Roteiro 7 - Subrotinas

1. Introdução

A complexidade dos algoritmos está intimamente ligada à da aplicação a que se destinam. Em geral, problemas complicados exigem algoritmos extensos para sua solução.

Sempre é possível dividir problemas grandes e complicados em problemas menores e de solução mais simples. Assim, pode-se solucionar cada um destes pequenos problemas separadamente, criando algoritmos para tal (subrotinas ou subalgoritmos). Posteriormente, pela justaposição destes subalgoritmos elabora-se "automaticamente" um algoritmo mais complexo e que soluciona o problema original.

Um **subalgoritmo** é um nome dado a um trecho de um algoritmo mais complexo e que, em geral, encerra em si próprio um pedaço da solução de um problema maior – o algoritmo a que ele está subordinado. Conceito muito presente na área de Engenharia de Software.

Vantagens da utilização de subprogramas:

- Economia de código: escreve-se menos;
- Desenvolvimento modularizado: pensa-se no algoritmo por partes;
- Facilidade de depuração (correção/acompanhamento): é mais fácil corrigir/detectar um erro apenas uma vez do que dez vezes;
- Facilidade de alteração do código: se é preciso alterar, altera-se apenas uma vez;
- Generalidade de código com o uso de parâmetros: escreve-se algoritmos para situações genéricas.

2. Funcionamento

Um algoritmo completo é dividido em um algoritmo principal (**main**) e diversas sub-rotinas (tantas quanto necessárias), sendo que o algoritmo principal é por onde sempre se inicia a execução. Esse algoritmo principal pode eventualmente invocar (chamar) as demais subrotinas implementadas no mesmo.

Durante a execução do algoritmo principal, quando se encontra um comando de invocação de um subalgoritmo, a execução do programa principal é interrompida. A seguir, passa-se à execução dos comandos do corpo do subalgoritmo. Ao seu término, retoma-se a execução do algoritmo que o chamou (no caso, o algoritmo principal) no ponto onde foi interrompida (comando de chamada do subalgoritmo) e prossegue-se pela instrução imediatamente seguinte. É importante ressaltar, também, que é possível que um subalgoritmo chame outro através do mesmo mecanismo.

3. Definição de Subrotinas

A definição de uma subrotina requer:

- **Cabeçalho**: define-se o **nome** e o **tipo** do subalgoritmo, bem como os seus **parâmetros** requisitados;

- **nome:** é o identificador da subrotina, ou seja, é o nome como a mesma é invocada por outros programas;
- **parâmetros:** são os dados requisitados para que a subrotina realize a tarefa, e/ou os dados de retorno da subrotina ao programa "chamador". Tais dados são necessários para execução da tarefa. Essa comunicação pode ser feita de duas maneiras (estudados mais à frente): **variáveis globais** ou **passagem de parâmetros;**
- **tipo**: definido pela existência ou não de um retorno ao programa que o chamou. Segundo essa classificação as sub-rotinas podem ser de dois tipos:
  - **Procedimentos:** não possuem retorno para o programa "chamador", por isso são do tipo **void**;
  - **Funções:** possui um, e somente um, valor de retorno para o programa "chamador", sendo que esse retorno é de um determinado tipo primitivo (**int**, **float**, **double**, etc);

- **Corpo do programa:** instruções realizadas pela subrotina;

4. Variáveis Globais e Variáveis Locais

Variáveis globais são aquelas declaradas no início de um algoritmo, antes mesmo do programa principal. Estas variáveis são **visíveis** (isto é, podem ser usadas) no algoritmo principal e por todas as subrotinas.

Variáveis locais são aquelas definidas dentro de um programa (subrotina ou programa principal) e, portanto, somente visíveis utilizáveis dentro do mesmo. Outros subalgoritmos, ou mesmo o algoritmo principal, não podem utilizá-las.

5. Mecanismos de passagem de parâmetros

A passagem de parâmetros formaliza a comunicação entre os módulos. Além disto, permite que um módulo seja utilizado com operandos diferentes, dependendo do que se deseja do mesmo.

Dá-se a designação de **parâmetro real ou de chamada** ao objeto utilizado na unidade chamadora/ativadora e de **parâmetro formal ou de definição** ao recebido como parâmetro na subrotina.

A passagem de parâmetros pode ser realizada segundo dois mecanismos: passagem por valor (ou **cópia**); passagem por referência.

IMPORTANTE: A linguagem c++ não permite que um vetor ou uma matriz sejam passados na íntegra para uma subrotina. Para resolver isso, deve-se passar apenas o endereço de início da posição inicial do vetor ou da matriz. Esse endereço é obtido utilizando o nome do vetor ou matriz sem a utilização do índice entre colchetes.

5.1. Passagem por valor (ou cópia)

Na passagem de parâmetros por valor (ou por cópia) o parâmetro real é calculado e uma cópia de seu valor é fornecida ao parâmetro formal, no ato da invocação da subrotina. A execução da subrotina prossegue normalmente e todas as modificações feitas no parâmetro formal não afetam o parâmetro real, pois trabalha-se apenas com uma cópia do mesmo.

5.2. Passagem por referência

Neste mecanismo de passagem de parâmetros não é feita uma reserva de espaço em memória para os parâmetros formais como na passagem por valor. Quando uma subrotina com parâmetros passados por referência é chamada, o espaço de memória ocupado pelos parâmetros reais é compartilhado pelos parâmetros formais correspondentes. Assim, as eventuais modificações feitas nos parâmetros formais também afetam os parâmetros reais correspondentes.

Um mesmo subalgoritmo pode utilizar diferentes mecanismos de passagem de parâmetros, para parâmetros distintos. Para diferenciar uns dos outros, em c++, coloca-se o prefixo & (e comercial) antes da definição dos parâmetros formais passados por referência.

6. Exemplos

6.1. Procedimentos

6.1.1. Procedimento com parâmetros passados por valor

Exemplo de um procedimento que calcula a média de dois valores reais, tais valores são passados **por valor** à subrotina, ou seja, os valores dos parâmetros reais não são alterados na execução da subrotina.

```
1   #include <iostream>
2
3   using namespace std;
4
5   // cabecalho do procedimento requisitando duas variaveis do tipo real (a e b)
6   void media(float a, float b);
7
8   int main()
9   {
10      float n1, n2;
11      cout<<"\n Digite os valores do numeros: "<<endl;
12      cin>>n1>>n2;
13      //chamada do procedimento passando as variaveis requisitadas (n1 e n2)
14      media(n1, n2);
15      cout<<"\n O valor de N1 "<<n1<<endl;
16      cout<<"\n O valor de N2 "<<n2<<endl;
17      return 0;
18  }
19
20  void media(float i, float j) //implementacao do procedimento
21  {
22      float media;
23      media = (i+j)/2;
24      i = 50;
25      j = 40;
26      cout<<"\n A media dos numeros e: "<<media<<endl;
27  }
```

Exemplo – Procedimento com parâmetros passados por valor

Nesse exemplo, inicialmente tem-se a definição do cabeçalho do procedimento **media** (linha 6), requisitando como parâmetro duas variáveis do tipo real (a e b) que são **passados por valor**; note que tal procedimento é do tipo **void**, pois não possui valor de retorno; o procedimento é chamado pelo algoritmo principal (linha 14) passando como parâmetros as variáveis n1 e n2; na implementação do procedimento (linha 20 à 27) tem-se a definição da variável **media** a qual recebe os cálculos necessários, sendo o resultado impresso ao final. Note, ao executar o programa, que, os parâmetros passados por valor são alterados na subrotina (linhas 24 e 25), mas os valores impressos no programa principal são os mesmos digitados inicialmente. Além disso, repare que os nomes da variáveis na definição do cabeçalho, na chamada do procedimento e na implementação do procedimento não precisam ser os mesmos.

6.1.2. Procedimento com definição de variável global

Exemplo de um procedimento semelhante ao anterior, porém a variável que armazena a média é definida como variável global do programa (linha 9).

```
1   #include <iostream>
2
3   using namespace std;
4
5   // cabecalho do procedimento requisitando duas variaveis do tipo real (a e b)
6   void media(float a, float b);
7
8   //Definicao da variavel global
9   float media1;
10
11  int main()
12  {
13      float n1, n2;
14      cout<<"\n Digite os valores do numeros: "<<endl;
15      cin>>n1>>n2;
16      //chamada do procedimento passando as variaveis requisitadas (n1 e n2)
17      media(n1, n2);
18      cout<<"\n A media dos numeros e: "<<media1<<endl;
19      return 0;
20  }
21
22  void media(float i, float j) //implementacao do procedimento
23  {
24      media1 = (i+j)/2;
25  }
```

Exemplo – Procedimento com parâmetros passados por valor e variável global

Nesse exemplo, inicialmente tem-se a definição do cabeçalho do procedimento **media** (linha 6), requisitando como parâmetro duas variáveis do tipo real (a e b) que são **passados por valor**; note que tal procedimento é do tipo **void**, pois não possui valor de retorno; na linha 9 define-se a variável global **media1** do tipo real que armazenará o valor dos cálculos. O procedimento é chamado pelo algoritmo principal (linha 17) passando como parâmetros as variáveis n1 e n2; na implementação do procedimento (linha 22 à 25) tem-

se somente os cálculos necessários armazenados na variável global **media1**. Repare que, nesse caso, o **nome da variável global** deve ser o mesmo em todos as subrotinas.

6.1.3. Procedimento com parâmetros passados por referência

Exemplo de um procedimento semelhante ao anterior, porém os valores dos parâmetros são passados **por referência** à subrotina, ou seja, os valores dos parâmetros reais podem ser alterados na execução da subrotina.

```
1   #include <iostream>
2
3   using namespace std;
4
5   // cabecalho do procedimento requisitando duas variaveis do tipo real (a e b)
6   void media(float &a, float &b, float &media1);
7
8   int main()
9   {
10      float n1, n2, media1;
11      cout<<"\n Digite os valores do numeros: "<<endl;
12      cin>>n1>>n2;
13      //chamada do procedimento passando as variaveis requisitadas (n1 e n2)
14      media(n1, n2, media1);
15      cout<<"\n O valor de N1 "<<n1<<endl;
16      cout<<"\n O valor de N2 "<<n2<<endl;
17      return 0;
18  }
19
20  void media(float &i, float &j, float &m) //implementacao do procedimento
21  {
22      m = (i+j)/2;
23      i = 50;
24      j = 40;
25      cout<<"\n A media dos numeros e: "<<m<<endl;
26  }
```

Exemplo – Procedimento com parâmetros passados por referência

Nesse exemplo, inicialmente tem-se a definição do cabeçalho do procedimento **media** (linha 6), requisitando como parâmetro três variáveis do tipo real (a, b e media1) que são **passados por referência**; note que tal procedimento é do tipo **void**, pois não possui valor de retorno. O procedimento é chamado pelo algoritmo principal (linha 14) passando como parâmetros as variáveis n1, n2 e media1; na implementação do procedimento (linha 20 à 26) tem-se os cálculos necessários armazenados na variável **media1**. Note, ao executar o programa, que, os parâmetros passados por valor são alterados na subrotina (linhas 23 e 24). Ao imprimir os valores de n1 e n2 (linhas 15 e 16) os valores são diferentes dos digitados inicialmente. Os valores dos parâmetros reais podem ser alterados, pois os parâmetros foram passados por referência.

6.2. Funções

6.2.1. Função com parâmetros passados por valor

Exemplo de uma função que calcula a média de dois valores reais, tais valores são passados **por valor** à subrotina, ou seja, os valores dos parâmetros reais não são alterados na execução da subrotina.

```
1   #include <iostream>
2
3   using namespace std;
4
5   // cabecalho da função requisitando duas variaveis do tipo real (a e b)
6   float media(float a, float b);
7
8   int main()
9   {
10      float n1, n2, media1;
11      cout<<"\n Digite os valores do numeros: "<<endl;
12      cin>>n1>>n2;
13      //chamada da função passando as variaveis requisitadas (n1 e n2)
14      media1 = media(n1, n2);
15      cout<<"\n A media dos numeros e: "<<media1<<endl;
16      return 0;
17  }
18
19  float media(float i, float j) //implementacao da função
20  {
21      return (i+j)/2;
22  }
```

Exemplo – Função com parâmetros passados por valor

Nesse exemplo, inicialmente tem-se a definição do cabeçalho da função **media** (linha 6), requisitando como parâmetro duas variáveis do tipo real (a e b) que são **passados por valor**; note que tal função é do tipo **float**, pois possui valor real como retorno; a função é chamada pelo algoritmo principal (linha 14) passando como parâmetros as variáveis n1 e n2 e atribuindo o valor de retorno à variável média1. Na implementação da função (linha 19 à 22) tem-se a os cálculos necessários que são retornados.

As demais questões (variável global, passagem por referência) abordadas nos exemplos das seções sobre procedimentos são similares quando se trata de funções. Ressaltando que a diferença é que as funções possuem valor de retorno e os procedimentos não.

6.3. Passagem de Vetores e Matrizes por parâmetro

Matrizes e vetores são um caso especial e exceção à regra em que os parâmetros são passados sempre por referência. Ou seja, qualquer alteração em seus elementos altera a variável usada como parâmetro na chamada da rotina.

Existem basicamente três maneiras de declarar um vetor como um parâmetro de uma função.

Primeira: passa-se o vetor conforme declarado. O exemplo abaixo mostra um programa que usa uma função para carregar um vetor. Observe que a dimensão do vetor foi declarada explicitamente.

```
#include <iostream>

#define tam 10

using namespace std;

void carrega_vetor(float v[tam]);

int main()

{

   float vetor[tam];

   carrega_vetor(vetor);

   return 0;

}

void carrega_vetor(float v[tam])

{

   for(int i=0; i<tam; i++)

   {

      cout<<"\n Digite a posicao "<<i+1<<" do vetor: "<<endl;

      cin>>v[i];

   }

}
```

==========Exemplo Usando matriz==========

```
#include <iostream>

#define tam 10

using namespace std;

void carrega_vetor(float v[tam][tam]);
```

```
int main()
{
   float matriz[tam][tam];
   carrega_vetor(matriz);
   return 0;
}
void carrega_vetor(float v[tam][tam])
{
   for(int i=0; i<tam; i++)
   {
      for(int j=0; j<tam; j++)
      {
         cout<<"\n Digite a posicao da linha "<<i+1<<" coluna "<<j+1<<" da matriz: "<<endl;
         cin>>v[i][j];
      }
   }
}
```

Outra maneira leva em conta que apenas o endereço do vetor é passado. Neste modo o parâmetro é passado como um vetor “sem dimensão”. A função somente precisa receber o endereço onde se encontra o vetor. O código abaixo exemplifica tal processo.

```
#include <iostream>
#define tam 10
using namespace std;
void carrega_vetor(float v[ ]);
```

```
int main()
{
   float vetor[tam];
   carrega_vetor(vetor);
   return 0;
}
void carrega_vetor(float v[ ])
{
   for(int i=0; i<tam; i++)
   {
      cout<<"\n Digite a posicao "<<i+1<<" do vetor: "<<endl;
      cin>>v[i];
   }
}
```

==========Exemplo Usando matriz==========

```
#include <iostream>
#define tam 10
using namespace std;
void carrega_vetor(float v[][tam]);
int main()
{
   float matriz[tam][tam];
   carrega_vetor(matriz);
   return 0;
}
```

```
void carrega_vetor(float v[][tam])
{
   for(int i=0; i<tam; i++)
   {
      for(int j=0; j<tam; j++)
      {
         cout<<"\n Digite a posicao da linha "<<i+1<<" coluna "<<j+1<<" da matriz: "<<endl;
         cin>>v[i][j];
      }
   }
}
```

Note que, para matriz não é necessário colocar o tamanho somente na primeira dimensão. Para as demais dimensões devem ser colocados os tamanhos respectivos. Por exemplo, se tivéssemos uma matriz de 3 dimensões a passagem do parâmetros poderia ser m[tam][tam][tam], ou m[ ][tam][tam].

A terceira maneira implica no uso de ponteiros, o que será visto no próximo semestre.

7. Recursividade

A recursividade é uma característica que alguns problemas apresentam: a de serem definidos em termos deles mesmos. Todo problema que se comporta assim é dito ser recursivo.

A recursão é uma técnica apropriada se o problema a ser resolvido tem as seguintes características:

1 - A resolução dos casos maiores do problema envolve a resolução de um ou mais casos menores;

2 - Os menores casos possíveis do problema podem ser resolvidos diretamente;

3 - A solução iterativa do problema (usando enquanto, para ou repita) é complexa.

Exemplo: O problema do fatorial é recursivo por definição: N! = N x (N - 1) x (N - 2) x (N - 3) ... x 1

A partir disso, pode-se concluir que o fatorial de N está expresso em termos do fatorial de N-1.

$$N! = N \times \underbrace{(N - 1) \times (N - 2) \times (N - 3) \times \ldots \times 1}_{(N - 1)!}$$

Resumindo:

$$N! = N \times (N - 1)! \qquad \text{(Equação 2)}$$

A equação 2 é válida para todos os números inteiros com exceção do 0 (zero), sendo, portanto, necessário um tratamento especial.

```
1   #include <iostream>
2
3   int fat(int n);
4
5   using namespace std;
6
7   int main()
8   {
9       int n, fatorial;
10      cout<<"\n Digite o numero a ser calculado o fatorial: "<<endl;
11      cin>>n;
12      fatorial = fat(n);
13      cout<<"\n O fatorial de "<<n<<" e "<<fatorial;
14      return 0;
15  }
16
17  int fat(int n)
18  {
19      if(n==0){
20          return 1;
21      }
22      else{
23          return n*fat((n-1));
24      }
25  }
26
```

Exemplo - Fatorial calculado de forma recursiva

Note que a função invoca ela mesmo em sua execução (linha 23) até que chegue a um critério de parada (n = 0).

Exercícios

1. Escreva um procedimento que receba um número inteiro como parâmetro e imprima o mês correspondente. Por exemplo, 2 corresponde a fevereiro. O procedimento deve retornar uma mensagem de erro caso o valor recebido não faça sentido. Gere um algoritmo que leia um valor inteiro e chame o procedimento implementado.

2. Escreva uma função que receba como argumento um ano qualquer e mostre qual dia e mês que ocorreu o feriado da Páscoa. O algoritmo para encontrar tal data segue abaixo:

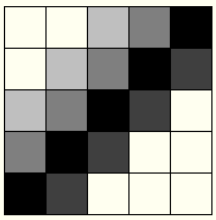
```
A=Ano%19
B=Ano/100
C=Ano%100
D=B/4
E=B%4
F=(B+8) / 25
G=(B-f+1) / 3
H=(19*A+B-D-G+15) % 30
I=C/4
K=C%4
L=(32+2*E+2*I-H-K) % 7
M=(A+11*H+22*L) / 451
MÊS = (H+L-7*M+114) / 31 [3=Março, 4=Abril]
Dia=((H+L-7*M+114) % 31) + 1
```

3. Faça um procedimento que imprima o cabeçalho de um relatório recebendo como parâmetro o nome de um aluno em uma cadeia de caracteres (string ou char). O cabeçalho deverá ter a seguinte forma:

```
==========================================
IFMG - Instituto Federal de Minas Gerais
Campus Sabará
Disciplina de Laboratório de Programação I
Nome: Fulano de Tal
==========================================
```

4. Implemente uma função que receba como parâmetro dois valores **A e B** e calcule o valor de $\mathbf{A^B}$**.** Sua subrotina deve se chamar **pot** e não poderá conter a função **pow.**

5. Para representar imagens a preto-e-branco (e com tons intermédios de cinzento) basta associar a cada pixel um valor numérico numa escala de tons. Se quisermos, por exemplo, uma escala com 256 tons de cinzento (do preto absoluto ao branco absoluto) podemos associar a cada pixel um valor inteiro entre 0 e 255. Convenciona-se que o valor 0 corresponde ao preto e 255 corresponde ao branco; os valores intermédios correspondem aos tons de cinzentos. Exemplo:



```
int imagem[5][5] = {
{ 255, 255, 127, 50, 0 },
{ 255, 127, 50, 0, 25  },
{ 127, 50, 0, 25, 255 },
{ 50, 0, 25, 255, 255 },
{ 0, 25, 255, 255, 255 }
}
```

Figura: Imagem em cizentos e representação

Para representar imagens a cores podemos decompor cada pixel em três componentes de côr: vermelho, verde e azul, designadas pelas suas iniciais em inglês R,G,B (``red", ``green" e ``blue"). Associamos a cada pixel três valores inteiros entre 0 e 255, cada qual indicando qual a intensidade de vermelho, verde e azul nesse ponto da imagem. Nota-se que neste esquema de côr, um pixel branco tem RGB = (255, 255, 255) enquanto um pixel negro tem RGB = (0, 0, 0). Faça:

a) Sabendo que a imagem em escala de cinza pode ser obtida pela média aritmética simples das três componentes de cor, implemente um procedimento que faça tal conversão.

b) Implemente um procedimento que converta a imagem em escala de cinza para seu o negativo, ou seja, a imagem no complemento de 255.

c) Implemente um procedimento que converta a imagem em escala de cinza para seu o poster, ou seja, se pixel for maior que 127, então pixel recebe 255; senão pixel recebe zero.

d) Implemente também a função main para testar os procedimentos anteriores.

6. Criar uma função (não recursiva) que calcule e retorne o valor do fatorial de um número natural. A função deve retornar -1 caso não seja possível calcular o valor do fatorial. Escreva também um algoritmo para testar tal função.

7. Crie uma função que dados os valores de N e P, calcule o número de arranjos e o número de combinações dos N elementos agrupados P a P.

$$A_p^n = \frac{n!}{(n-p)!} \qquad C_p^n = \frac{n!}{p!(n-p)!}$$

Caso não seja possível calcular tal combinação ou arranjo, a função deve retornar -1.

8. Escreva um programa que calcule o valor do coseno de X através de 20 termos da série abaixo:

$$1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \ldots$$

Observações: O valor de x será lido em graus; Deve ser implementado em funções independentes a conversão de x em graus para radianos, o cálculo do fatorial e o cálculo das potências.

9. Implemente uma função soma divisores que recebe como parâmetro um número inteiro e retorna a soma dos divisores do mesmo. Utilizando essa função, faça um programa que identifique se um determinado número digitado pelo usuário é ou não perfeito (um número é dito perfeito quando a soma dos seus divisores é igual ao próprio número. Ex: 6 : 1 + 2 + 3).

10. Uma boa função para geração de números aleatórios deve gerar valores com igual probabilidade, i.e., se eu quero gerar 100 valores entre 1 e 10, o número de vezes que cada número é gerado deve ser próximo de 10. Nesse exercício queremos testar a qualidade da função **rand( )**. Para isso, implemente um algoritmo que contenha uma sub-rotina gere 10000 números aleatórios entre 0 (inclusive) e 99

(inclusive) e conte quantas vezes cada número é gerado. Imprima a diferença entre a maior e a menor contagem. Exemplo: se o número 83 foi aquele que mais vezes foi gerado, com 108 gerações, e o número 12 foi aquele que menos vezes foi gerado, com 93 gerações, então seu programa deve imprimir 108−93 = 15. **Dica:** crie um vetor de inteiros de 100 posições para armazenar quantas vezes cada número (entre 0 e 99) é gerado. Exemplo: se você gerar o número 88, faça vetor_ contagem[88]++;. Além da diferença entre o maior e menor, calcule também o desvio padrão da amostra gerada.

11. Faça um procedimento que recebe como parâmetro um vetor qualquer e ordene o mesmo crescentemente. Faça também um algoritmo para testar a função feita.

12. Faça uma função booleana que receba como parâmetros um vetor e um valor **K.** A função deve pesquisar se o valor K está presente no vetor e retornar **true**, caso verdadeiro e **false,** caso falso. Implemente também um programa para testar tal função.

13. Os elementos a_{ij} de uma matriz $A_{n \times n}$ representam os custo de transporte da cidade *i* para a cidade *j*. O caminhão de uma empresa de logística deve passar exatamente uma vez por cada uma da **n** cidades. Sabendo que, uma rota é representada por um vetor de ***n*** posições em que cada posição representa a cidade visitada, calcule o custo da rota de um caminhão da empresa. Note que, a rota e a matriz de custo devem ser fornecidas pelo usuário.
    Faça uma função que calcule o valor da rota de um determinado caminhão recebendo como parâmetros um vetor que armazena a rota realizada e o número de cidades presentes na mesma. **Observação: defina sua matriz de custos como uma variável global!**
    Exemplo: Considere um exemplo com 4 cidades em que os custos de transporte entre as mesmas são representados pela matriz de custo M abaixo. Considere ainda uma rota denotada pelo vetor = [4 2 1 3] significando que o caminhão parte da cidade 4 com destino à cidade 2; sai de 2 e vai pra 1; e por fim parte da cidade 1 para a cidade 3. Portanto, o custo de transporte dessa rota é: M[4][2] + M[2][1] + M[1][3] = 19 + 15 + 15 = 49 unidades.

    Matriz de custos M = [0 10 15 20
    15 0 25 18
    12 14 0 20
    20 19 35 0]

14. Implemente as seguintes funções:

    a. preencheVetor: preenche um vetor de inteiros de tamanho N.

    b. imprimeVetor: imprimi o conteúdo de um vetor qualquer;

    c. intercalaVetores: recebe dois vetores A e B, o tamanho de A (n), o tamanho de B (m) e produzir o vetor C formado pela intercalação dos vetores A e B, C = {a0, b0, a1, b1, a2, b2, a3, b3...};

    d. valoresComuns: faça uma subrotina que receba dois vetores A e B e gere um novo vetor C contendo apenas os elementos que estão tanto em A quanto em B.

e. semRepeticao: faça uma subrotina que receba dois vetores A e B e gere um novo vetor C conteúdo todos os elementos que estão em A e B sem elementos repetidos.

f. Construa um programa usando as funções acima que leia 4 conjuntos de números inteiros A, B, C e D e imprima o conjunto R, tal que R = (AUB)∩(CUD) (interseção entre a união de A com B e a união de C com D).

15. Uma floricultura conhecedora de sua clientela gostaria de fazer um algoritmo que pudesse controlar sempre um estoque mínimo de determinadas plantas, pois todos os dias, pela manhã, o dono faz novas aquisições. Crie um algoritmo que possua as seguintes opções:

==========

FLORICULTURA MARIASFLOR

==========

1. CADASTRAR NOVA PLANTA
2. RETIRAR PLANTA
3. INSERIR PLANTA
4. IMPRIMIR RELATÓRIO
5. SAIR

==========

O algoritmo deve permitir o cadastro de 50 tipos de plantas. Na opção de cadastro, para cada planta, o algoritmo deve cadastrar um código (inteiro), o nome (string), o estoque ideal e a quantidade atual em estoque. Na opção retirar planta, o sistema deve permitir a retirada do estoque atual de uma planta, dado o código digitado pelo usuário, lembre-se de verificar se o estoque é suficiente para atender o pedido. Para a opção de inserção, seu algoritmo deve atualizar o estoque atual de uma certa planta, dadas as informações de quantidade comprada e do código da planta que são digitadas pelo usuário. Na opção de imprimir relatório, seu algoritmo deve imprimir os nomes, os estoques atuais e as quantidades a serem compradas das plantas que possuem estoque abaixo do ideal. **Obs.: Você deverá utilizar sub-rotinas para cada funcionalidade, tal como no exercício passado em sala.**

16. EXERCÍCIO SIMULAÇÃO BANCÁRIA PASSADO EM SALA.

17. Uma empresa de possui ônibus com 48 lugares (24 nas janelas e 24 no corredor). Faça um programa que utilize uma matriz para controlar as poltronas ocupadas no corredor e janela. Considere que 0 representa poltrona desocupada e 1, poltrona ocupada. Inicialmente, todas as poltronas estarão livres. Depois disso, o programa deverá apresentar as seguintes opções:

==========

MARIAS UNIDAS

==========

a. Vender passagem
b. Mostrar mapa de ocupação do ônibus
c. Encerrar

==========

- Quando a opção escolhida for a letra ‘a’, deve-se verificar se o ônibus ainda possui poltronas a serem vendidas. Caso haja poltrona vazia, deve ser perguntado ao usuário qual poltrona o mesmo deseja ocupar. O programa deverá, então dar uma das seguintes mensagens: 1) Venda efetivada – se a poltrona solicitada estiver livre, marcando-a como ocupada; 2) Poltrona ocupada – se a poltrona solicitada não estiver disponível para venda.

- Quando a opção escolhida for letra ‘b’, deverá ser mostrada uma listagem separando as poltronas da janela e corredor, o número da poltrona e se está livre ou ocupada.
- Quando for escolhida a opção letra ‘c’, a execução do programa deverá ser finalizada.

Obs.: Você deverá utilizar sub-rotinas para cada funcionalidade, tal como no exercício passado em sala.

Exercícios Recursividade

18. Um problema típico em ciência da computação consiste em converter um número da sua forma decimal para a forma binária. Por exemplo, o número 12 tem a sua representação binária igual a 1100. A forma mais simples de fazer isso é dividir o número sucessivamente por 2, onde o resto da i-ésima divisão vai ser o dígito i do número binário (da direita para a esquerda). Faça uma função recursiva chamada dec2bin, que converta um número decimal em binário.

19. Faça uma função recursiva que imprima a série de Fibonacci:

$$Fib(n) = \begin{cases} 1 & \text{se} \quad n = 1 \vee n = 2 \\ Fib(n-1) + Fib(n-2) & \text{se} \quad n > 2 \end{cases}$$

20. Faça uma função recursiva **comb(n, k)** que retorne o número de grupos distintos que podem ser formados com **k** pessoas a partir de **n** pessoas.

$$Comb(n,k) = \begin{cases} n & \text{se} \quad k = 1 \\ 1 & \text{se} \quad k = n \\ Comb(n-1,k-1) + Comb(n-1,k) & \text{se} \quad 1 < k < n \end{cases}$$

21. Implemente uma função recursiva que calcule a função de Ackerman para dois inteiros positivos, conforme definição abaixo:

$$A(m,n) = \begin{cases} n+1 & \text{se } m = 0 \\ A(m-1,1) & \text{se } m > 0 \text{ e } n = 0 \\ A(m-1, A(m,n-1)) & \text{se } m > 0 \text{ e } n > 0. \end{cases}$$

22. O máximo divisor comum (MDC) de dois números inteiros x e y pode ser calculado usando-se uma definição recursiva: Se y = 0, então x; senão mdc(y, x%y).

23. Crie uma função recursiva que calcule o fatorial de um valor inteiro positivo qualquer passado por parâmetro.

24. Vários algoritmos em computação usam a técnica de “Dividir para Conquistar”: basicamente eles fazem alguma operação sobre todos os dados, e depois dividem o problema em sub-problemas menores, repetindo a operação. Uma equação de recorrência típica para esse tipo de algoritmo é mostrada abaixo. Resolva essa equação de recorrência.

```
T(n) = 2T(n/2) + n;
T(1) = 1;
```