registros).

Cada qual dos tipos acima citados tem sua particularidade, as quais já foram descritas nos itens 6.1, 6.2 e 6.3 respectivamente. Adiante mostraremos então a implementação dos mesmos softwares apresentados nestes itens, porém, ao invés de utilizarmos arranjos, utilizaremos ponteiros ou apontadores.

Uma particularidade deste tipo de implementação é que não há um tamanho definido para Lista, Pilha ou Fila, ou seja, qualquer uma destas pode crescer indiscriminadamente. De acordo com Ziviani (1996:38), em aplicações em que não existe previsão sobre o crescimento da lista é conveniente utilizar Listas encadeadas por apontadores. E a maior desvantagem deste tipo de implementação é a utilização de memória extra para armazenar os apontadores.

7.1. Lista

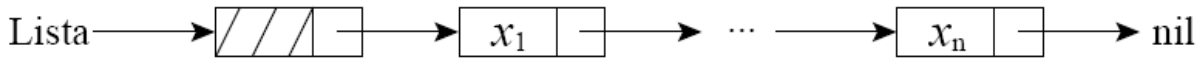

Fig. 04 – Implementação de Lista Encadeada por Apontadores
Fonte: < http://www.dcc.ufmg.br/algoritmos/transparencias.php> Acesso em: 19 nov. 2007

Para termos uma idéia mais clara do funcionamento da lista encadeada por ponteiros, vejamos a seguir a figura 05 que mostra como se procede para inserção de um novo elemento na lista. Já a figura 06 que mostra a retirada de um elemento localizado no meio da lista encadeada.

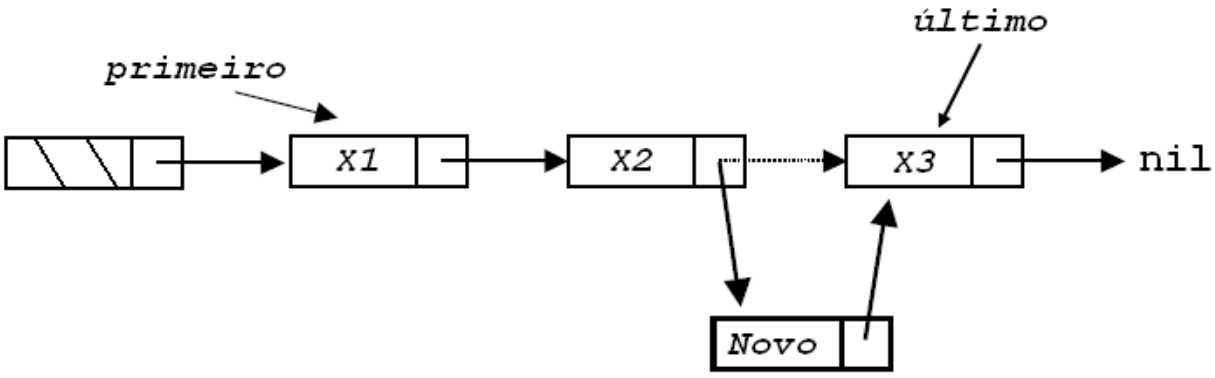

Fig. 05 – Inserção de um nodo na Lista Encadeada por Apontadores

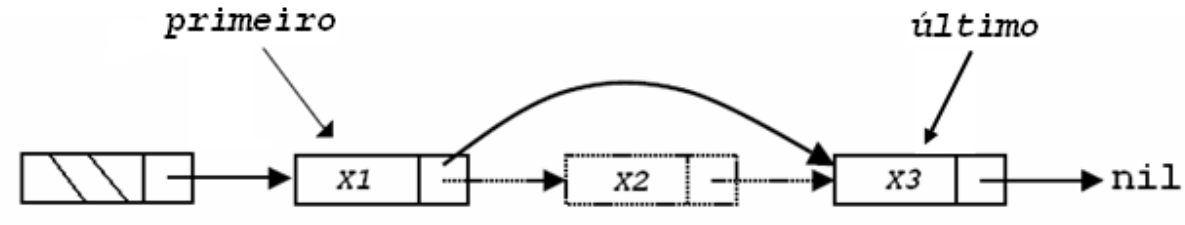

Fig. 06 – Retirada de um nodo do meio da Lista Encadeada por Apontadores

A seguir o código de um software para criação e manipulação de uma lista simples encadeada por ponteiros:

```
#include <cstdlib>
#include <iostream>
#include <conio.h>

using namespace std;

typedef struct TipoCelula *Apontador;

typedef struct TipoItem {
    int Codigo;
    char Nome[10];
};

typedef struct TipoCelula {
    TipoItem Item;
    Apontador Prox;
} Celula;

typedef struct TipoLista {
    Apontador Primeiro;
    Apontador Ultimo;
};

void FazListaVazia (TipoLista *Lista) {
    Lista->Primeiro = (Apontador) malloc (sizeof(Celula));
    Lista->Ultimo = Lista->Primeiro;
    Lista->Primeiro->Prox = NULL;
}

int ListaVazia (TipoLista Lista) {
    return (Lista.Primeiro == Lista.Ultimo);
}

void InserirNaLista (TipoItem X, TipoLista *Lista) {
     Lista->Ultimo->Prox = (Apontador) malloc(sizeof(Celula));
     Lista->Ultimo = Lista->Ultimo->Prox;
     Lista->Ultimo->Item = X;
     Lista->Ultimo->Prox = NULL;
}

void RetiraDaLista (Apontador P, TipoLista *Lista, TipoItem *Item) {
     /* O item a ser retirado é o seguinte ao apontado por P. */
     Apontador Q;
     if (ListaVazia (*Lista) || (P == NULL) || (P->Prox == NULL)) {
```

```
          cout << "Erro: Lista vazia ou posição não existe!";
     }
     else {
          Q = P->Prox;
          *Item = Q->Item;
          P->Prox = Q->Prox;

          if (P->Prox == NULL)
             Lista->Ultimo = P;

          free(Q);
     }
}

void ImprimeLista (TipoLista Lista) {
     Apontador Aux;
     int cont = 0;

     Aux = Lista.Primeiro->Prox;
     while (Aux != NULL) {
         cout << "\nPosição: " << cont;
         cout << "\nCódigo : " << Aux->Item.Codigo;
         cout << "\nNome   : " << Aux->Item.Nome << "\n";

         cont++;
         Aux = Aux->Prox;
     }
}

Apontador RetornaPonteiroParaUmaCelulaDaLista (TipoLista Lista, int
Posicao) {
     Apontador Aux;
     int cont = 0;

     Aux = Lista.Primeiro;
     while ((Aux != NULL) && (cont != Posicao)) {
         Aux = Aux->Prox;
         cont++;
     }

     return (Aux);
}

int Menu () {
    int Opcao = 5;

    do {
       system ("CLS");
       cout << "Escolha uma opção\n";
       cout << "=================\n";
       cout << "\n 1 - Inserir ";
       cout << "\n 2 - Excluir ";
       cout << "\n 3 - Alterar ";
       cout << "\n 4 - Imprimir ";
       cout << "\n 0 - Sair\n";
       cout << "\nOpção: ";
       cin >> Opcao;
    } while ((Opcao < 0) || (Opcao > 4));

    return (Opcao);
```

```
}

int main(int argc, char *argv[])
{
    TipoLista Lista;
    TipoItem Aux;    /* Auxiliar para entrada de dados. */
    Apontador P;
    int Opcao,
        Posicao = 0;

    /* Faz a lista ficar vazia. */
    FazListaVazia (&Lista);

    Opcao = Menu ();
    while (Opcao != 0) {
         switch (Opcao) {
              case 1: /* Inserir */
                   system ("CLS");
                   cout << "Inserir Item";
                   cout << "\n============\n\n";
                   cout << "\nDigite um Código: ";
                   cin >> Aux.Codigo;
                   cout << "\nDigite um Nome: ";
                   cin >> Aux.Nome;

                   InserirNaLista (Aux, &Lista);
                   break;

              case 2: /* Excluir */
                   system ("CLS");
                   cout << "Excluir Item\n";
                   cout << "============\n\n";

                   ImprimeLista (Lista);

                   cout << "\nQual registro você deseja excluir? Digite a
posição: ";
                   cin >> Posicao;

                   P = RetornaPonteiroParaUmaCelulaDaLista (Lista,
Posicao);

                   RetiraDaLista (P, &Lista, &Aux);

                   cout << "\n\nO item a seguir foi removido da
lista:\n\n";
                   cout << "\nCódigo: " << Aux.Codigo;
                   cout << "\nNome  : " << Aux.Nome;
                   fflush(stdin);
                   getchar();
                   break;

              case 3: /* Alterar */
                   system ("CLS");

                   ImprimeLista (Lista);

                   cout << "\n\nAlterar Item";
                   cout << "\n============\n\n";
                   cout << "\nDigite a posição a ser alterada: ";
```

```
                    cin >> Posicao;

                    P = RetornaPonteiroParaUmaCelulaDaLista (Lista,
Posicao);

                    cout << "\n\n";
                    cout << "Valores atuais";
                    cout << "\n==============\n\n";
                    cout << "Código: " << P->Prox->Item.Codigo;
                    cout << "\nNome  : " << P->Prox->Item.Nome << "\n";

                    cout << "\nDigite o novo Código: ";
                    cin >> P->Prox->Item.Codigo;
                    cout << "\nDigite o novo Nome  : ";
                    cin >> P->Prox->Item.Nome;
                    break;

               case 4: /* Imprimir */
                    system ("CLS");
                    cout << "Itens da Lista";
                    cout << "\n==============\n\n";

                    ImprimeLista (Lista);

                    fflush(stdin);
                    getchar();
          }
          Opcao = Menu ();
     }

     system("PAUSE");
     return EXIT_SUCCESS;
}
```

7.1.1. Lista Duplamente Encadeada

Quando falamos de listas encadeadas por ponteiros, podemos nos deparar como um pequeno problema ao implementarmos uma lista simples como mostrado na figura 04. O problema é que, de acordo com Ziviani (1996:59), na manipulação de listas lineares simples, não há como “voltar atrás” na lista, ou seja, percorre-la no sentido inverso ao dos apontadores. A solução para esta situação é a incorporação à célula de um apontador para o seu antecessor. Listas deste tipo são chamadas de **listas duplamente encadeadas**.

Vejamos a seguir a figura 07 que mostra a representação de uma lista duplamente encadeada.

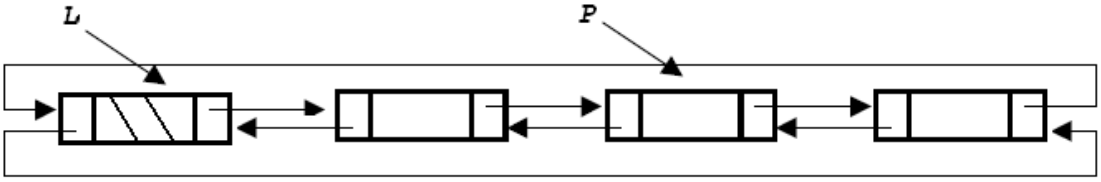

Fig. 07 – Lista Duplamente Encadeada por Apontadores

7.2. Pilha

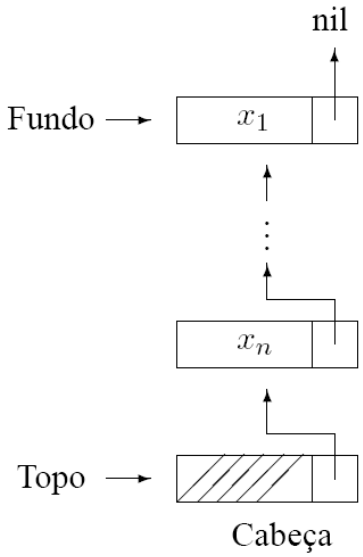

Fig. 08 – Implementação de Pilha Encadeada por Apontadores
Fonte: < http://www.dcc.ufmg.br/algoritmos/transparencias.php> Acesso em: 19 nov. 2007

Vejamos agora um exemplo de software para gerenciamento de uma pilha utilizando ponteiros ou apontadores:

```
#include <cstdlib>
#include <iostream>
#include <conio.h>

using namespace std;

typedef struct TipoCelula *Apontador;

typedef struct TipoItem {
    int Codigo;
    char Nome[10];
};

typedef struct TipoCelula {
    TipoItem Item;
    Apontador Prox;
} Celula;

typedef struct TipoPilha {
    Apontador Topo;
    Apontador Fundo;
    int Tamanho;
};

void FazPilhaVazia (TipoPilha *Pilha) {
    Pilha->Topo = (Apontador) malloc(sizeof(Celula));
    Pilha->Fundo = Pilha->Topo;
    Pilha->Topo->Prox = NULL;
    Pilha->Tamanho = 0;
}
```

```
int PilhaVazia (TipoPilha Pilha) {
    return (Pilha.Topo == Pilha.Fundo);
}

void Empilhar (TipoItem X, TipoPilha *Pilha) {
    Apontador Aux;

    Aux = (Apontador) malloc(sizeof(Celula));
    Pilha->Topo->Item = X;
    Aux->Prox = Pilha->Topo;
    Pilha->Topo = Aux;
    Pilha->Tamanho++;
}

void Desempilhar (TipoPilha *Pilha, TipoItem *Item) {
    Apontador Q;

    if (PilhaVazia (*Pilha)) {
        cout << "Erro: Pilha está vazia!";
        Item->Codigo = -1;
    }
    else {
        Q = Pilha->Topo;
        Pilha->Topo = Q->Prox;
        *Item = Q->Prox->Item;
        free (Q);
        Pilha->Tamanho--;
    }
}

int TamanhoPilha (TipoPilha Pilha) {
    return (Pilha.Tamanho);
}

int Menu () {
    int Opcao = 5;

    do {
       system ("CLS");
       cout << "Escolha uma opção\n";
       cout << "=================\n";
       cout << "\n 1 - Empilhar ";
       cout << "\n 2 - Desempilhar ";
       cout << "\n 3 - Verificar Tamanho da Pilha";
       cout << "\n 0 - Sair\n";
       cout << "\nOpção: ";
       cin >> Opcao;
    } while ((Opcao < 0) || (Opcao > 3));

    return (Opcao);
}

int main(int argc, char *argv[])
{
    TipoPilha Pilha; /* Cria uma Pilha. */
    TipoItem Aux;    /* Auxiliar para entrada de dados. */
    int Opcao;

    /* Faz a Pilha ficar vazia. */
```

```
    FazPilhaVazia (&Pilha);

    Opcao = Menu ();
    while (Opcao != 0) {
         switch (Opcao) {
              case 1: /* Inserir */
                   system ("CLS");
                   cout << "Inserir Item";
                   cout << "\n============\n\n";
                   cout << "\nDigite um Código: ";
                   cin >> Aux.Codigo;
                   fflush(stdin);
                   cout << "\nDigite um Nome: ";
                   gets (Aux.Nome);

                   Empilhar (Aux, &Pilha);
                   break;

              case 2: /* Excluir */
                   system ("CLS");
                   cout << "Excluir Item";
                   cout << "\n============\n\n";

                   Desempilhar (&Pilha, &Aux);

                   if (Aux.Codigo != -1) {
                      cout << "\nO item a seguir foi removido da
Pilha:\n\n";
                      cout << "\nCódigo: " << Aux.Codigo;
                      cout << "\nNome  : " << Aux.Nome;
                   }

                   fflush(stdin);
                   getchar();
                   break;

              case 3: /* Tamanho da Pilha */
                   system ("CLS");
                   cout << "Pilha";
                   cout << "\n=====\n\n";
                   cout << "Quantidade de itens na Pilha: " << TamanhoPilha
(Pilha);

                   fflush(stdin);
                   getchar();
         }
         Opcao = Menu ();
    }

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

7.3. Fila

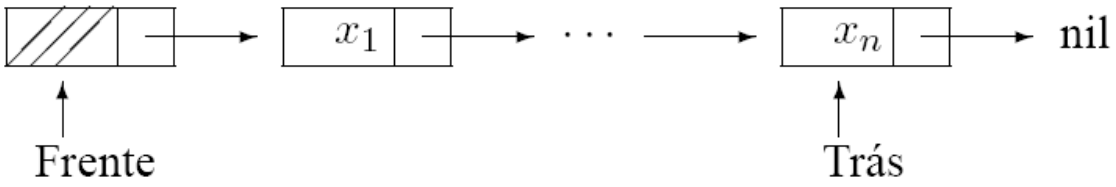

Fig. 09 – Implementação de Fila Encadeada por Apontadores
Fonte: < http://www.dcc.ufmg.br/algoritmos/transparencias.php> Acesso em: 19 nov. 2007

Abaixo o software para manipulação do conceito de fila através da utilização de apontadores.

```
#include <cstdlib>
#include <iostream>
#include <conio.h>

using namespace std;

typedef struct Celula *Apontador;

typedef struct TipoItem {
    int Codigo;
    char Nome[10];
};

typedef struct Celula {
    TipoItem Item;
    Apontador Prox;
};

typedef struct TipoFila {
    Apontador Frente;
    Apontador Tras;
};

void FazFilaVazia (TipoFila *Fila) {
    Fila->Frente = (Apontador) malloc(sizeof(Celula));
    Fila->Tras = Fila->Frente;
    Fila->Frente->Prox = NULL;
}

int FilaVazia (TipoFila Fila) {
    return (Fila.Frente == Fila.Tras);
}

void InserirNaFila (TipoItem X, TipoFila *Fila) {
     Fila->Tras->Prox = (Apontador) malloc(sizeof(Celula));
     Fila->Tras = Fila->Tras->Prox;
     Fila->Tras->Item = X;
```

```
     Fila->Tras->Prox = NULL;
}

void RetiraDaFila (TipoFila *Fila, TipoItem *Item) {
     Apontador Q;

     if (FilaVazia (*Fila)) {
          cout << "\n\nFila vazia!";
          Item->Codigo = -1;
          getchar();
     }
     else {
          Q = Fila->Frente;
          Fila->Frente = Fila->Frente->Prox;
          *Item = Fila->Frente->Item;
          free(Q);
     }
}

int Menu () {
    int Opcao = 5;

    do {
       system ("CLS");
       cout << "Escolha uma opção\n";
       cout << "=================\n";
       cout << "\n 1 - Enfileirar ";
       cout << "\n 2 - Desenfileirar ";
       cout << "\n 0 - Sair\n";
       cout << "\nOpção: ";
       cin >> Opcao;
    } while ((Opcao < 0) || (Opcao > 2));

    return (Opcao);
}

int main(int argc, char *argv[])
{
    TipoFila Fila; /* Cria uma Fila. */
    TipoItem Aux;    /* Auxiliar para entrada de dados. */
    int Opcao;

    /* Faz a Fila ficar vazia. */
    FazFilaVazia (&Fila);

    Opcao = Menu ();
    while (Opcao != 0) {
         switch (Opcao) {
              case 1: /* Inserir */
                   system ("CLS");

                   cout << "Inserir Item";
                   cout << "\n============\n\n";
                   cout << "\nDigite um Código: ";
                   cin >> Aux.Codigo;
                   fflush(stdin);
                   cout << "\nDigite um Nome: ";
                   gets (Aux.Nome);

                   InserirNaFila (Aux, &Fila);
```

```
                    break;

               case 2: /* Excluir */
                    system ("CLS");
                    cout << "Excluir Item";
                    cout << "\n============\n\n";

                    RetiraDaFila (&Fila, &Aux);

                    if (Aux.Codigo != -1) {
                         cout << "\nO item a seguir foi removido da
Fila:\n\n";
                         cout << "\nCódigo: " << Aux.Codigo;
                         cout << "\nNome  : " << Aux.Nome;
                    }
                    Aux.Codigo = -1;

                    fflush(stdin);
                    getchar();
          }
          Opcao = Menu ();
     }

     system("PAUSE");
     return EXIT_SUCCESS;
}
```

7.4. Exercício

1) Refaça os exercícios do item 6.4 utilizando Listas, Pilhas e Filas implementadas com apontadores.

8. Árvores de Pesquisa

De acordo com Ziviani (1996:111), árvores de pesquisa são estruturas de dados eficientes para armazenar informação. Uma árvore de pesquisa é particularmente adequada quando existe necessidade de considerar todos ou alguma combinação de requisitos, tais como: (i) acesso direto e seqüencial eficientes; (ii) facilidade de inserção e retirada de registros; (iii) boa taxa de utilização de memória; (iv) utilização de memória primária e secundária.

8.1. Árvores Binárias de Pesquisa Sem Balanceamento

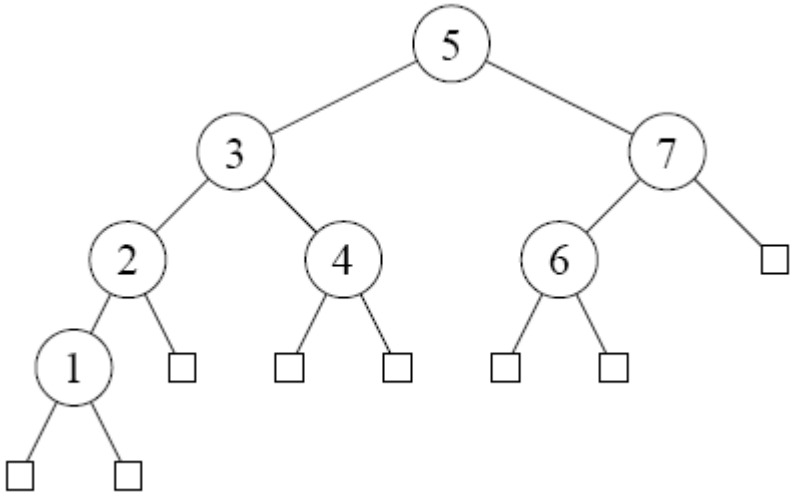

Fig. 10 – Árvore Binária de Pesquisa
Fonte: <http://www.dcc.ufmg.br/algoritmos/transparencias.php> Acesso em: 19 nov. 2007

Segundo Knuth apud Ziviani (1996:112), uma **árvore binária** é formada a partir de um conjunto finito de nodos, consistindo de um nodo chamado raiz mais 0 ou 2 subárvores binárias distintas. Resumindo, uma árvore binária é aquela onde cada nodo tem exatamente 0 ou 2 filhos. Quando um nodo tem dois filhos, eles são chamados filhos à esquerda e à direita do nodo.

Cada nodo contém apontadores para subárvores esquerda e direita. O número de subárvores de um nodo é chamado grau daquele nodo. Um nodo de grau zero é chamado de nodo externo ou folha. Os demais são chamados nodos internos.

Uma **árvore binária de pesquisa** é uma árvore binária em que todo nodo interno contém um registro, e, para cada nodo, a seguinte propriedade é verdadeira: todos os registros com chaves maiores estão na subárvore direita.

O **nível** do nodo raiz é 0 (zero). Se um nodo está no nível *i*, então a raiz de suas subárvores estão no nível *i + 1*.

A **altura** de um nodo é o comprimento do caminho mais longo deste nodo até um nodo folha. A altura de uma árvore é a altura do nodo raiz.

A estrutura de dados árvore binária de pesquisa será utilizada para implementar o **tipo abstrato de dados Dicionário**.

O TAD Dicionário possui as seguintes operações:

- Inicializa
- Pesquisa
- Insere
- Retira

Um procedimento “Pesquisa” para uma árvore binária de pesquisa é bastante simples. **Para encontrar um registro** de valor *X*, primeiro compare-o com o valor que está na raiz. Se for menor, vá para a subárvore esquerda, caso contrário vá para subárvore direita. Este procedimento repetir-se-á recursivamente até que o valor desejado seja encontrado ou então um nodo folha seja atingido. Se obtivermos sucesso na pesquisa, então o conteúdo do registro retornará o valor desejado. Se atingirmos um apontador nulo em um processo de pesquisa, significa que não encontramos o valor desejado, ou seja, este registro não está na árvore. Caso queira inseri-lo na árvore, o apontador nulo atingido é justamente o ponto de inserção.

Vejamos agora um exemplo de retirada de registro de uma árvore:

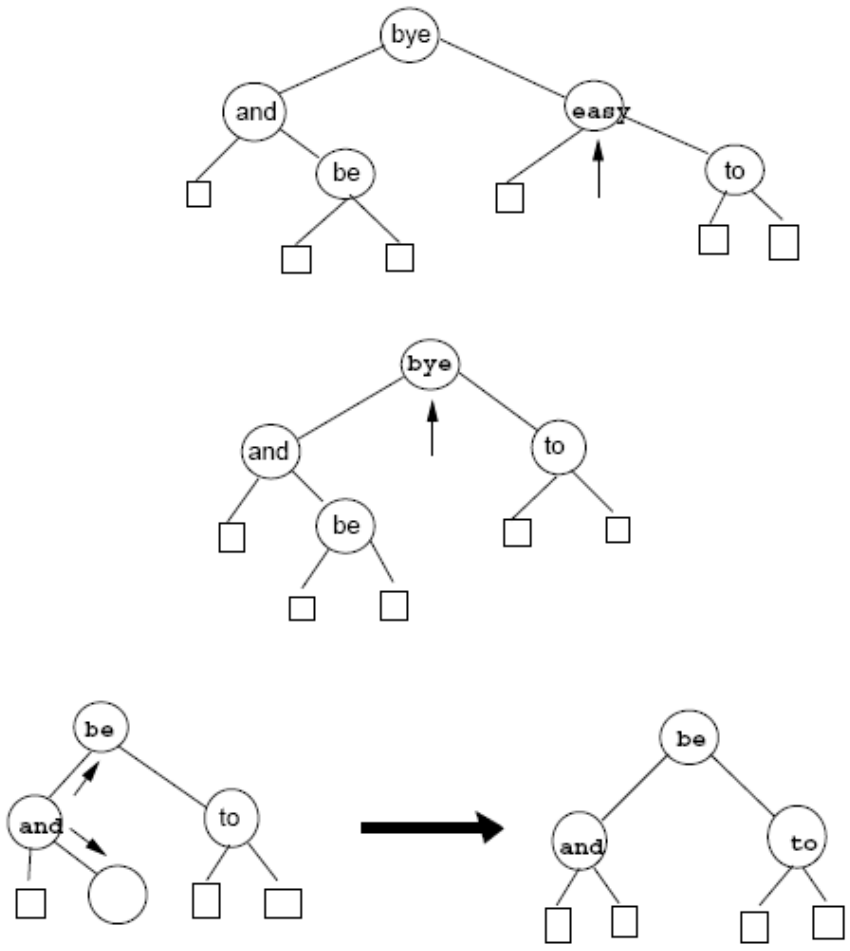

Fig. 11 – Exemplo de retirada de registro da Árvore Binária
Fonte: <http://www.dcc.ufmg.br/algoritmos/transparencias.php> Acesso em: 19 nov. 2007

A seguir, o software implementando o conceito de árvore binária de pesquisa sem balanceamento:

```
#include <cstdlib>
#include <iostream>

using namespace std;

typedef struct Nodo *Apontador;

typedef struct TipoItem {
    int Codigo;
    char Nome[50];
};

typedef struct Nodo {
    TipoItem Item;
    Apontador Esq;
    Apontador Dir;
};

typedef Apontador TipoDicionario;

void Pesquisa (TipoItem *X, Nodo **P, int &Encontrou) {
    Encontrou = 0;
    if (*P == NULL) {
        return; /* Árvore vazia. */
    }
    else {
        if (X->Codigo < (*P)->Item.Codigo) {
            Pesquisa(X, &(*P)->Esq, Encontrou);
            return;
        }
        else {
            if (X->Codigo > (*P)->Item.Codigo) {
                Pesquisa(X, &(*P)->Dir, Encontrou);
            }
            else {
                /* Retorna 1 caso o registro esteja na árvore. */
                Encontrou = 1;
                return;
            }
        }
    }
}

void Insere (TipoItem X, Apontador *P) {
    if (*P == NULL) {
        *P = (Apontador) malloc (sizeof(Nodo));
        (*P)->Item = X;
        (*P)->Esq = NULL;
        (*P)->Dir = NULL;
    }
    if (X.Codigo < (*P)->Item.Codigo) {
        Insere (X, &(*P)->Esq);
    }
    if (X.Codigo > (*P)->Item.Codigo)
        Insere (X, &(*P)->Dir);
    else
        cout << "Erro: Registro já existe na árvore!";
```

```
}

void Inicializa (TipoDicionario *Dic) {
    *Dic = NULL;
}

void Antecessor (Apontador Q, Apontador *R) {
    if ((*R)->Dir != NULL) {
        Antecessor (Q, &(*R)->Dir);
    }
    Q->Item = (*R)->Item;
    Q = *R;
    *R = (*R)->Esq;
    free(Q);
}

void Retira (TipoItem &X, Apontador *P) {
    Apontador Aux;

    if (*P == NULL) {
        cout << "Erro: Registro não está na árvore!";
        getchar();
    }
    else {
        if (X.Codigo < (*P)->Item.Codigo) {
            Retira (X, &(*P)->Esq);
        }
        else {
            if (X.Codigo > (*P)->Item.Codigo) {
                Retira (X, &(*P)->Dir);
            }
            else {
                if ((*P)->Dir == NULL) {
                    Aux = *P;
                    X = (*P)->Item;
                    *P = (*P)->Esq;
                    free(Aux);
                }
                else {
                    if ((*P)->Esq != NULL) {
                        Antecessor(*P, &(*P)->Esq);
                    }
                    else {
                        Aux = *P;
                        X = (*P)->Item;
                        *P = (*P)->Dir;
                        free(Aux);
                    }
                }
            }
        }
    }
}

void CaminhamentoCentral(Apontador P) {
    if (P != NULL) {
        CaminhamentoCentral (P->Esq);
        cout << "\nCódigo: " << P->Item.Codigo;
        cout << "\nNome  : " << P->Item.Nome << "\n";
        CaminhamentoCentral (P->Dir);
```

```
    }
}

int Menu () {
    int Opcao;

    do {
       system ("CLS");
       cout << "Escolha uma opção\n";
       cout << "=================\n";
       cout << "\n 1 - Inserir ";
       cout << "\n 2 - Excluir ";
       cout << "\n 3 - Pesquisar ";
       cout << "\n 4 - Imprimir ";
       cout << "\n 0 - Sair\n";
       cout << "\nOpção: ";
       cin >> Opcao;
    } while ((Opcao < 0) || (Opcao > 4));
    fflush(stdin);
    return (Opcao);
}

int main(int argc, char *argv[])
{
    TipoDicionario Dic; /* Cria uma Árvore. */
    TipoItem Aux;    /* Auxiliar para entrada de dados. */
    int Opcao, RegEncontrado;

    /* Faz a Fila ficar vazia. */
    Inicializa(&Dic);

    Opcao = Menu ();
    while (Opcao != 0) {
         /* Limpa Aux. */
         Aux.Codigo = 0;
         Aux.Nome[0] = '\0';

         switch (Opcao) {
              case 1: /* Inserir */
                   system ("CLS");
                   cout << "Inserir Item";
                   cout << "\n============\n\n";
                   cout << "\nDigite um Código: ";
                   cin >> Aux.Codigo;
                   fflush(stdin);
                   cout << "\nDigite um Nome: ";
                   gets(Aux.Nome);

                   Insere (Aux, &Dic);
                   break;

              case 2: /* Excluir */
                   system ("CLS");
                   cout << "Excluir Item";
                   cout << "\n============\n\n";

                   cout << "\nDigite um Código: ";
                   cin >> Aux.Codigo;
                   fflush(stdin);
```

```
                    Retira (Aux, &Dic);

                    break;

               case 3: /* Pesquisa */
                    system ("CLS");
                    cout << "Pesquisar Item";
                    cout << "\n==============\n\n";

                    cout << "\nDigite um Código: ";
                    cin >> Aux.Codigo;
                    fflush(stdin);

                    Pesquisa (&Aux, &Dic, RegEncontrado);

                    if (RegEncontrado)
                        cout << "\nRegistro encontrado.\n\n";
                    else
                        cout << "\nRegistro não encontrado.";

                    getchar();
                    break;

               case 4: /* Imprimir */
                    system ("CLS");
                    cout << "Imprimir Itens";
                    cout << "\n===============\n\n";

                    CaminhamentoCentral (Dic);

                    getchar();
          }
          Opcao = Menu ();
     }

     system("PAUSE");
     return EXIT_SUCCESS;
}
```

8.2. Árvores Binárias de Pesquisa Com Balanceamento

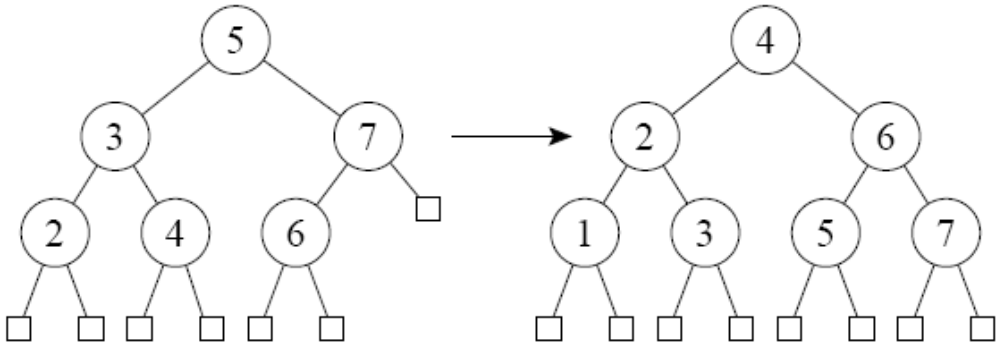

Fig. 12 – Árvore Binária de Pesquisa Completamente Balanceada
Fonte: <http://www.dcc.ufmg.br/algoritmos/transparencias.php> Acesso em: 19 nov. 2007

Para uma distribuição uniforme das chaves, segundo Ziviani (1996:117), onde cada chave é igualmente provável de ser usada em uma pesquisa, a árvore completamente balanceada[1] minimiza o tempo médio de pesquisa. Entretanto, o custo para manter a árvore completamente balanceada após cada inserção é muito alto. Por exemplo, para inserir a chave 1 na árvore à esquerda na Fig. 12 e obter a árvore à direita na mesma figura, é necessário movimentar todos os nodos da árvore original.

Uma das formas de contornar este problema é procurar uma solução intermediária que possa manter a árvore "quase balanceada", ao invés de completamente balanceada. O objetivo é procurar obter bons tempos de pesquisa, próximos do tempo ótimo da árvore completamente balanceada, mas sem pagar muito para inserir ou retirar da árvore.

Existem inúmeras heurísticas fundamentadas nos princípios acima. Gonnet e Baeza-Yates apud Ziviani (1996:118) apresentam algoritmos que utilizam vários critérios de balanceamento para árvores de pesquisa, tais como restrições impostas na diferença das alturas de subárvores de cada nodo da árvore, na redução do **comprimento do caminho interno**[2] da árvore ou que todos os nodos externos apareçam no mesmo nível.

8.3. Árvores SBB (Symmetric Binary B-trees)

De acordo com Ziviani (1996:118), as árvores B foram apresentadas por Bayer e McCreight, como uma estrutura para memória secundária.

Um caso especial da árvore B apropriado para memória primária é a **árvore 2-3**, onde cada nodo tem duas ou três subárvores.

Bayer apud Ziviani (1996:118) mostrou que as árvores 2-3 podem ser representadas por árvores binárias, conforme mostrado na figura 13, a seguir:

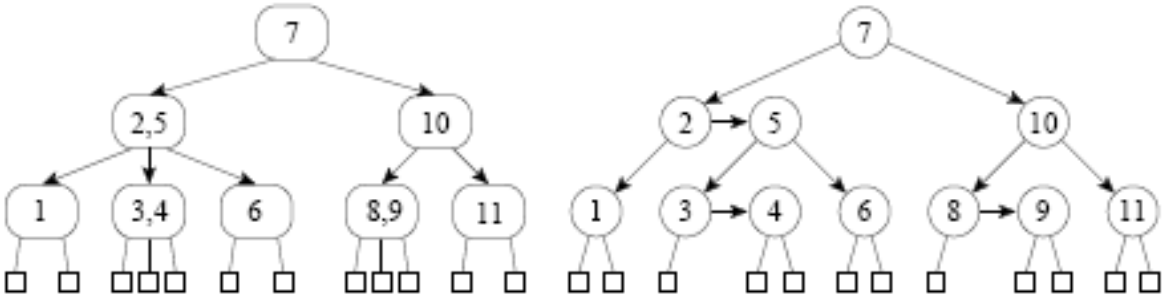

Fig. 13 – Uma Árvore 2-3 e a Árvore Binária correspondente
Fonte: <http://www.dcc.ufmg.br/algoritmos/transparencias.php> Acesso em: 19 nov. 2007

[1] Em uma árvore completamente balanceada os nodos externos aparecem em no máximo dois níveis adjacentes.
[2] O comprimento do caminho interno corresponde à soma dos comprimentos dos caminhos entre a raiz e cada um dos nodos internos da árvore. Por exemplo, o comprimento do caminho interno da árvore à esquerda na Fig. 08 é igual a 8 = (0 + 1 + 1 + 2 + 2 + 2).

A árvores SBB é uma árvores binária com dois tipos de apontadores, chamados de “verticais” e “horizontais”, onde:

- Todos os caminhos da raiz até cada nodo externo possuem o mesmo número de apontadores verticais, e
- Não podem existir dois apontadores horizontais sucessivos.

Vejamos na figura 14, a seguir, um exemplo de árvore SBB:

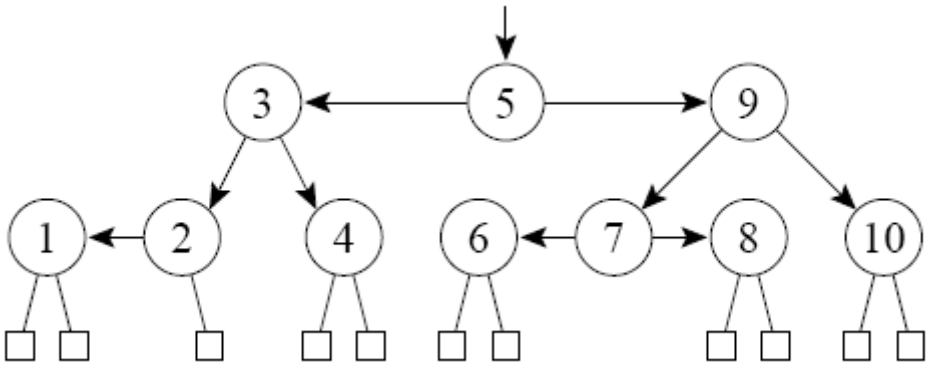

Fig. 14 – Árvore SBB
Fonte: <http://www.dcc.ufmg.br/algoritmos/transparencias.php> Acesso em: 19 nov. 2007

8.3.1. Manutenção da árvore SBB

Segundo Ziviani (1996:119), o algoritmo para árvores SBB usa transformações locais no caminho de inserção ou retirada para preservar o balanceamento. A inserção ou retirada de uma chave é sempre feita após o apontador vertical mais baixo na árvore. Dependendo da situação anterior à inserção ou retirada, podem aparecer dois apontadores horizontais sucessivos e, neste caso, faz-se necessário realizar uma transformação. Se uma transformação é realizada, a altura da subárvore transformada é um mais do que a altura da subárvore original, o que pode provocar outras transformações ao longo do caminho de pesquisa, até a raiz da árvore.

A figura 15, a seguir, mostra as transformações propostas por Bayer apud Ziviani (1996:120), onde transformações simétricas podem ocorrer.

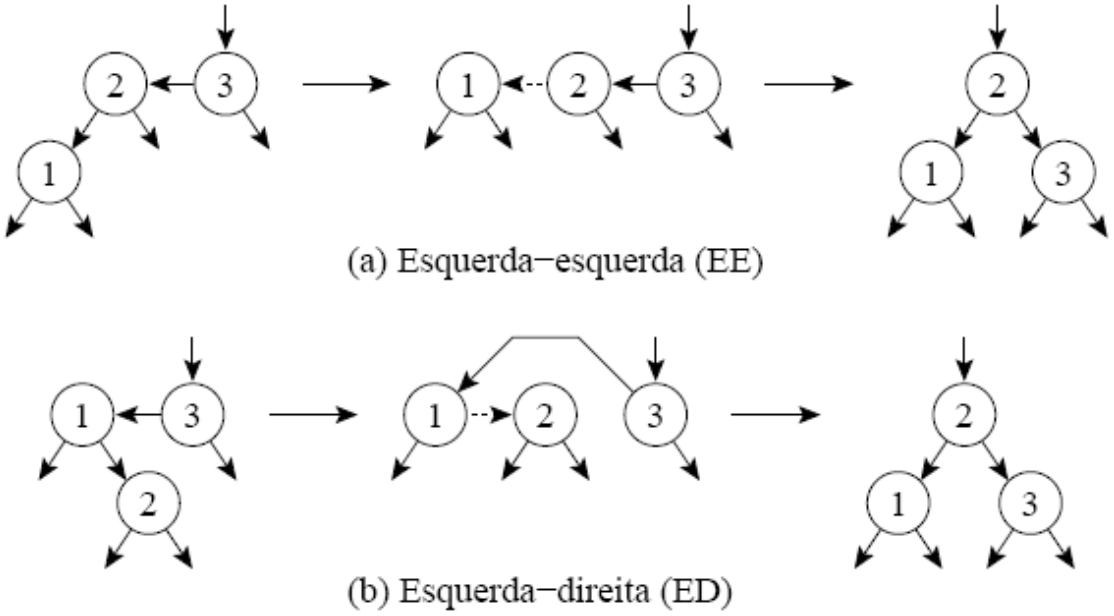

Fig. 15 – Transformações propostas por Bayer
Fonte: <http://www.dcc.ufmg.br/algoritmos/transparencias.php> Acesso em: 19 nov. 2007

Vejamos agora, um exemplo de crescimento de uma árvore SBB (figura 16), através da inserção de diversas chaves em uma árvore inicialmente vazia. Onde:

- Árvore à esquerda é obtida após a inserção das chaves 7, 10, 5.
- Árvore do meio é obtida após a inserção das chaves 2, 4 na árvore anterior.
- Árvore à direita é obtida após a inserção das chaves 9, 3, 6 na árvore anterior.

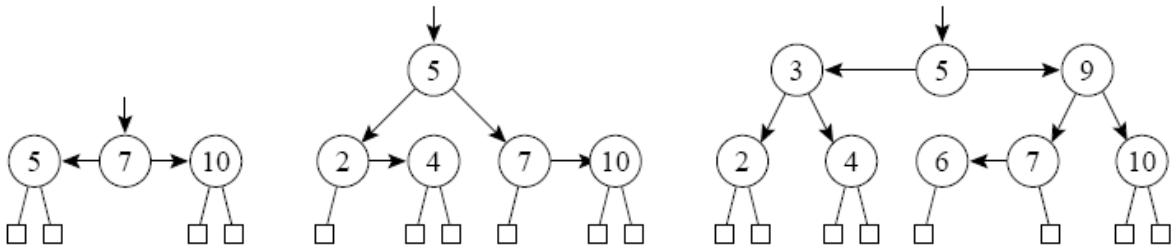

Fig. 16 – Crescimento de uma árvore SBB
Fonte: <http://www.dcc.ufmg.br/algoritmos/transparencias.php> Acesso em: 19 nov. 2007

A seguir, a figura 17 mostra a decomposição da árvore SBB mostrada na figura 16 (árvore mais a direita). Onde:

- A árvore à esquerda é obtida após a retirada da chave 7 da árvore à direita acima.
- A árvore do meio é obtida após a retirada da chave 5 da árvore anterior.
- A árvore à direita é obtida após a retirada da chave 9 da árvore anterior.

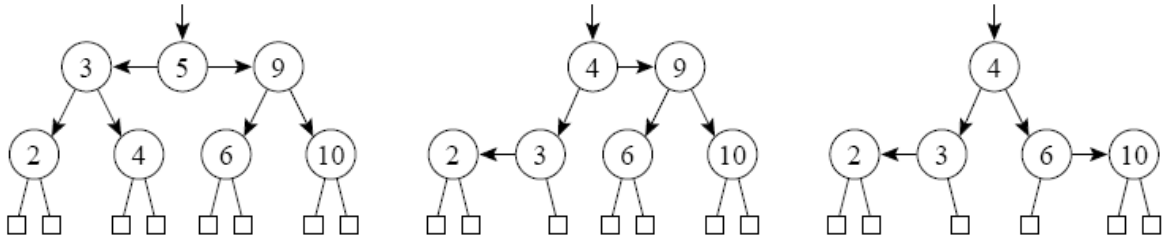

Fig. 17 – Decomposição de uma árvore SBB
Fonte: <http://www.dcc.ufmg.br/algoritmos/transparencias.php> Acesso em: 19 nov. 2007

Para as árvores SBB, faz-se necessário distinguir dois tipos de alturas. Uma delas é a altura vertical *h*, necessária para manter a altura uniforme e obtida através da contagem do número de apontadores verticais em qualquer caminho entre a raiz e um nodo externo. A outra é a altura *k*, que representa o número máximo de comparações de chaves obtido através da contagem do número total de apontadores no maior caminho entre a raiz e um nodo externo. A altura k é maior que a altura h sempre que existirem apontadores horizontais na árvore. (Ziviani, 1996, p. 126-127)

A seguir, o software implementando o conceito de árvore SBB:

```
#include <cstdlib>
#include <iostream>

using namespace std;

typedef enum { Vertical, Horizontal } Inclinacao;

typedef struct TipoItem {
    int Codigo;
    char Nome[50];
};

typedef struct Nodo_Est {
    TipoItem Item;
    Nodo_Est *Esq;
    Nodo_Est *Dir;
    Inclinacao BitE, BitD;
} Nodo;

void EE (Nodo **Ap) {
    Nodo *Ap1;
    Ap1 = (*Ap)->Esq;
    (*Ap)->Esq = Ap1->Dir;
    Ap1->Dir = *Ap;
    Ap1->BitE = Vertical;
    (*Ap)->BitE = Vertical;
    *Ap = Ap1;
}

void ED (Nodo **Ap) {
    Nodo *Ap1, *Ap2;

    Ap1 = (*Ap)->Esq;
    Ap2 = Ap1->Dir;
    Ap1->BitD = Vertical;
    (*Ap)->BitE = Vertical;
    Ap1->Dir = Ap2->Esq;
    Ap2->Esq = Ap1;
    (*Ap)->Esq = Ap2->Dir;
    Ap2->Dir = *Ap;
    *Ap = Ap2;
}

void DD (Nodo **Ap) {
    Nodo *Ap1;

    Ap1 = (*Ap)->Dir;
    (*Ap)->Dir = Ap1->Esq;
    Ap1->Esq = *Ap;
    Ap1->BitD = Vertical;
    (*Ap)->BitD = Vertical;
    *Ap = Ap1;
}

void DE (Nodo **Ap) {
    Nodo *Ap1, *Ap2;
```

```
    Ap1 = (*Ap)->Dir;
    Ap2 = Ap1->Esq;
    Ap1->BitE = Vertical;
    (*Ap)->BitD = Vertical;
    Ap1->Esq = Ap2->Dir;
    Ap2->Dir = Ap1;
    (*Ap)->Dir = Ap2->Esq;
    Ap2->Esq = *Ap;
    *Ap = Ap2;
}

void IInsere (TipoItem X, Nodo **Ap, Inclinacao *IAp, int *Fim) {
    if (*Ap == NULL) {
        *Ap = (Nodo *) malloc (sizeof(Nodo));
        *IAp = Horizontal;
        (*Ap)->Item = X;
        (*Ap)->BitE = Vertical;
        (*Ap)->BitD = Vertical;
        (*Ap)->Esq = NULL;
        (*Ap)->Dir = NULL;
        *Fim = 0;
    }
    else {
        if (X.Codigo < (*Ap)->Item.Codigo) {
            IInsere (X, &(*Ap)->Esq, &(*Ap)->BitE, Fim);
            if (!*Fim) {
                if ((*Ap)->BitE != Horizontal) {
                    *Fim = 1;
                }
                else {
                    if ((*Ap)->Esq->BitE == Horizontal) {
                        EE(Ap);
                        *IAp = Horizontal;
                    }
                    else {
                        if ((*Ap)->Esq->BitD == Horizontal) {
                            ED(Ap);
                            *IAp = Horizontal;
                        }
                    }
                }
            }
        }
        if (X.Codigo <= (*Ap)->Item.Codigo) {
            cout << "Erro : Chave já está na árvore!\n";
            *Fim = 1;
        }
        else {
            IInsere (X, &(*Ap)->Dir, &(*Ap)->BitD, Fim);
            if (!*Fim) {
                if ((*Ap)->BitD != Horizontal) {
                    *Fim = 1;
                }
                else {
                    if ((*Ap)->Dir->BitD == Horizontal) {
                        DD(Ap);
                        *IAp = Horizontal;
                    }
                    else {
                        if ((*Ap)->Dir->BitE == Horizontal) {
```

```
                            DE(Ap);
                            *IAp = Horizontal;
                        }
                    }
                }
            }
        }
    }

void Insere (TipoItem X, Nodo **Ap) {
    int Fim;
    Inclinacao IAp;

    IInsere (X, Ap, &IAp, &Fim);
}

void Inicializa (Nodo **Dic) {
    *Dic = NULL;
}

void EsqCurto (Nodo **Ap, int *Fim) {
    /* Folha esquerda retirada => árvore curta na altura esquerda. */
    Nodo *Ap1;

    if ((*Ap)->BitE == Horizontal) {
        (*Ap)->BitE = Vertical;
        *Fim = 1;
    }
    else {
        if ((*Ap)->BitD == Horizontal) {
            Ap1 = (*Ap)->Dir;
            (*Ap)->Dir = Ap1->Esq;
            Ap1->Esq = *Ap;
            *Ap = Ap1;
            if ((*Ap)->Esq->Dir->BitE == Horizontal) {
                DE(&(*Ap)->Esq);
                (*Ap)->BitE = Vertical;
            }
            else {
                if ((*Ap)->Esq->Dir->BitD == Horizontal) {
                    DD(&(*Ap)->Esq);
                    (*Ap)->BitE = Vertical;
                }
                *Fim = 1;
            }
        }
        else {
            (*Ap)->BitD = Horizontal;
            if ((*Ap)->Dir->BitE == Horizontal) {
                DE(Ap);
                *Fim = 1;
            }
            else {
                if ((*Ap)->Dir->BitD == Horizontal) {
                    DD(Ap);
                    *Fim = 1;
                }
            }
        }
```

```
    }
}

void DirCurto(Nodo **Ap, int *Fim) {
/* Folha direita retirada => árvore curta na altura direita. */
    Nodo *Ap1;

    if ((*Ap)->BitD == Horizontal) {
        (*Ap)->BitD = Vertical;
        *Fim = 1;
    }
    else {
        if ((*Ap)->BitE == Horizontal) {
            Ap1 = (*Ap)->Esq;
            (*Ap)->Esq = Ap1->Dir;
            Ap1->Dir = *Ap;
            *Ap = Ap1;
            if ((*Ap)->Dir->Esq->BitD == Horizontal) {
                ED(&(*Ap)->Dir);
                (*Ap)->BitD = Vertical;
            }
            else {
                if ((*Ap)->Dir->Esq->BitE == Horizontal) {
                    EE(&(*Ap)->Dir);
                    (*Ap)->BitD = Vertical;
                }
            }
            *Fim = 1;
        }
        else {
            (*Ap)->BitE = Horizontal;
            if ((*Ap)->Esq->BitD == Horizontal) {
                ED(Ap);
                *Fim = 1;
            }
            else {
                if ((*Ap)->Esq->BitE == Horizontal) {
                    EE(Ap);
                    *Fim = 1;
                }
            }
        }
    }
}

void Antecessor (Nodo *Q, Nodo **R, int *Fim) {
    if ((*R)->Dir != NULL) {
        Antecessor (Q, &(*R)->Dir, Fim);
        if (!*Fim)
            DirCurto(R, Fim);
    }
    else {
        Q->Item = (*R)->Item;
        Q = *R;
        *R = (*R)->Esq;
        free(Q);

        if (*R != NULL)
            *Fim = 1;
    }
```

```
}

void IRetira (TipoItem X, Nodo **Ap, int *Fim) {
    Nodo *Aux;

    if (*Ap == NULL) {
        cout << "Chave não está na árvore!\n";
        *Fim = 1;
    }
    else {
        if (X.Codigo < (*Ap)->Item.Codigo) {
            IRetira(X, &(*Ap)->Esq, Fim);
            if (!*Fim)
                EsqCurto(Ap, Fim);
        }
        else {
            if (X.Codigo > (*Ap)->Item.Codigo) {
                IRetira(X, &(*Ap)->Dir, Fim);
                if (!*Fim)
                    DirCurto(Ap, Fim);
            }
            else {
                *Fim = 0;
                Aux = *Ap;
                if (Aux->Dir == NULL) {
                    *Ap = Aux->Esq;
                    if (*Ap != NULL)
                        *Fim = 1;
                }
                else {
                    if (Aux->Esq == NULL) {
                        *Ap = Aux->Dir;
                        if (*Ap != NULL)
                            *Fim = 1;
                    }
                    else {
                        Antecessor(Aux, &Aux->Esq, Fim);
                        if (!*Fim)
                            EsqCurto(Ap, Fim);
                    }
                }
            }
        }
    }
}

void Retira(TipoItem X, Nodo **Ap) {
    int Fim;

    IRetira(X, Ap, &Fim);
}

void CaminhamentoCentral(Nodo *P) {
    if (P != NULL) {
        CaminhamentoCentral (P->Esq);
        cout << "\nCódigo: " << P->Item.Codigo;
        cout << "\nNome  : " << P->Item.Nome << "\n";
        CaminhamentoCentral (P->Dir);
    }
}
```

```
void Pesquisa (TipoItem *X, Nodo **P, int &Encontrou) {
    Encontrou = 0;
    if (*P == NULL) {
        return; /* Árvore vazia. */
    }
    else {
        if (X->Codigo < (*P)->Item.Codigo) {
            Pesquisa(X, &(*P)->Esq, Encontrou);
            return;
        }
        else {
            if (X->Codigo > (*P)->Item.Codigo) {
                Pesquisa(X, &(*P)->Dir, Encontrou);
            }
            else {
                /* Retorna 1 caso o registro esteja na árvore. */
                Encontrou = 1;
                return;
            }
        }
    }
}

int Menu () {
    int Opcao;

    do {
       system ("CLS");
       cout << "Escolha uma opção\n";
       cout << "=================\n";
       cout << "\n 1 - Inserir ";
       cout << "\n 2 - Excluir ";
       cout << "\n 3 - Pesquisar ";
       cout << "\n 4 - Imprimir ";
       cout << "\n 0 - Sair\n";
       cout << "\nOpção: ";
       cin >> Opcao;
    } while ((Opcao < 0) || (Opcao > 4));
    fflush(stdin);
    return (Opcao);
}

int main(int argc, char *argv[])
{
    Nodo *Dic; /* Cria uma Árvore. */
    TipoItem Aux;    /* Auxiliar para entrada de dados. */
    int Opcao,
        RegEncontrado;

    /* Faz a Fila ficar vazia. */
    Inicializa(&Dic);

    Opcao = Menu ();
    while (Opcao != 0) {
         /* Limpa Aux. */
         Aux.Codigo = 0;
         Aux.Nome[0] = '\0';

         switch (Opcao) {
```

```
                case 1: /* Inserir */
                     system ("CLS");
                     cout << "Inserir Item";
                     cout << "\n============\n\n";
                     cout << "\nDigite um Código: ";
                     cin >> Aux.Codigo;
                     fflush(stdin);
                     cout << "\nDigite um Nome: ";
                     gets(Aux.Nome);

                     Insere (Aux, &Dic);
                     break;

                case 2: /* Excluir */
                     system ("CLS");
                     cout << "Excluir Item";
                     cout << "\n============\n\n";

                     cout << "\nDigite um Código: ";
                     cin >> Aux.Codigo;
                     fflush(stdin);

                     Retira (Aux, &Dic);
                     break;

                case 3: /* Pesquisa */
                     system ("CLS");
                     cout << "Pesquisar Item";
                     cout << "\n==============\n\n";

                     cout << "\nDigite um Código: ";
                     cin >> Aux.Codigo;
                     fflush(stdin);

                     Pesquisa (&Aux, &Dic, RegEncontrado);

                     if (RegEncontrado)
                         cout << "\nRegistro encontrado.\n\n";
                     else
                         cout << "\nRegistro não encontrado.";

                     getchar();
                     break;

                case 4: /* Imprimir */
                     system ("CLS");
                     cout << "Imprimir Itens";
                     cout << "\n===============\n\n";

                     CaminhamentoCentral (Dic);

                     getchar();
           }
           Opcao = Menu ();
     }

     system("PAUSE");
     return EXIT_SUCCESS;
}
```

8.4. Exercício

1) Se copiarmos os códigos apresentados para o compilador, veremos que funcionam perfeitamente. Contudo, não se pode provar, através do simples funcionamento do software, que uma pesquisa numa árvore SBB pode ser mais eficiente que em uma árvore Binária sem balanceamento (levando-se em consideração que ambas terão as mesmas chaves). Assim sendo, pede-se: Crie um software, onde se faça a entrada de chaves aleatórias idênticas, numa árvore binária sem balanceamento e, em seguida, esta mesma chave em uma árvore SBB. Certifique-se de entrar com um número considerável de chaves, para que se possa medir o tempo de pesquisa das chaves. Implemente um mecanismo em seu software, em que se possa solicitar a pesquisa de existência de uma determinada chave; esta pesquisa deverá ser feita em ambas as árvores, e o tempo de resposta deverá ser medido e mostrado como resultado para que se possa comparar a eficiência de uma árvore e de ou outra.

9. Bibliografia

FUNDAMENTOS DE LINGUAGEM C: Tudo que você precisa saber sobre C para não passar vergonha!, Centro Tecnológico de Mecatrônica. Caxias do Sul, RS, 1997

MIZRAHI, Victorine Viviane. **Treinamento em Linguagem C :** Curso Completo – Módulo 1, São Paulo:McGraw-Hill, 1990

MIZRAHI, Victorine Viviane. **Treinamento em Linguagem C :** Curso Completo – Módulo 2, São Paulo:McGraw-Hill, 1990

SANTOS, Wellington Lima dos. **Algoritmos e Estruturas de Dados II** : Exercícios Listas Lineares. UFGD/FACET – Disponível em: < http://www.ufgd.edu.br/~wlsantos/Algo/ListasLineares_Exercicios.pdf> Acesso em: 27 nov. 2007

TENENBAUM, M. Aaron; LANGSAM, Yedidyah; AUGENSTEIN, Moshe J. **Estrutura de Dados Usando C**. São Paulo: Pearson, 1995

ZIVIANI, Nivio. **Projeto de Algoritmos** : com implementações em Pascal e C. 3ª ed., São Paulo: Pioneira, 1996